INDEXING AND PARTITIONING SCHEMES FOR DISTRIBUTED TENSOR COMPUTING WITH APPLICATION TO MULTIPLE SEQUENCE ALIGNMENT

Manal Ezzat A. HELAL

A thesis submitted in fulfilment of the requirements
for the award of the degree of Doctor of Philosophy

Computer Science and Engineering
Faculty of Engineering
University of New South Wales

Sydney, Australia

August 2009

COPYRIGHT STATEMENT

Signed …….....

Date …….....

AUTHENTICITY STATEMENT

'I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.'

Signed …….....

Date …….....

Abstract

This thesis investigates indexing and partitioning schemes for high dimensional scientific computational problems. Building on the foundation offered by Mathematics of Arrays (MoA) for tensor-based computation, the ultimate contribution of the thesis is a unified partitioning scheme that works invariant of the dataset dimension and shape. Consequently, portability is ensured between different high performance machines, cluster architectures, and potentially computational grids.

The Multiple Sequence Alignment (MSA) problem in computational biology has an optimal dynamic programming based solution, but it becomes computationally infeasible as its dimensionality (the number of sequences) increases. Even sub-optimal approximations may be unmanageable for more than eight sequences. Furthermore, no existing MSA algorithms have been formulated in a manner invariant over the number of sequences.

This thesis presents an optimal distributed MSA method based on MoA. The latter offers a set of constructs that help represent multidimensional arrays in memory in a linear, concise and efficient way. Using MoA allows the partitioning of the dynamic programming algorithm to be expressed independently of dimension. MSA is the highest dimensional scientific problem considered for MoA-based partitioning to date.

Two partitioning schemes are presented: the first is a master/slave approach which is based on both master/slave scheduling and slave/slave coupling. The second approach is a peer-to-peer design, in which the scheduling and dependency communication are calculated independently by each process, with no need for a master scheduler.

A search space reduction technique is introduced to cater for the exponential expansion as the problem dimensionality increases. This technique relies on defining a hyper-diagonal through the tensor space, and choosing a band of neighbouring partitions around the diagonal to score. In contrast, other sub-optimal methods in the literature only consider projections on the surface of the hyper-cube.

The resulting massively parallel design produces a scalable solution that has been implemented on high performance machines and cluster architectures. Experimental results for these implementations are presented for both simulated and real datasets. Comparisons

between the reduced search space technique of this thesis with other sub-optimal methods for the MSA problem are presented.

Acknowledgements

I owe a deep debt of gratitude for the efforts of Dr. Lenore Mullin, from the National Science Foundation, VA, USA, for her continuous support of my research over the last 10 years. Dr. Mullin supported my implementation of her Mathematics of Arrays formalizations for my Master's thesis for basic image and video processing, and also the application of Mathematics of Arrays in this thesis. She supported my participation in the Tensor-Computing Workshop held in the NSF, February 2009, where I learned more about open problems in the field. She also provided me with access to the computer systems at the Computational Centre for Nanotechnology Innovations at Rensselaer Polytechnic Institute.

I gratefully thank Dr. John Potter and Dr. Vitali Sintchenko for their assistance with the completion of this thesis and its final shape. Their meticulous reading and supervision has raised the quality of my work and related it to real biological problems. I have learned, and continue to learn, by working with them.

I thank Dr. Hossam El-Gindy and Dr. Bruno Gaeta for the original formulation of the problem addressed in this thesis. I also received a lot of constructive criticism from Dr. Mike Bain in my annual reviews throughout the years of the PhD candidature. Similar useful criticism was given to me from Dr. Albert Zomaya from University of Sydney. I also received a lot of useful suggestions from Dr. Kai Engelhardt in regards to parallel implementation and MPI.

There are a number of institutions that assisted with computational facilities for the work presented in this thesis. The Australian National Computational Infrastructure (NCI) provided me with grants to use their facilities in my experiments. The continuous support from Dr. Margret Khan and David Singleton has made my experience using the NCI HPC facilities a rewarding educational experience. I also received grants from the Australian Centre for Advanced Computing and Communications to use their computers clusters. In that connection, I would like to acknowledge the courses administered by Dr. Joachim Mai and his helpful replies by emails, and the support from the UNSW representative Dr. Robert Bursill. The funding received from The University of Sydney for a Sun Fire X2200 M2 Server with 2xAMD Opteron quad core, has enabled further development of this thesis implementation and is gratefully acknowledged.

From the American University in Cairo, I would like to thank Dr. Soha Zaghloul for reading my thesis fully and providing very useful remarks on all parts of it. I would like to acknowledge my previous supervisor Dr. Ahmed Sameh for his continuous support through useful comments and remarks, and Dr. Hoda Hosny for her feedback and advice all the way. Intelligent discussions with Dr. Lamia Youseff have also expanded the application and future trends of this work.

I would like to thank all my colleagues in the Centre for Infectious Disease and Microbiology (CIDM) – Westmead Hospital, Western Clinical School, Sydney University for the educational seminars where I learned various applications and future projects ideas. Everyone in CIDM has answered all my questions and provided me with an excellent interdisciplinary research environment that helped me in writing and evaluating my work. In particular I acknowledge Dr. Matthew O' Sullivan, Dr. Kong Fanrong, Dr. Anna Lau, Dr. Sharon Chen, Dr. Neisha Jeoffreys, Prof. John Iredell, Prof. Dominic Dwyer, Prof. Lyn Gilbert, and Prof. Tania Sorrell.

Last but not least, this thesis would not have reached completion without the continuous support from the IT unit in CSE/University of NSW and Western Clinical School/University of Sydney, specifically the support of Robert Doran, Sara Hakami, Chris Petrov, Simon Garrod, and Ian Magee.

Table of Contents

Publications Resulting from this Thesis

1. Helal, M., Mullin, L., Gaeta, B., El-Gindy, H. Multiple Sequence Alignment Using Massively Parallel Mathematics of Arrays. Poster presentation presented at the APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch, Royal Pines Resort, Gold Coast, Australia, September 2005.
2. Helal, M., Mullin, L., Gaeta, B., El-Gindy, H. Multiple Sequence Alignment Using Massively Parallel Mathematics of Arrays. Poster presentation presented at BioInfoSummer 2005 - ICE-EM Summer Symposium in Bioinformatics, The Australian National University, Canberra, Australia, November 2005.
3. Helal, M., Mullin, L., El-Gindy, H., Gaeta, B. Optimal Parallel Solution for Multiple Sequence Alignment Using Mathematics of Arrays. Poster presentation presented at Bioinformatics Australia 2006 "Connecting Australian Bioinformatics" 21-22 November 2006, Sydney Convention and Exhibition Centre, Darling Harbour, Sydney, NSW, November 2006.
4. Helal, M., Mullin, L.M., Gaeta, B., El-Gindy, H. Multiple sequence alignment using massively parallel mathematics of arrays. In proceedings of the International Conference on High Performance Computing, Networking and Communication Systems (HPCNCS- 07), Orlando, FL. USA, 2007. pp. 120-7.
5. Helal M., El-Gindy, H., Gaeta, G., Sintchenko, V. High Performance Multiple Sequence Alignment Algorithms for Comparison of Microbial Genomes. In proceedings of the 19th International Conference on Genome Informatics - GIW 2008. - Gold Coast, 2008.
6. Helal, M., El-Gindy, H., Mullin, L., Gaeta, B. Parallelizing Optimal Multiple Sequence Alignment by Dynamic Programming. In Proceedings of the International Symposium on Advances in Parallel and Distributed Computing Techniques (APDCT-08) held in conjunction with 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA-08). Sydney, Australia, December 2008. pp. 669-674. ISBN: 978-0-7695-3471-8.

7. [1] Helal, M., Sintchenko, V. Dynamic programming algorithms for discovery of antibiotic resistance in microbial genomes, In Proceedings of the Health Informatics Conference (HIC-09). - Canberra, Australia, August 2009.
8. Helal, M., Mullin, L., Potter, J., Sintchenko, V. Search Space Reduction Technique for Distributed Multiple Sequence Alignment. In Proceedings of the 6th IFIP International Conference on Network and Parallel Computing (NPC 2009). Gold Coast, Queensland, Australia, October 2009.

[1] This paper received the Branko Cesnik Awards for HIC 2009 Best Student Scientific Paper.

List of Figures

List of Tables

List of Equations

Dedication

I dedicate this thesis and all the hard work and time and effort invested to achieve it to my parents. To all people, parents are the ones who primarily gave them their lives after God's will. However, only some understand what that means. Some are lucky enough to pay back their parents and make their main objective in life is to make their parents happy and make all their dreams come true. The others who never appreciated their parents did that primarily because they never appreciated their lives in the first place.

To me, my parents were always pushing me forward, and did all they can to shield me from this life's hardships; and teach me how to rise high and above every challenge. They filled my life with what really matters; with what I might leave behind and at the same time take with me for eternity. They taught me how to value every moment I was given on earth, how to fully live it, invest it, and enjoy its returns without regrets. They were simply the best.

Chapter 1: Introduction

Advances in multi-core and high performance computing (HPC) technologies have enabled solutions to problems that were previously considered intractable or computationally prohibitive. Developing efficient distributed and parallel applications using these technologies is more complex than sequential application development. The basic difference is the ability to run on more than one processing element, whether in the same computer for parallel applications, or over a network for distributed applications. The complications stem from design, implementation, debugging, testing, and deployment issues. The basic design complication is based on data partitioning of a sequential solution to run in parallel, which is not a trivial task. Communication between the data partitions is required and optimizing the communication overhead versus employing more processors is an open area of research.

Bioinformatics and computational biology are sometimes referred to as comparative genomics. This field is inter-disciplinary and requires various skills to collect, store, handle analyse, interpret and spread biological information. Because of the enormous quantities of genomic and proteomic data, and extensive computation that is needed, high performance computers and innovative software solutions are required to manage the complexity (ZOMAYA, Albert, 2006). Consequently, this thesis investigates the suitability of a tensor indexing scheme using Mathematics of Arrays (MoA) and Ψ-Calculus, for representing high dimensional problems with associated data partitioning schemes and dependency modelling suitable for parallel processing.

Section 1 introduces the need for such research, as high dimensional problems are becoming more approachable and the need for optimal solutions is increasing. The aims and the motivation for this work are provided in section 2. The contributions to the literature and the impact of the work of this thesis are discussed in section 3. The research method followed is described in section 4, while the thesis organization is explained in section 5.

1.1 Dimensionality Curse in High Dimensional Scientific Problems

The dimensionality curse is characterized as the exponential growth in space as extra dimensions are added to the space (BELLMAN, RE, 1961). In other words, in computational terms, the number of grid-points where a solution is computed normally grows exponentially in the number of dimensions for the underlying problem. Hence, high-

dimensional problems often require a large amount of data storage and extreme computational capacities. Dealing with high dimensional scientific problems is currently an active area of research. Most current research is directed at the design and analysis of approximation algorithms for computationally hard problems in combinatorial optimization.

Example problems include, but are not limited to, clustering algorithms, such as the use of high-dimensional Singular Value Decompositions (SVDs), and nearest neighbour search, an example of which is the design of a data structure that finds the closest point—taken from a large collection of possibly very high-dimensional data points—to an arbitrary input. The applications of these problems vary in different fields such as: computational statistics, pattern and facial recognition, multimedia information retrieval, vector compression, the description of queuing networks, routing and flow control in communication networks, partial differential equations, high dimensional database mining and query processing. Financial mathematics is also an application area for high dimensional computation, especially non-linear and/or high-dimensional optimization problems that are commonly encountered in portfolio allocation, asset pricing, various models for the financial derivatives pricing and contract theory. Other application areas include quantum physics, global optimization, genetic analysis, reaction mechanisms in molecular biology, and visco-elasticity in polymer fluids.

Furthermore, homogenizations with multiple scales as well as stochastic elliptic equations result in high-dimensional PDEs. Finally, in quantum mechanics, the Schrödinger equation is naturally high-dimensional and thus Heisenberg's equivalent, which resulted in Matrix Mechanics, is relevant to these endeavours. All these problems raise theoretical mathematical questions. While many theoretical and numerical advances are recently realized in the field of optimization techniques such as the Forward Backward Stochastic Differential Equations, numerical methods, PDE's, Monte Carlo simulation, quantitative finance, Malliavin Calculus, and deterministic optimization algorithms.

More open research areas are still active, such as employing distributed high dimensional computation for these problems. This is the main scope of this thesis. Since high-dimensional mathematical models often lead to large-scale computations, especially with the current advancement in high performance machines, grid computing and even highly capable stand-alone machines, more investigations are necessary to achieve better solutions for high-dimensional problems.

1.2 Thesis Motivations, Aims and Objectives

Underlying the research for this thesis is the advancement in high performance computers (HPCs), grid computing, and high capability multi-threaded stand-alone machines. The key motivation is the need for more near-optimal solutions for high dimensional scientific problems. Given the above need of addressing high-dimensional scientific computing problems, and the availability of high capacity and capability computers, there is a need for further investigations for parallel algorithms addressing high-dimensional problems. On the other side, Conformal Computing methods (MULLIN, LR, Raynolds, JE, 2008) (as further explained in Chapter 2) promised to address these problems and provide a computing paradigm suitable for parallelization. Conformal Computing is a systematic, algebraically based, design methodology for efficient implementation of computer programs optimized over multiple levels of the processor/memory and network hierarchy. Hence, this research was motivated by the need for better techniques for parallelizing solutions to high dimensional problems, and the Multiple Sequence Alignment (MSA) problem was identified as an ideal candidate to be transformed into Conformal Computing methods. This thesis is focusing on the use of Conformal Computing methods for data partitioning for concurrent/distributed applications. It is not addressing optimisation of data layout for memory caches/memory hierarchy

A high dimensional problem like MSA in computational biology is a suitable problem representing the class of problems that suffer from the curse of dimensionality. The MSA is a very important problem that is addressed by a number of algorithms from several statistical and algorithmic points of view (TOMPA, M, 2000). Further details on MSA existing methods are described in Chapter 2. MSA methods can be classified as simultaneous or progressive/iterative. Simultaneous methods are optimal but with high computational complexity, whereas progressive and iterative methods rely on heuristics that attempt to find suboptimal solutions quickly. Simultaneous methods include MSA (LIPMAN, DJ, Altschul, SF, Kececioglu, JD, 1989), DCA (STOYE, J, Moulton, V, Dress, AWM, 1997), and an integer programming method to optimize a Maximum Weight Trace (MWT) objective function (KECECIOGLU, J, 1993). Attempts to parallelize sequence comparisons are limited to pair-wise alignments (DRIGA, AR, 2002). Due to the various definitions of optimal alignments with regards to mathematical scoring vs. biological meanings, further research is required to optimize the quality of the alignment in various scenarios.

The thesis objective is to employ the HPC technology to optimize MSA outcomes. The presented work aims to:

1. Extend the dynamic programming algorithm for higher dimensions using tensor indexing schemes.
2. Partition the scoring of the tensor for parallel/distributed processing.
3. Reduce the search space for global alignments
4. Provide execution parameters to accommodate for various HPC machine architectures, and various scoring schemes for different data sets.

1.3 Contributions and Impact

The main contribution of this work is a unified partitioning scheme for high dimensional computational problems for distribution over parallel processors. Nine specific contributions of the thesis are as follows:

1. An ANSI C tensor representation library, containing data structures with n-dimensional array manipulation constructs based on the MoA and Ψ-Calculus. The original MoA and Ψ-Calculus libraries were implemented as parsers that convert n-dimensional arrays code to the tensor notation as expressed in (MULLIN, LR, 1988). The latest published in the literature is presented in (LANDRY, W, 2003) with a survey of other methods existing to date. Landry surveys the limitations of Blitz, POOMA and template based libraries and presents a notation based on Einstein summation notation and expression templates, template meta-programs, and traits. This library improved the FORTRAN type tensor representation with an application to General Relativity expression templates. However, the programmer still needs to include manual loops to fully represent the tensor for different problems, in addition to handling the performance penalties that are sometimes very severe, and portability issues, across different compilers and hardware. Landry's survey missed the original MoA implementation in (HELAL, M, 2001), which is used as the foundation of the added partitioning methods presented in this work. There is also the MatLab Tensor toolbox presented in (BADER, BW, KOLDA, TG, 2007) that implemented a sparse tensor storage and decompositions that is efficient for various mathematical operations using models such as Tucker (TUCKER, LR, 1966), CANDECOMP and PARAFAC as defined in (HARSHMAN, RA, 1970) and (CARROLL, JD, Chang, J, 1970). In correspondence with CANDECOMP/PARAFAC models, the Kruskal format stores a tensor as the sum of rank-

1 tensors, which over simplifies the tensor structure and doesn't encapsulate its geometrical and algebraic properties away from the pair-wise views of the dimensions. The basis for the tensor representation library developed for this thesis is introduced in Chapter 2 and further implementation details are presented in Chapter 3.

2. Two unified partitioning schemes for high dimensional computational problems for distribution over parallel processors. These are defined as follows:

Master/slave partitioning scheme that divide the tensor space cubically into partitions on equal cubic perimeter distance from the origin as discussed in Chapter 3.

Peer-to-peer more sensitive partitioning scheme that diagonalizes the tensor space logically based on total distance from the origin as discussed in Chapter 4.

3. A high dimensional dependency analysis model using neighbourhood functions as discussed in Chapters 3 and 4.

4. A simple load balancing technique based on data neighbourhood considerations as discussed in Chapter 4.

5. A generic granularity parallel processing model based on computation vs. communication optimizations as discussed in Chapter 5 in the optimizations section.

6. An algebraic analysis of a high dimensional diagonalization of a tensor, and distance between vectors as discussed in Chapter 5.

7. A dynamic programming MSA scoring function that is based on dimension invariant index neighbouring transformations as discussed in Chapter 3.

8. An optimal MSA tool based on dynamic programming and simultaneous alignment methods, called "mmDst" in the remaining of the thesis; this name stands for Distributed MSA based on MoA as discussed in Chapter 4.

9. MSA heuristic techniques that are based on simultaneous alignment with no bias to sequence ordering, and no need for prior knowledge about the input sequences structure or similarities. This is discussed in Chapter 5.

The impact of this work is listed in the following points:

- The first true library to create and manipulate arrays invariant of shape and dimension. Previous work created the memory allocations only for the multidimensional arrays, and/or provided basic indexing methods. The presented MoA library provides various indexing, partitioning, retrieval of neighbours or distant neighbours given a distance or a stride, among other functions. This requires the problem statement itself to be expressed

in MoA notation and the solution to be programmed using MoA data structures and Ψ-Calculus, rather than parsing existing code and generating optimized MoA code.

- The first distributed solution for a high dimensional problem like MSA based on the MoA library, that simultaneously aligns the variable number of sequences in the data set, with their variable lengths. The solution results don't include any bias to the order of the sequences in the input set, and is an exact solution.
- The dependency analysis modelling invariant of dimension and shape is the first true dimension-invariant data alignment scheme. Other methods only work for low dimensionality and fail as the dimension grows (PRAKASH, SR, 1998).
- The distribution over processors for high dimensional problems is first investigated in this thesis. The literature doesn't include any work about reducing the dimensionality curse through parallel processing.

1.4 Research Method

This research started with the hypothesis that Conformal Computing methods provide a unified partitioning scheme invariant of data dimensionality and shape, thereby allowing simpler distributed processing with load balancing on multiple processors. Further experimentation was needed to either prove or negate this hypothesis. The Multiple Sequence Alignment in Computational Biology was chosen as a high dimensional problem. The optimal dynamic programming algorithm was sequentially transformed to the conformal computing indexing methods, which is the only method that defines the high dimensionality of the problem as is. Then dependency analysis took place. Two models of distributions over processors were implemented: A master/slave approach was formulated to cubically partition the tensor spaces on equal cubical perimeter distance from the origin, and its performance was unsatisfactory. An optimization of a Peer-to-Peer Model was designed to partition the tensor space diagonally on equal distances from the origin, and achieved a speed up to 4 times in execution time. A search space reduction method was introduced that is based on defining a hyper-diagonal line through the tensor space, and measures an absolute distance to be the upper and lower bounds navigating all dimensions from the middle partition index in the corresponding wave. Partitions that fall within this band will be scored only and the rest will be ignored. Then the effects of search space reduction on optimality of the scores were analysed. Further simulations and experiments on High Performance

machines were conducted, and results were collected and analysed to identify optimization chances and the conclusions and projections were drawn.

An iterative experimental approach was followed based on the experimental results and the expected results, trying to come forward from one level of performance to a higher level. Then execution time minimization was done by linear algebra methods to find optimal partition size and number of processors. A full disclosure was addressed, by designing the test cases to cover all scalability issues: processors scalability, data size scalability over all processors, and partition size to fit in one processor scalability.

1.5 Thesis Organization

The first Chapter is an introduction to the ideas that motivated this research. The next Chapter provides the literature review of all methods used in the presented solutions, the investigated problem, the current existing solutions and their attributes. The third Chapter describes the methods used in the master/slave approach presented. The fourth Chapter presents an optimized peer-to-peer solution for the MSA problem. The fifth Chapter discusses the implemented search space reduction methods and the optimization techniques. The sixth Chapter details the results of two experiments conducted on real biological sequences, in comparison to the existing methods. The seventh Chapter is the conclusion and the future work that can be investigated based on the results achieved in this work.

Chapter 2: Literature Review and Problem Definition

In this Chapter, a detailed literature review is presented of all concepts and methods used in the approaches and solutions used in this thesis. Because the work in this thesis overlaps with two areas of Computing Science and with an application in an interdisciplinary field like Computational Biology, a huge literature had to be reviewed for a sufficient background and coverage of basic concepts and alternative methods. The main Computing Science concepts are distributed and parallel processing on one side and multi-dimensional arrays (tensors) on the other. The first section describes distributed processing as well as high performance machines architectures and development models. The second section introduces array functions, methods and history. The third section describes high dimensional computing, dimensionality reduction methods and their effects on the solution optimality, and high dimensional data partitioning for parallel processing. The fourth section introduces new advances in computational biology, the multiple sequence alignment problem, existing solutions and their advantages and problems.

2.1 Distributed Processing

Moore's law as introduced in (MOORE, GE, 1965) is expected to continue only through parallel processing, since sequential processing is not providing any further speed-ups due to smaller chip sizes based on the Very Large Scale Integration (VLSI) technology. Alternatively, the increasing demand of the computing power will shift research towards allowing users to exploit massively parallel execution through the aggregation of multiple microchips. Based on the science of silicon photonics, silicon is used as an optical medium to interconnect low-cost microchips to construct what is essentially a single, virtual "macro-chip". Furthermore, massively parallel execution is becoming necessary because of the availability of multi-processor and/or multi-core chips workstations, clusters of commodity workstations, world ranking High Performance Computers (HPC), and the grid and cloud computing. Figure 1.c. shows how the latest development in the HPC as continuously monitored by the Top 500 organization (MEUER, H, Strohmaier, E, Dongarra, J, Simon, H), will keep Moore's law progressing beyond the transistors' size boundaries.

It is widely accepted that the existing transistor density in multi-core CPUs does not necessarily reflect a similar increase in practical computing power, due to the un-parallelised

nature of most applications. The execution time of large-scale computationally intensive applications can be reduced by using multiple processors, given that the code is efficiently parallelized and communication is minimized. The process involves using a large number of processors connected on the same chip, package, system, or over high-speed networks.

This section surveys the state of the art in distributed processing technology. The first subsection describes the technology and its advantages, and available machines. The second subsection describes the different parallel processing hardware architectures. The third subsection presents Amdahl's law and the effectiveness measure of using an extra processor for a given problem. The fourth subsection surveys some concepts to consider while developing for a parallel processing system, such as the communication and scheduling models, and granularity issues.

Figure 1: Moore's Law and Top500 List

2.1.1 Capacity and Capability Computing

As shown in (GRAHAM, SL, Snir, M, Patterson, CA, 2004) capability computing applies maximum processing power in order to solve a large problem in a short period of time. From another point-of-view, capability computing is the ability to solve problems of a magnitude that are not solved before. Examples of such systems are the Department of Energy (DOE) supercomputers such as ASCI White and the recently demonstrated ASC Purple.

Parallel computing is performed at different levels of parallelism: bit-level, instruction-level, data-level, and task-level. Some HPC systems are Shared Memory Multicomputers (SMM). SMMs are constrained by their bus or switch bandwidth to shared memory and their ability to successfully provide cache coherence. As the number of processors connected to the shared memory increases, the performance degrades. Most supercomputers are Massively Parallel Processing (MPP) with Distributed Memory Multicomputers (DMM), which have hundreds and thousands of independent computing nodes with their own memory connected together over a high speed networks. Some of these machines are listed in Appendix B.

In contrast, capacity-based systems are typically cheaper and perform less than capability-based systems on a per-node basis as well as relative to the entire system. Capacity-based systems allow scientists to explore design alternatives that are often needed to prepare for larger-scale runs on capability-based systems. In addition, capacity-based systems typically solve a multitude of smaller problems simultaneously. Systems such as Green Destiny, MegaProto, Orion Multisystems DS-96, and arguably Blue Gene/L fit into this category. The Top 500 list (MEUER, H, Strohmaier, E, Dongarra, J, Simon, H) is updated every six months to include the highest performing capacity machines worldwide according to the LINPACK benchmarks. Based on December 2008 list, the first position is occupied by the IBM Roadrunner BladeCenter QS22/LS21 cluster. It has PowerXCell cpu 8i 3.2 Ghz and Opteron DC 1.8 GHz, and is capabale of 12.8 GFlops, and is connected via Voltaire Infiniband.

2.1.2 High Performance Machines

Parallel computing models are depicted in Figure 2. A clustered computing model, specifically a Beowulf cluster, is the most affordable and available one. The project started in 1993 by NASA. It uses available processors and networks, free operating system (Linux, BSD), standardized programming model (MPI, LAM, and PVM). Beowulf is suitable for embarrassingly parallel applications that can be formulated in a message-passing environment at 1/10th of the cost of commercial high performance machines. The cost of adding the required memory, disk space, memory bandwidth, and disk bandwidth is much lower than that of the commodity high performance machines; it therefore suits the research environment. There are many factors that favour the Beowulf PC cluster parallel computing model; namely, the recent narrow gap between high-end processors and commodity

processors, the availability of Ethernet switches, open-source free operating systems and standard messaging libraries.

Meta-computing is focused on technological and methodological aspects of large computer networks and grids and territorially distributed computer networks. These include internet/intranet as peer-to-peer or centralized solutions. The tightly coupled parallel processors systems are also called multi-processor systems. CPUs share the same memory and I/O resources within a single computer system. These include multiple cores on one die, multiple chips in one package, and multiple packages in one system unit architectures. A vector processor (array processor) is designed to perform operations on multiple data elements simultaneously. These are the basis for supercomputers used for scientific applications, such as the Cell processor for the eight CPUs known as Synergistic Processing Elements (SPE), while the Power 4 architecture is used for the Power Processing Element (PPE). Vectors processors are also used in the video game consoles and computer-graphics hardware like Graphical Processing Units (GPUs).

The experimentation presented in Chapter 6 used the computing clusters that are available through grants from Australian Partnership for Advanced Computing (APAC)[2]. The APAC National Facility is available for use by researchers and postgraduate students at Australian universities and the Commonwealth Scientific and Industrial Research Organisation (CSIRO). The cluster used in the experiments is the ac cluster which is an APAC SGI Altix 3700 Bx2 cluster with 1928 1.6Ghz Itanium2 processors (1920 dedicated to batch jobs) with 16 kbytes (D) + 16 kbytes (I) as L1 cache with 64 bytes cache line, 256 kbytes as L2 cache with 128 bytes cache line, and 6 Mbytes as L3 cache with 128 bytes cache line. The processors are grouped into 30 "partitions" (single system image nodes) of 64 processors each, where 17 processors have 128 Gbytes of memory, 12 processors have 256 Gbytes of memory, and 1 has 384 Gbytes of memory. The total memory of the cluster is 5.6 Tbytes. SGI's NUMAlink4 interconnect both is used within and between partitions providing 3.2 Gbytes/s bidirectional bandwidth per link and less than 2 μs MPI latency. Approximately 30 Tbytes of global storage attached to five SGI InfiniteStorage TP9500 controllers. Storage is

---

[2] This work was supported by an award under the Merit Allocation Scheme on the National Facility of the Australian Partnership for Advanced Computing. (http://nf.apac.edu.au/facilities/ac/hardware.php)

available to all nodes under SGI's CXFS file system with an aggregate bandwidth of more than 2Gbytes/s. The cluster has a usable total of 27+ Tbytes (47+ Tbytes of raw disk) of SGI TP9100 storage local to the partitions and connected to each by two dual port HBA's (~800 MBytes/s). The operating system is based on SUSE SLES9 Linux with an enhanced Linux 2.6 kernel and SGI ProPack4. It offers a total peak speed of over 11Tflops. The ac cluster is ranked 26th in June 2005 Top500 list. The distributed messaging library installed in the SGI cluster is the Intel MPI Library.

Another HPC was available for experimentation through a grant from University of Sydney. It is a SunFire X2200 machine with 2xAMD Opteron quad processors of 2.3 GHz, 512 Kb L2 cache and 2 MB L3 cache on each processor, and 8GB RAM. The AMD64 Opteron architecture is illustrated in Figure 4. The AMD Opteron Quad-Core processors provide Direct Connect Architecture that is designed for optimum multi-threaded application performance. It starts with a native quad-core design — featuring four cores on a single piece of silicon for more efficient data sharing; it then adds an enhanced cache structure and integrated memory controller designed to sustain multi-threaded application throughput. Being in the market since 2003, Direct Connect Architecture is a well-tested and proven architecture. It is exclusively produced by AMD.

Other tests have been done using a grant from AC3 on the Barossa machine, which is a Linux Beowulf cluster with 300 nodes (150 boxes with 2 CPUs), Xeon processors running at 3 GHz, 2 GB RAM per CPU supplied by Dell. The machine is ranked 108 in the November 2003 (MEUER, H, Strohmaier, E, Dongarra, J, Simon, H) Top500 list. There are 16 dedicated nodes, in addition to 3 nodes used for login, compilation and file serving. This leaves 128 nodes available for general computation. Sixteen of these nodes have greater cache (1MB), and some other processor optimization features that suit certain codes such as Gaussian (AC3). The cluster can execute programs requiring up to 256 GBytes of memory. Each processor is capable of a peak speed of about 6 GFlops, or 12 GFlops per node, leading to a theoretical maximum speed of 1.82 Tflops. The machine has around 1 TBytes of networked attached disk storage. Additionally, each node has 36GB of local disk storage.

Furthermore, access to a Blue Gene machine was allowed through a grant from Computational Centre for Nanotechnology Innovation (CCNI) in Rensselaer Polytechnic Institute (RPI). The experiments were conducted on a 16 rack IBM Blue Gene\L machine with 32768 IBM POWER 440 processors each with 1 GB or 512 MB RAM with a total of 12

TB. It has 32 KB per processor as L1 cache, 2 KB per processor as L2 cache, and 4 MB embedded DRAM shared by the processors as L3 cache. It is connected as a 3-dimensional torus with global collectively of 350 mbps (1.5 μs latency). The machine has a peak performance of 5.6 GFLOPS per compute node. The architecture of the machine is shown in Figure 5(ALMASI, G et al, 2002). However, the 512 MB or max 1 GB per node required further optimization on the experimentation code to reduce memory consumption. This is left for future work as mentioned in Chapter 7.

There are also hybrid systems using OpenMP AND MPI and other Reconfigurable Machines (RASCs) such as machines with GPUs, ASICs, and FPGAs. The experiments in this thesis were not done in these kinds of machines, although they provide scalability that parallel algorithms can benefit from.

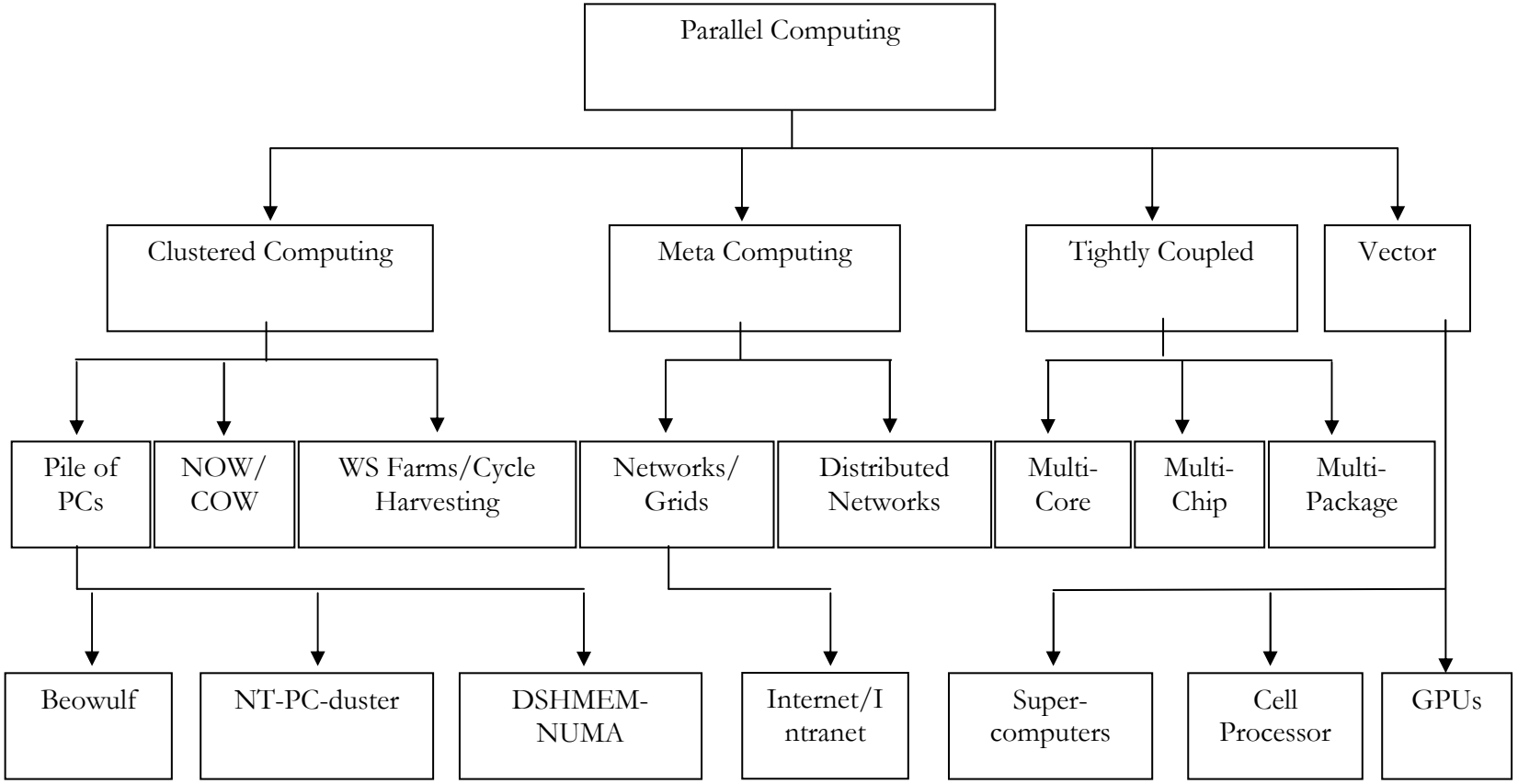


Figure 2 Parallel Computing Architectures

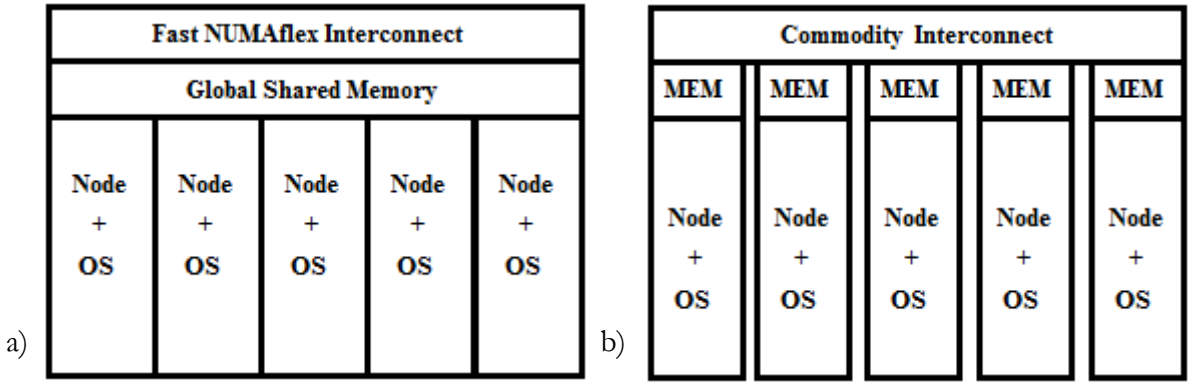
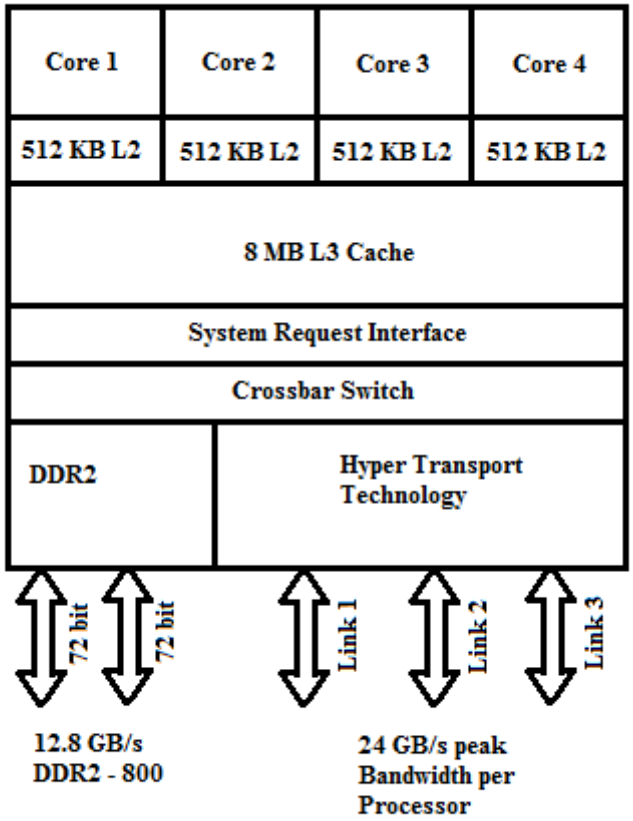
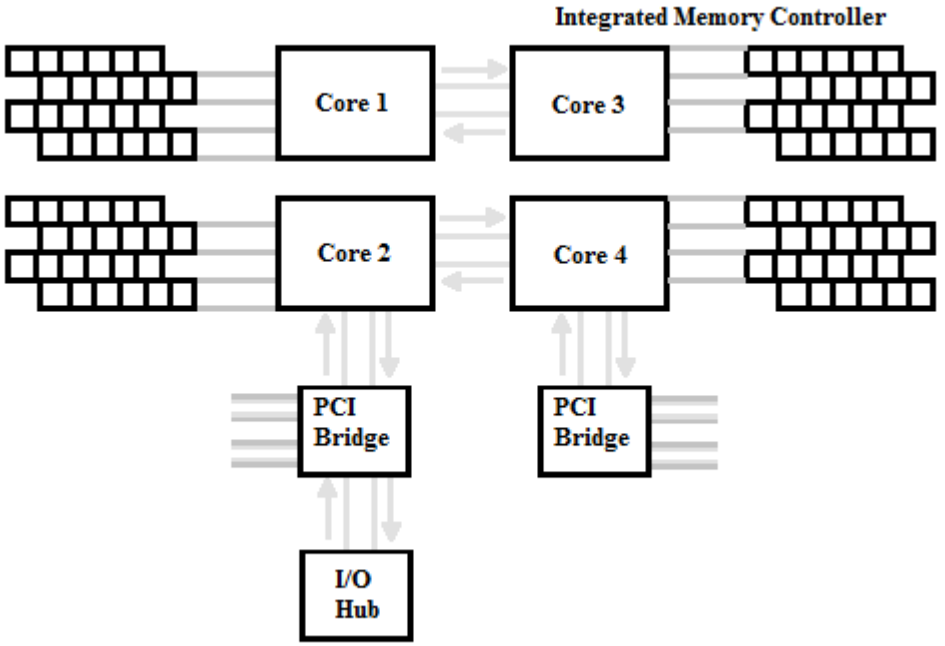


Figure 3: The SGI NUMAflex Global Shared Memory Architecture (a), vs. the Commodity Interconnect (b).



a)



b)

Figure 4: AMD Quad Core Architecture

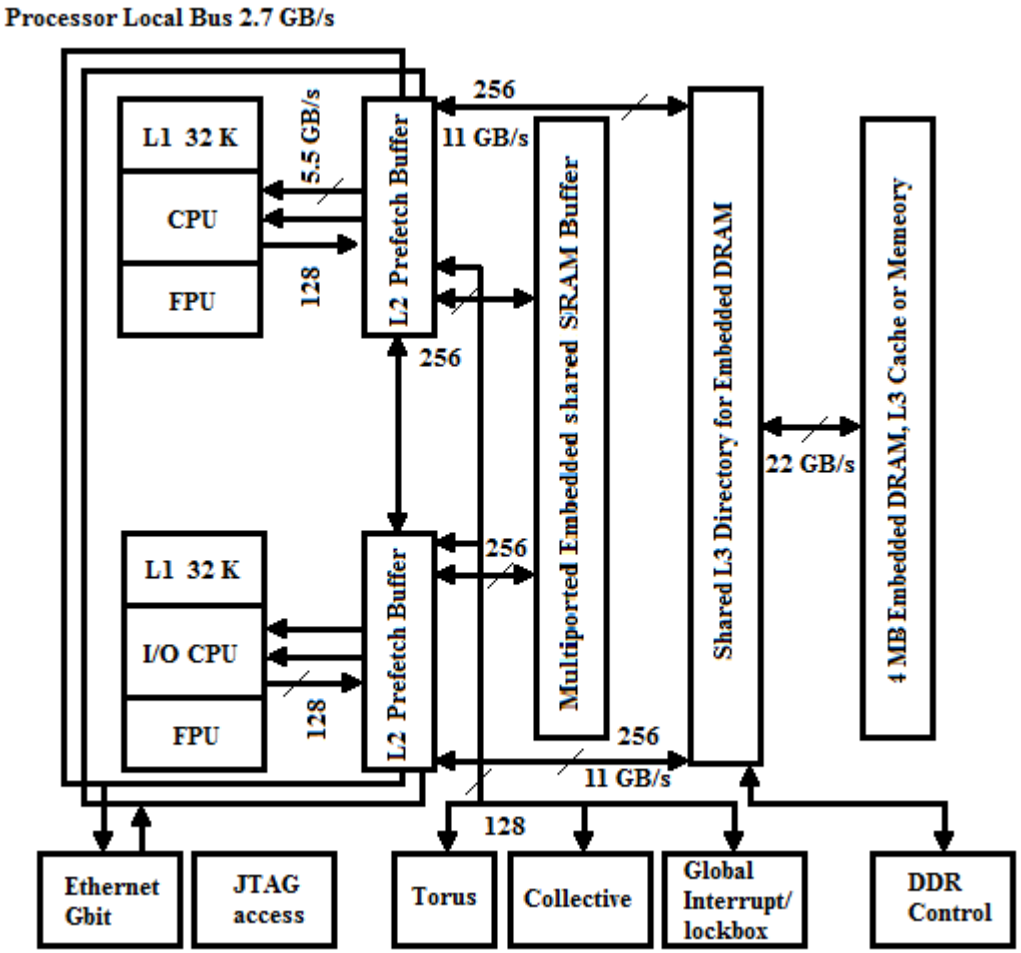


Figure 5: Blue Gene/L Computer (BLC) Chip Architecture.

2.1.3 Amdahl's Law

Amdahl's law describes the speed up gained due to using a parallelized algorithm instead of a sequential alternative (AMDAHL, G, 1967). For example, if ***P*** is the percentage of instructions of a program that can be executed in parallel and can run arbitrarily fast, and ***N*** is the number of processors used, then the speedup of the whole program is as shown in Equation 1. The law shows that no matter how many processors are employed, the minimum execution time cannot be less that the non-parallelizable portion of the algorithm. Figure 6 plots different graphs for the speed up, per extra processor, for ***p*** values of 95%, 90%, 75%, and 50%. This shows that for low ***P*** values, such as 50%, no speedup is gained after only 16 processors than for higher ***P*** values.

A processor scalability experiment is conducted in Chapter 4, and the implementation showed good near-linear scalability. This is due to the fact that the scoring program is efficiently parallelized with the exception of the communication of dependencies. Complexity analysis and generic optimization of the parallelized portion (independent simultaneous partitions) vs. sequential portion (communication of dependencies) is provided. The trace back routine as explained in section 3.5 is implemented sequentially to assemble the alignment from the distributed partitions in the memory of the used processors.

2.1.4 Parallel Processing Aspects

This subsection discusses various aspects to be considered when developing for a parallel processing system. Process communication is the first decision to be made and is discussed first. Then scheduling and load balancing is considered. A last concern is the granularity of the parallelization and its effects on resource consumption.

2.1.4.1 Communication Methods

In parallel simultaneous processing, processes need the same data elements to read and/or update. The accuracy of data across processes is essential for correctness of the assembled solution. In shared memory architecture, there are four models to maintain this accuracy and these are explained as follows:

EREW: Exclusive Read, Exclusive Write models. This model sets a limitation on the running time of the algorithm, as only one computing node is allowed to access the shared memory whether for read or write.

CREW: Concurrent Read, Exclusive Write models. This model sets much fewer limitations on the running time of the algorithm as compared to EREW. This is due to the allowed simultaneous access of computing nodes to the shared memory for reading. However, a single computing node is allowed to access the shared memory for writing. The implementation in this thesis used this model. Each process had its process memory space and writes its own status files. Then processes communicate by MPI (message passing interfaces) to move overlapping data to be read by the dependent processes' memory as will be further explained later in this subsection.

CRCE: Concurrent Read, Concurrent Write models. This model allows simultaneous access of computing nodes to the shared memory for reading, and for writing as well. However, values to be written are aggregated in a way similar to the summation and averaging.

ERCW: Exclusive Read, Concurrent Write models. This model is provided for completeness; however there is very little application for it.

For distributed processing nodes, there are two main process communication methods. The first method is a Parallel Virtual Machine (PVM), which is a library that allows a heterogeneous computer cluster to appear as one large parallel computer with distributed shared memory. PVM offers some other useful features such as: resource management (adding/deleting hosts from a virtual machine); process control (spawn/kill tasks dynamically); message passing (blocking send, blocking and non-blocking receive with no limit to size and number of messages); dynamic task group (tasks join and leave a group dynamically); and fault tolerance (detects faults – specifies loss of network or hosts - reconfigures the VM transparently).

Equation 1: Amdahl's Law

$$\frac{1}{(1-P)+\frac{P}{N}}$$

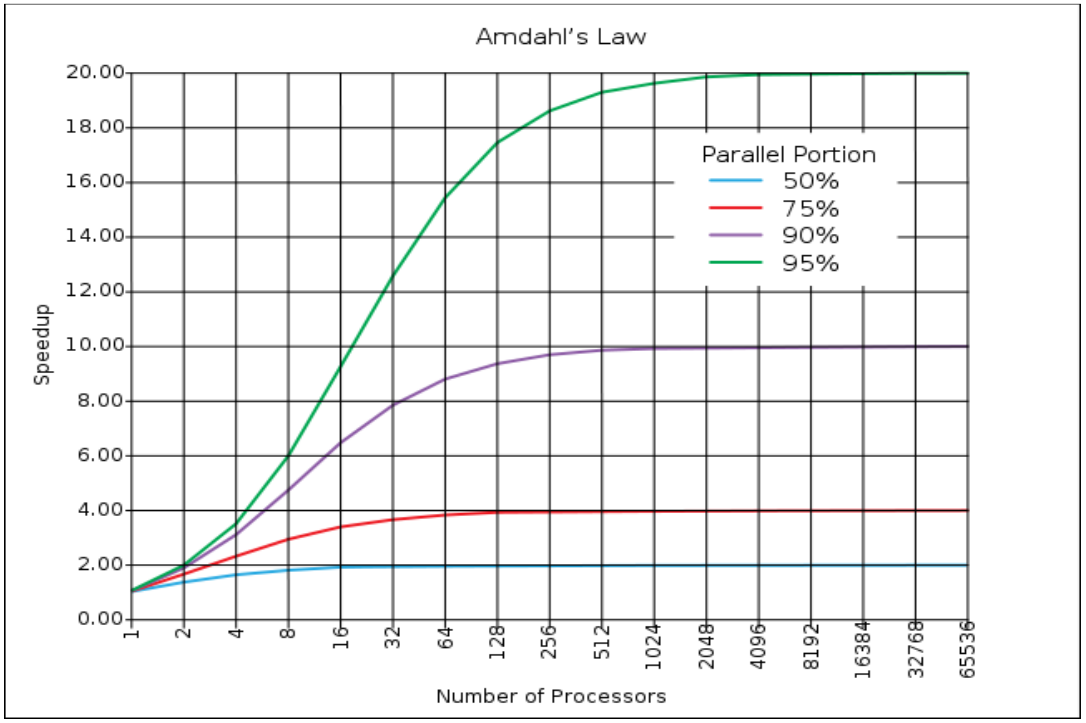


Figure 6: Amdahl's Law

The second method is the Message Passing Interfaces (MPI). MPI are a common Application Program Interface (API) that is used to allow process communication by sending messages. MPI abstracts the hardware architecture to hide implementation details. In shared memory architecture it just passes the messages from the source process memory space to the destination process's memory space. In a distributed architecture, the network infrastructure is used such as Fibre Channel, PCI Express, Serial ATA, InfiniBand, or others. MPIs are implemented on different machines such as Intel, IBM, HP, SGI, COWS, NOWS, PoPCs and Beowulf. The library is implemented using standard networking primitives, thus hiding such details from the user. Other useful features of MPI libraries may be summarized as follows:

- Different name spaces and address spaces amongst processes.
- Handling all network reliability issues like retransmission / handshake.
- Point-to-point primitives (gather, scatter, broadcast … etc).
- User defined data types and topologies.
- Non-blocking send and receive functions.

As mentioned earlier, MPI is the communication library used in the implementation of this thesis work, due to its suitability for shared memory architecture and distributed processing elements. Several implementations have been working well with the current code. MPICH is the library used for simulations on PCs as explained in (GROPP, W, Lusk, E, Skjellum, A, 1999) and (GROPP, W, Lusk, E, Thakur, R, 1999). On the SGI Altix ac cluster, the SGI Message Passing Toolkit (MPT) was used since it is optimized for SGI computer systems running SGI IRIX and Linux operating systems.

2.1.4.2 Scheduling and Load Balancing Techniques

Scheduling is the process of allocating resources to processes so that they can terminate within a reasonable amount of time. The idea is based on finding a feasible and optimal job sequence that can pass on the resources (the machines in a distributed environment) that satisfy some scheduling criteria (JONES, A, 1998). Scheduling methods are necessary to balance between the communications costs against the computation complexity costs. Fixed scheduling is based on dividing the computation over the available processors using a fixed

equation that is hard coded in the implementation. However, dynamic scheduling methods consider the dependency and neighbourhood between the different partitions assigned to each processor. A functional classification scheme for scheduling problems is defined in (GRAVES, S, 1981), and categorizes problems using various criteria. There are a number of scheduling approaches used in the literature (KUMUR, V, Grama, Y, Nageshwara, V, 1993). These include the following:

- Mathematical programming – limited because they are NP-complete problems such as: Integer programming, Mixed-integer programming, and Dynamic programming
- Dispatching Rules (also known as scheduling rule, sequencing rule, or heuristic), such as:
    - Simple priority rules (Processing times: SPT, Due dates: EDD, Slack: MINSLACK, Arrival times: FIFO),
    - Combinations of rules, example: use SPT until the queue length exceeds 5 then switch to FIFO.
    - Weight Priority Indexes, using an objective function to assign weights to different pieces of information about the job.
- AI techniques such as: Expert systems, Knowledge-based systems, several search techniques.
- Neural networks, such as: Supervised learning neural networks, Relaxation models, unsupervised neural networks (competition-based), and Temporal reinforcement learning.
- Neighbourhood search methods such as: Tabu search, Simulated Annealing, and Genetic Algorithms.
- Fuzzy logic,
- Inductive learning

Chapter 3 uses MoA partitioning to parallelize the MSA problem. A spare scheduler process is required. A number of simple scheduler approaches were implemented such as: bag of tasks and round robin. However, in Chapter 4, another partitioning approach is adopted where scheduling becomes a simple assignment of partitions over processors based on the neighbourhood of the partition's indices. This approach keeps the inter-processor communication of dependency between waves of computation to the minimum and requires no further computation cost or dedicated process.

Using an effective scheduling method should balance the computation load among the processors. Load Balancing is the process of efficiently distributing tasks over processors in a parallel distributed environment by making the best use of the processor time (CHINGCHIT, S, 1999). This is achieved by leaving the least number of processors idle because of inter-processes dependence (KUMUR, V, Grama, Y, Nageshwara, V, 1993). In the MSA problem, the dependence on the lower index cells in the alignment tensor structure will cause the first and last partitions to be processed on one processor leaving the remaining processors idle. As we go deeper to the centre of the hyper-cube structure, more processors can be employed based on the dataset size and the partitioning size used. The number of partitions that can be distributed starts small, it then grows towards the centre, and then decays again as we move forward to the last partition.

The objective is to minimize the load imbalance percentage (((largest_load – smallest_load) / largest_load) * 100) as much as possible. This could be achieved by running the program per wave of computation. In each wave, the number of required processors is determined. Then another run starts for the next wave, and so forth.

2.1.4.3 Granularity

Another issue to consider while developing for a parallel processing system is the choice of granularity. The choice of fine-grain or coarse-grain parallel models is to be made based on the hardware architecture used and the problem requirement and its implemented solution. Many Computational Biology problems require fine-grain parallelism. In this thesis, a generic grain parallelism is implemented for the MSA problem. The granularity is controlled by the chosen partitioning size and the number of processors. This is due to the high degree of dependence in the calculation of lower neighbours first as well as the high cost of communication.

2.2 Tensor Indexing Methods

This section surveys tensor computing techniques in the literature. The first subsection presents a historical timeline of all work developing the tensor computing to its current status. The second subsection describes the mathematical background. The third subsection presents the Conformal Computing methods, the Mathematics of Arrays (MoA) and the Ψ-

Calculus methods that form the foundation used for the partitioning in the implementations in this thesis.

2.2.1 Array Functions Operators History

The following is the timeline of the development of array, matrix, and tensor indexing methods and data structures.

1853: Matrix, Matrix Theory, and the principles of Universal Algebra are developed by Joseph Sylvester and Arthur Cayley (BELL, ET, 1986).

1874: Set theory is developed by Georg Cantor; it represents collections of abstract objects including notions, like Venn diagrams and set memberships (TILES, M, 2004).

1908: Axiomatic Set Theory is developed by Ernst Zermelo reformulating the now "naive set theory" in first order logic to resolve its paradoxes, for example, the Russell's paradox, the Burali-Forti paradox, and Cantor's paradox. This theory does not allow the construction of the ordinal numbers while most of ordinary mathematics can be developed without using ordinals. The later are an essential tool in most set-theoretic investigations (HEINZ-DIETER, E, 2007).

1925: Werner Heisenberg, Max Born, and Pascual Jordan formulate matrix mechanics, a formulation of quantum mechanics (GILDER, L, 2008).

1922-1940: Von Neumann-Bernays-Godel set theory (NBG) can be finitely axiomatized. The ontology of NBG includes classes as well as sets; a set is a class that is a member of another class. NBG and ZFC are equivalent set theories in the sense that any theorem about sets is provable in NBG if and only if it is provable in ZFC (MENDELSON, E, 1997).

1942-1945: The category theory is first introduced by Samuel Eilenberg and Saunders Mac Lane in connection with the algebraic topology. It has several aspects such as "general abstract nonsense". The later refers to the high abstraction level, compared to more classical branches of mathematics. The homological algebra is a category theory in its aspect of organising and suggesting calculations in abstract algebra. Diagram chasing is a visual method of arguing with abstract "arrows". The topos theory is a form of abstract sheaf theory, with geometric origins; it leads to ideas such as the pointless topology (AWODEY, S, 2006).

1964: Iverson uses the Array Programming Language (APL) APL notation to describe IBM's system 360. APL is the first homogenous simple array programming language designed by Kenneth E. Iverson. The language works on entire arrays at once, like the vector instruction set of the SIMD architecture. It yields smaller and more concise programs though no iteration is involved (FALKOFF, AD, Iverson, KE, Sussenguth, EH, 1964).

1973: Based on APL, Trenched More proposes an array theory that offers a powerful set of operators and operations on nested heterogeneous rectangular arrays (MORE, T, 1973).

1979: Programming Language/Systems PL/S II or AT/370 language is developed to implement the More's array theory operations, using an APL interface. $NIAL_2$ (Nested Interactive Array Language) is developed as a programming language based on array theory and its applications to make it easy to rapidly develop loop-free data-driven algorithms (JENKINS, MA, Franksen, OI, 1992). APL2 was another implementation of More's nested Array theory. For a while it was IBM's strategic language for HPC.

1988: Mathematics of Arrays and The Ψ-Calculus were first introduced in the PhD thesis of Dr. Lenore Mullin in Computer and Information Science at Syracuse University, Syracuse, NY (MULLIN, LR, 1988). The thesis introduces an algebraic formulation of representing all data structures, invariant of dimensionality and shape. A MoA structure describes scalars as rank 0, linear arrays (vectors) as rank one, 2-D arrays (matrices) as rank two, and so forth. The representation is stored in memory in a linear structure with elements stored in row or column major order in a linear array, a dimensionality scalar, and a shape vector. A list of constructs is provided. The Ψ-Calculus is a way to combine expressions in the MoA algebra by composing indices; it uses the Ψ-Function as its foundation. Mullin's dissertation put closure on work started by Phil Abrams ("An APL Machine", Stanford '72, Harold Stone advisor) who believed there was formalism for array reasoning based on shapes and indexing. His work was augmented by Hassett and Lyon, Guibas and Wyatt, Perlis, Miller, Minter, Tu, Gerhart, Berkling, and Budd, to name a few.

1990: A Comparison of Array Theory with Mathematics of Arrays is presented by L. Mullin and M. Jenkins. (JENKINS, MA, Mullin, LR, 1991).

1993: L. Mullin and G. Hains show how to use the Bird-Meertens Formalism to define MoA. (HAINS, G, Mullin, LR, 1993).

2000: MoA library is built as dynamic link library (dll); it is then used in a basic image and video processing applications in an MSc. thesis presented by the author of this PhD thesis (HELAL, M, 2001).

2001: Faster Fast Fourier Transform (FFT) and generalized Radix n FFT using MoA are presented by L. Mullin and S. Small (MULLIN, LR, Small, S, 2002).

2005: L. Mullin uses MoA and the Ψ-Calculus to map Digital Signal Processing (DSP) algorithms to multiple processor/memory hierarchies. "A Uniform way of reasoning about array-based computation in radar: Algebraically connecting the hardware/software boundary" presented by Mullin (MULLIN, LR, 2005).

2005: Mullin and Raynolds apply the Conformal Computing Techniques by using MoA and Ψ-Calculus to solve problems in Computational Physics (MULLIN, LR, Raynolds, J, 2005).

2008: Mullin and Raynolds write a book which describes how to use MoA to connect the hardware/software boundary of scientific algorithms. Quantum simulation and quantum algorithms are also discussed (MULLIN, LR, Raynolds, JE, 2008).

2004-2009: This thesis applied the MoA methods in high dimensional scientific computation problem "Multiple Sequence Alignment in Bioinformatics", in the MSA dynamic programming algorithm to score a tensor of alignments. Partitioning is processed in parallel providing automatic load balancing (HELAL, M, Mullin, LM, Gaeta, B, El-Gindy, H, 2007), (HELAL, M, El-Gindy, H, Mullin, LM, Gaeta, B, 2008), (HELAL, M, Mullin, L, Potter, J, Sintchenko, V, 2009).

2009: The NSF held a workshop to decide future trends in tensor computation. Researchers from mathematics, physics, and computing presented the state of the art in the field in this workshop (VAN LOAN, CF, 2009).

2.2.2 Mathematical Background

A tensor is a multidimensional mathematical object (n-d arrays) as defined in (KOLDA, TG, Bader, BW, 2009). It can be thought of as a linear machine for performing some operations. A zero-th order tensor is a scalar; a first order tensor is a vector; a second order tensor is a

matrix; a third order tensor is a cube; . . . etc. A k-th order Cartesian tensor, in a k-dimensional Cartesian coordinate system is defined as follows:

- It lives as an entity in the k- dimension coordinate systems.
- Can be represented by *k* indices (subscripts) and N^K components total, where N is the size of every dimension.
- The term "tensor" will be used for structures that are multi-indexed (multidimensional) arrays.

The word "tensor" stems from the Latin word "tensus" meaning "stretched". It was first utilized to describe the elastic deformation of solids. Tensor calculus was refined in the beginning of the 20th century by the Italian mathematicians Ricci and Levi-Cevita. Since then, tensor calculus became an invaluable tool in differential geometry, special and general relativity, and several branches of Physics. The abstract modern view of a tensor is that tensors express some definite type of a multi-linear concept, as linear maps, manipulated with extensions of linear algebra to multi-linear algebra. The classical way of using tensors is to let them define coordinate invariant linear operators. Famous Turing Award winner, Andrew Yao, is using tensors and Dirac Notation to prove satisfiability in quantum algorithms.

Tensors, also referred to as multiway arrays, are higher-order generalizations of vectors and matrices, which are represented as $\xi \in X^{I_1 \times I_2, \ldots, \times I_k}$, where the order of ξ is *k (k >2)* while a vector and a matrix are arrays of orders 1 and 2, respectively. Tensors have a different terminology compared to two-way data sets. Each dimension of a multiway array is also called a mode or way, and the number of variables in each mode is used to indicate the dimensionality of a mode. For instance, $\xi \in X^{I_1 \times I_2, \ldots \times I_k}$ is a multiway array with *k* modes (called k-way array or k^{th}-order tensor) with $I_1, I_2 \ldots I_k$ dimensions in the first, second, ..., and k^{th} mode, respectively. Each entry (referred to as element or cell later in the text) of ξ is denoted by $c_{i_1, i_2, \ldots, i_n}$. For a special case, where *k* = 3, let $\xi \in X^{I_1 \times I_2 \times I_3}$ be a three-way array. Then, element c_{i_1, i_2, i_3} denotes the entry in the i_1^{th} row, i_2^{th} column, and i_3^{th} tube of ξ as shown in Figure 7 (a) to (c). When an index is fixed in one of the modes and the indices vary in the two other modes, this data partition is called a slice (or a slab). For example, when the

i_1^{th} row of ξ is fixed, then it is a horizontal slice of size $I_2 \times I_3$ as shown in Figure 7 (d) to (f) (ACAR E, Yener B, 2009).

2.2.3 Conformal Computing Methods

Conformal Computing [3] as described in (MULLIN, LR, Raynolds, J, 2005) is a formalism based on an algebra of abstract data structures, namely, Mathematics of Arrays (MoA), array indexing calculus, and the Ψ-Calculus. The method allows the composition of a sequence of algebraic manipulations in terms of array shapes and abstract indexing.

It is called "Conformal Computing" because the mathematics used to describe the problem is the same as that used to describe the details of the hardware. Thus, at the end of a derivation, the resulting final expression can simply be translated into portable efficient code for implementation in hardware and/or software. The approach allows for partitioning an n-dimensional tensor based on a given MoA function. MoA offers a set of constructs that represents multidimensional arrays in memory in a linear, concise and efficient way, with many useful properties, and applications. It describes an array calculus containing a set of operator definitions, shape definitions, and reduction rules all based on a single indexing operator. Algebraic operators are included in the Ψ-Calculus to form a broad set of operators needed to describe complex array operations. All the operators are extended for scalars, vectors, and multi-dimensional arrays; i.e. it works invariant of dimension and shape.

MoA provides a higher dimensional Euclidean space. The design of MoA starts by analysing the operational requirements for one dimension, then expands the requirements for two dimensions, then three dimensions, and finally to an arbitrary dimension. Like the high dimensional Euclidean space, MoA provides index transformations, namely, rotational, multidimensional transpose, slicing, partitioning, reshaping, copying, and shifting amongst other manipulations.

---

[3] The name Conformal Computing © is protected. Copyright 2003, the Research Foundation of State University of New York, University at Albany.

This technique reduces the complexity of indexing and nested loops in handling the arrays; it uses an array of indices and a Ψ function to create a semantic normal form. This normal form is independent of layout, i.e. row, column, or other. Using a mapping function "gamma" as explained in Table 1, we can calculate the flat array index, i.e. offsets, in the flat one- dimensional array in memory. This approach reduces computation steps and allows a simpler representation of the high dimensionality complexity involved with many scientific computational problems. This method also minimizes the use of temporary arrays for storing intermediate results in computations.

MoA mathematically relates the array indices in multidimensional arrays to one flat array using the Ψ-Calculus, and implements operations as index transformations. Note that a layout may be in either row or column major ordering. Consequently, MoA is a set of mathematical functions and a data structure with a closed algebra. The work in this thesis uses MoA to partition the Multiple Sequence Alignment (MSA) scoring high dimensional lattice for distribution on high performance parallel machines and/or PC clusters.

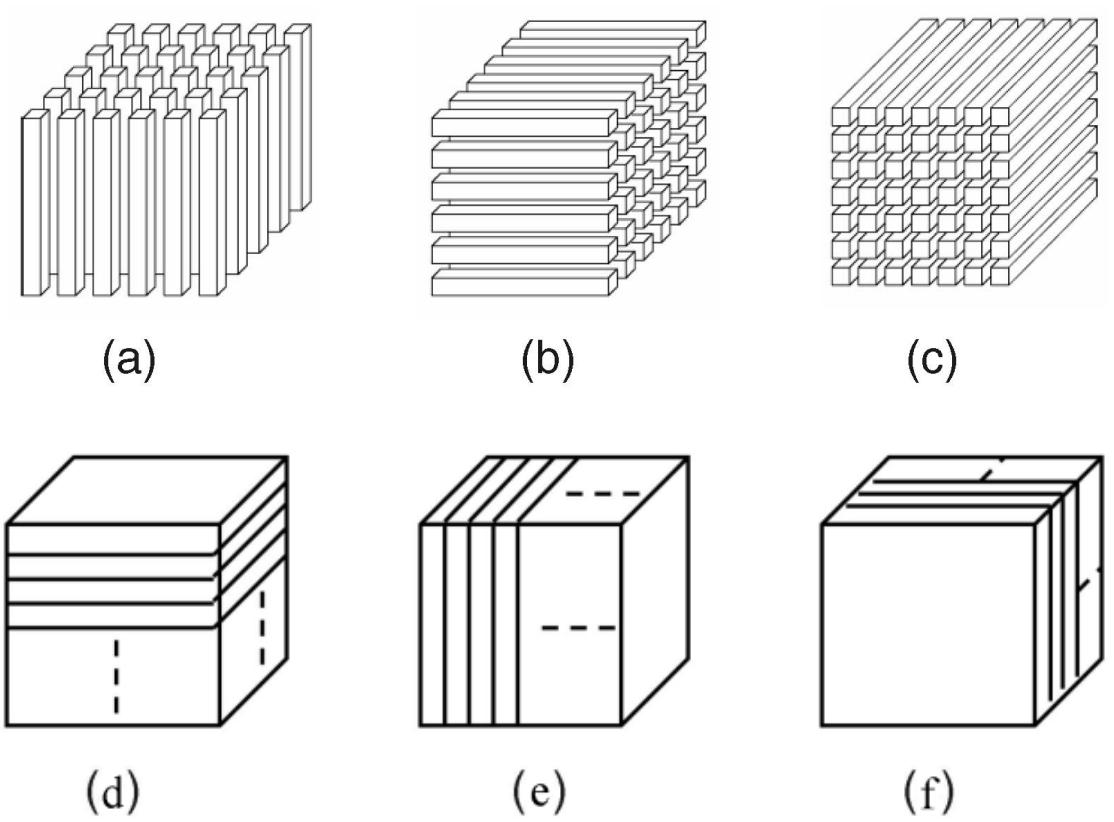


Figure 7: Multi-way Arrays (a) Columns. (b) Rows. (c) Tubes. (d) Horizontal Slices. (e) Vertical Slices. (f) Frontal Slices.

Table 1: Some MOA Operators

Operator	Function	Example
δ	Delta (dimensionality)	δ ξ = 2
ϱ	Rho (shape)	ϱ ξ = <2 3>
↑	Take (subarray)	1 ↑ ξ = < 1 2 3>
↓	Drop (subarray)	1 ↓ ξ = < 4 5 6 >
rav	Ravel (flatten)	rav ξ = < 1 2 3 4 5 6 >
ι	Iota (count)	ι 5 = < 0 1 2 3 4 >
ψ	Psi (Indexing function)	< 1 0 > ψ ξ = 4
π	Pi (Product operator)	π = <2 3> = 6
τ	Tau (Total Elements Operators)	τξ = 6
γ	Gamma (Flat Index Retrieval)	<0 0 > γ ξ = 0, < 1 0 > γ ξ =
γ'	Gamma' (Multidimensional Index Retrieval)	0 γ' ξ = <0 0 >, 3 γ' ξ = < 1

For a full listing of the MoA constructs, please refer to (MULLIN, LR, 1988). Table 1 lists some of the more useful Ψ-Calculus operators together with an example usage on the following array:

$$\xi_2 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Data transformations occur by index manipulations in order to facilitate programming invariant of dimension and shape. For example, in the implementation of this thesis, Multiple Sequence Alignment (MSA) is solved by initializing border cells (where one index is zero), and using a recurrence that reads neighbouring cells to score an inner cell. In MSA, dependency is defined as neighbouring cell scores that need to be calculated first before a dependant cell score can be calculated. In distributed processing, these are the overlapping cells among partitions that are calculated by one processor and read by another processor for the calculations of dependant cells that fall in the later processor's partitions assignment. A window of dependency is created once, which is a tensor of the same dimensionality as the data set, and of shape equals to one in all dimensions to retrieve direct neighbours only as required by the problem statement in section 2.4.2.3. This window is implemented as a linear array buffer that is created for every process; it sweeps the required dependent cell values to the current cell position (whether local – in the same partition – or remote – read from

another processor). As the scoring proceeds to new cell position, the dependency window moves with it to retrieve the new neighbours. This is further detailed in Chapter 3.

MoA allows reading large contiguous blocks of data, thus achieving speed-up in a uni-processor and in a multi-processor environment. Since the cache reads the nearest memory locations, MoA is well suited to cache structures as well as to scheduling over shared-memory multiple- processor (SMP) MIMD architecture. Implementing MSA using MoA, can benefit from the model of memory caches, concurrency of sub-array operations, and low process communication. Moreover, since the access patterns are determined prior to execution, pre-fetching at all levels of the processor/memory hierarchy are possible.

MoA models the tensor points (each data element) individually, without assuming any structure, using all degrees of freedom. This would give the perfect fit of the data, and is considered the simplest model. The MoA has been given a full closure in (MULLIN, LR, 1988). However, because of the mathematical difficulty and the computational complexity of the methods, the methods were not adopted for many applications since then.

A generic computational implementation of the MoA constructs was presented in (HELAL, M, 2001) to serve as a library of data structures and operations to perform k-way array computation invariant of dimension and shape. The library was applied in basic operations for images as 2 dimensional arrays, and videos as 3 dimensional arrays. The main advantages of using this library rather than using static dimensional array computation are:

1. Elimination of static nested loops per dimension, and hence generating different code for each data set based on its dimensionality.
2. Optimized memory allocation, as the whole tensor is stored linearly in memory and the indexing system defines the position in the k dimensional space. If this is optimally utilized in the context of the problem at hand, fewer cache misses should occur.
3. Re-use of the same problem based code to higher dimensional spaces without changes to accommodate for neighbourhood relationships.

Further applications are mentioned in the timeline in section 2.2.1.

This thesis uses the MoA constructs to solve a higher dimensional dynamic programming algorithm to be partitioned for parallel processing on different architectures. The partitioning

of the multidimensional scoring tensor in this problem according to the dependency requirements (as explained earlier and further explained in Chapter 3) reveals a structure around a novel definition of the tensor points into waves of computations. These waves form a higher level clustering of the partitions that can be scored simultaneously in all processors (with minimal dependency in Chapter 3, and without dependency in Chapter 4), and later waves of computations read the overlapping cell scores. The waves of computations for the MSA problem are orthogonal to a diagonal of the tensor. A later optimization in Chapter 5 creates a logical middle point in each computation wave that can form a logical hyper-diagonal through the tensor space. This structure allows for scoring only a small band around these middle points in each computation wave, thus reducing the exponential complexity down to polynomial complexity. Similar studies can transform data points using indices to reveal solution-based structures that can be partitioned independently.

The tensor indexing scheme presented from the MoA represent the axes in a cubical form as shown in Figure 9. The axes are represented as the gray lines, while the first three axes representing the smallest cube are omitted. The basic smallest shape is a three dimensional cube that can be of any defined length. For simplicity, we will consider the points on the corners of the cube in the shape only, therefore the upper bounds on the first initial axes is 2. This makes the shape in Figure 9 a 7 dimensional hyper cube of shape vector equal to <2, 2, 2, 4, 3, 3, 2>. The first three dimensions are the basic cube shape, and then the four cubes are repeated on the fourth dimension shown with the small horizontal gray axes. Then all points in the four dimensions covered so far are repeated upwards three times on the small fifth vertical gray axes. Again all points will be further enclosed on another larger cube for the five dimensions covered so far, and repeated on the large horizontal sixth dimension gray axis. Finally a seven dimension is drawn on the large vertical seventh gray axes, by repeating all smaller shapes upwards. For any number of dimensions, this enclosure of cubes within each other, up to the bounds (shape) of each dimension represent the row major order of indexing used in the MoA tensor. Similar plots of data points can be used for visual classification. Each point in the tensor grid is the intersection of all axes lines on the corner of its enclosing cube.

2.3 High Dimensional Data Computation

This section surveys the various methods used in the literature to handle high dimensional data computation. Mainly dimensionality reduction methods are used and these are presented in the first subsection. However, mapping to higher dimensions can reveal more information and this is examined in the second subsection. The approach used in this thesis is described in the third subsection, which is employing parallel processing to partition a tensor space.

2.3.2 High Dimensional Computing and Dimensionality Reduction Methods

Regression analysis is difficult for high dimensional computational problems. This is due to the large number of variables and parameters $\boldsymbol{p}$ (for example genes) and a very small number of cases or records $\boldsymbol{n}$ (for example replicas or patients). Examples include microarray data in Computational Biology amongst other relevant high dimensional computational problems. Therefore, many methods are developed to approximate the high dimensional problems. The Johnson-Lindenstrauss Lemma states that a small set of points in a high dimensional space can be embedded in a much lower dimensional space (approximately $\log k$ dimensions – where k is the original dimensionality), while the distances between the points are preserved. There are various dimensionality reduction techniques in literature, including the following (Fodor, 2002):

- Linear Methods such as: Principal Component Analysis: PCA (JOLLIFFE, IT, 2002), Factor Analysis: FA (TUCKER, L, MacCallum, R, 1997), and Maximum likelihood factor analysis for second order data summaries(EDGEWORTH, FY, Fisher, RA, 1976), and Projection Pursuit: PP (FRIEDMAN, JH, Tukey, JW, 1974), Independent Component Analysis: ICA(COMON, P, 1994), one-unit or multi-unit objective functions, optimization algorithms for non-Gaussian datasets.
- Non-Linear methods such as: Non-linear PCA, Non-linear ICA, Random Projections, Principal Curves, Multi-Dimensional Scaling (MDS), Topologically Continuous Maps, Neural Networks (NN) models, Vector Quantization, Genetic and Evolutionary algorithms, and Regression Methods.

Generally, there is a data loss in all dimensionality reduction techniques while trying to preserve the maximum amount of information from the original data. The study in (Fodor, 2002) shows that the effects of dimensionality reduction are largely data dependent. It also

proves that simple changes in the implementation details can often lead to considerable improvements. However, some observations may be made as follows:

- Distance functions for high- dimensional applications are often heuristically designed; there is no firm consensus on how similarity may be measured for an arbitrary data set.
- There are considerable dependencies among attributes in high-dimensional data, which are ignored by distance functions such as the Euclidean metric on the original representation. These result in poor measurements of similarity.

According to (WANG, X, 2006), it is desirable to transform the functions such that the effective dimensions and the mean dimension are reduced, i.e., transform the functions in a way that enhances the degree of additivity. It is shown that PCA changes the functions to be strongly weighted, with the first few weights substantially more significant than others. Thus, it reduces the effective dimension and increases the degree of additivity. However, it does not necessarily reduce the effective dimension of an arbitrary problem. PCA is found to undesirably increase the mean dimension remarkably, 1.873 as compared to 1.336 for Brownian Bridge (BB) and 1.035 for the standard construction of Brownian motion. This makes the problem harder for Quasi Monte Carlo Analysis (QMC), since it may be useless to apply dimension reduction techniques when the function is additive.

2.3.3 Mapping to higher dimensions: Support Vector Machine (SVM)

Instead of reducing the dimensionality of the data, mapping it to higher dimensions can increase its chances for classification and learning. Support vectors are the data points that lie closest to the decision surface, and the most difficult to classify. These will change the position of the dividing hyper-plane if removed from the dataset. SVM is a quadratic programming problem that identifies the optimal hyper-plane for linearly separable classes. SVM can be extended to patterns that are not linearly separable by transformations of original data to map into new space using a Kernel function. The optimal hyper-plane stems from the function class with the lowest "capacity" (VC dimension). SVMs maximize the margin around the separating hyper-plane. The decision function is fully specified by a subset of training samples, the support vectors. In linear programming, finding a line that separates

two clusters of points is defined as shown in Figure 8. The objective is to find a, b, c, such that $ax + by \geq c$ for circular points $ax + by \leq c$ for square points. To find out which line is optimal for all points, linear regression and Naïve Bayes can be used. However, to find the optimal line for the points of the support vectors, SVM and logistic regression can be used.

SVM methods can benefit from the Conformal Computing indexing scheme and its parallelizable attributes, for more accurate simulations. This may be a fruitful area for future research for applications of MoA.

2.3.4 High Dimensional (Tensor) Partitioning for Parallel Processing

High dimensional partitioning for parallel processing is another approach in large scale computation. The work in (PRAKASH, SR, 1998) presents a hyper plane partitioning method that partitions the iteration space into as many partitions as there are number of processors keeping communication to a minimum. The method finds a best local distribution of data for every loop, then finds the best way of grouping different adjacent loops to find the best global data partition. This process constructs a data distribution tree that is traversed to find the shortest path from the source to a node. This cost constitutes the communication cost to run a loop through this path. This method worked well for a few dimensions, and then heuristics were used to run full programs for higher dimensions.

In this thesis, modelling high dimensional spaces and partitioning them for parallel processing is presented, with an application to the MSA problem. The same modelling techniques can be used for data classification and clustering algorithms. In Chapter 7, the future work points identify possible research directions that attempt to find structure in data points read from a database and modelled as tensor points using the model presented in this thesis.

2.4 Computational Biology

The techniques used in biological laboratories have advanced considerably since the discovery of the DNA structure in 1953, generating a large amount of data that needs to be studied by analysis and comparison (WATSON, JD, Crick, FHC, 1953). Bioinformatics, computational molecular biology, or comparative genomics are all terms used among others to refer to a class of problems that are concerned with the automated analysis of genome

structures and functions across different biological species or strains. Gene finding, the discovery of non-coding functional elements of the genome, identifying evolution mechanisms, and gene expression analysis, are all goals of comparative genomics studies or phylogenomics (DURET, L, Abdeddaim, S, 2000).

Sequence comparison is one of the major computational biology approaches that are used in several contexts. The idea is to compute the degree of similarity between 2 sequences or more, by aligning them in rows on top of each other so that more matching residues or substrings can be obtained by using insertions, deletions, and substitutions as needed to maximize the number of matched residues in the columns of the alignment. The score of the produced alignment describes the degree of similarity of the tested sequences. The sequences might be homologues (sharing the same common evolutionary ancestry), or have some degree of similarity (sharing functionality - regulatory role in case of DNA, or similar biochemical function or 3 dimensional structure in the case of proteins).

This section reviews the literature of computational biology. Background information is presented in the first subsection. The second subsection presents the Multiple Sequence Alignment (MSA) problem formal definition and the various algorithms and methods implemented to solve it. The third subsection lists a number of problems in the existing methods and emphasizes the need for new approaches.

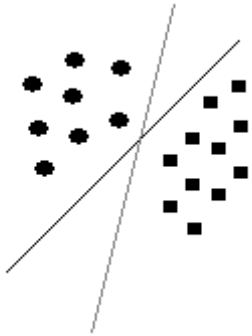


Figure 8: Linear Programming

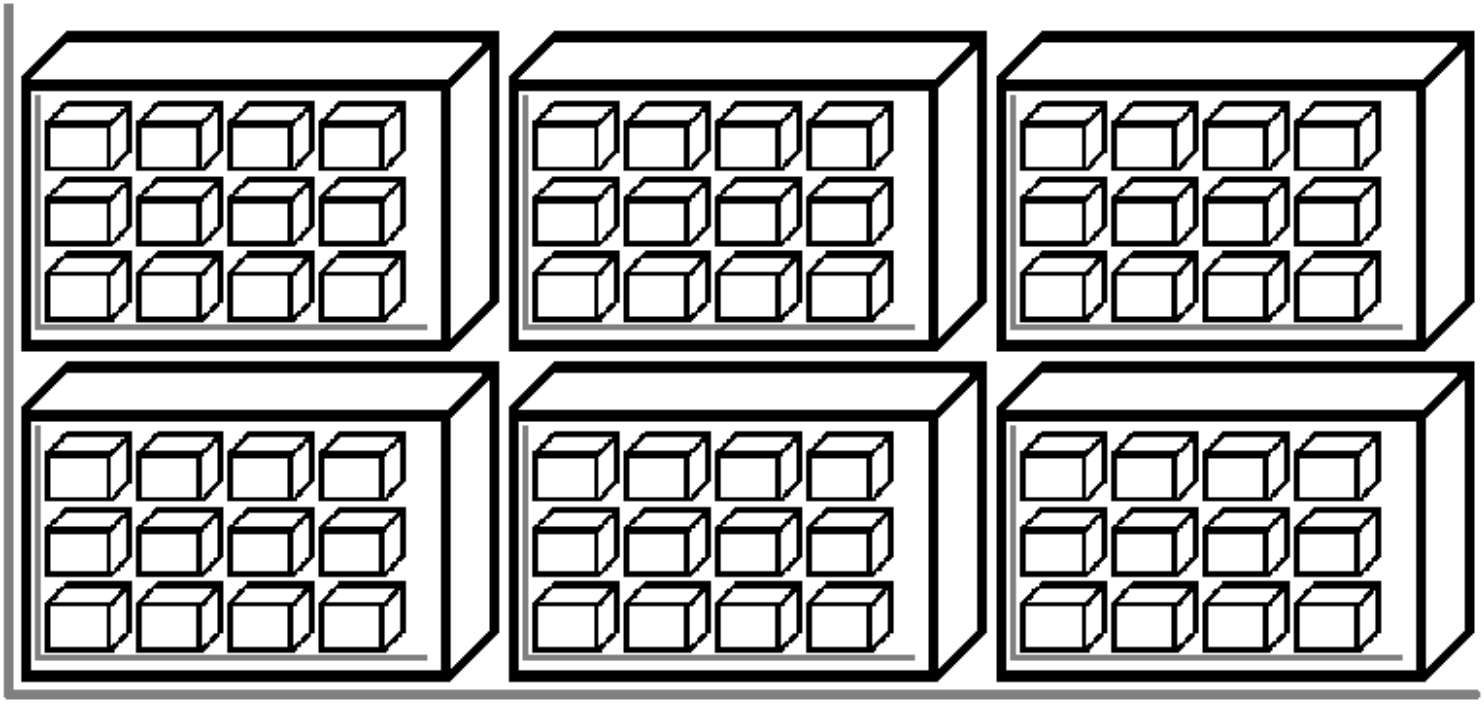


Figure 9: Tensor Coordinates Based on the MoA Indexing Scheme

2.4.1 Background Information

This section provides the background information and terms definitions that will be used in the subsequent Chapters. Various concepts and methods are used in computational biology. Only the ones used in the implementation of this thesis, or in the evaluation of the produced tool will be explained in this section.

Sequences File Formats: The biological sequencing technology determines the primary structure (sequence) of an unbranched biopolymer, which are polymers produced by living organisms. A sequence is a symbolic linear depiction of the nucleotide order of a given DNA fragment, which succinctly summarizes much of the atomic-level structure of the sequenced molecule. The sequences are strings of characters from a defined set of alphabet, such as {A, C, G, T} for DNA, and {A, R, N, D, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V} for proteins. Wild card characters (representing two or more of the defined alphabet) are used in sequence family profile generation denoting positions with more than one acceptable residue. The generated information is saved in computer files using different formats such as ASCII text files that hold the sequence and information about the sequence data. There are a couple of dozen sequence formats currently in use, each package comes with its own format, some of the most widespread sequence formats used by the major sequence databases are: *fasta*, *EMBL*, *GenBank*, *SwissProt*, and *PIR*.

Scoring Methods: In comparative genomics, the alignment of sequences is based on a scoring scheme to evaluate when inserting a gap (denoting an insertion or a deletion is the respective sequence) is preferred rather than a substitution of the residues at the alignment column. The identity matrix defines a score for exact matches and mismatches as shown in Figure 10.a. Since mutations happen randomly at the DNA level, causing the fitness of the organism to increase or decrease, and to continue to next generations, or removed by natural selection, the first order Markovian Model assumes that subsequent amino acid substitutions occur with a probability independent of previous substitutions, or substitutions at other positions in the polypeptide chain (RAO, JKM, 1987). It also means that a single substitution matrix can represent the probability of all amino acid substitutions at any and all positions in a protein.

Allowable differences are accepted in DNA since amino acid sequences diverge over evolutionary time. Some mutations are favourable, in the sense that the amino acids have similar properties and thus do not affect the overall structure of the resulting protein. Studies have been performed to find which substitutions are allowable. There are various scoring matrices provided in literature such as PAM250 that is based on statistical analysis of the likelihood of the biological substitutions of proteins. There is also Dayhoff, which is based on percent-accepted mutations. Other scoring schemes can be used to reveal more or different information such as: 1 PAM Unit Mutation Matrix, 100 PAM Unit Mutation Matrix, 250 PAM Unit Mutation Matrix, Blosum62 Mutation Matrix, Gonnet, and Johnson (SCHWARTZ, RM, Dayhoff, MO, 1978). It is believed that for protein sequence comparison, the PAM-250 matrix has been widely used in the literature; for database searches the PAM-I20 matrix is more appropriate, while the PAM-200 matrix is indicated for two specific proteins with suspected homology (ALTSCHUL, SF, 1991). There is also BLOSUM (HEINIKOFF, S, Heinikoff, JG, 1992) matrices as shown in Figure 10.b.

Gap Penalties: Insertions and deletions cause gaps to be inserted in the aligned sequence. There are a number of methods for handling gap insertions. Linear gap penalty, adds the same penalty for inserting a gap of equal weight all the time. Affine gap penalties gives higher weight to the first gap inserted, and then less constant weight to all additional spaces in the same gap.

Benchmark Alignment databases: To measure the performance of an MSA tool, some benchmarks databases are available that has been manually assembled by biologists. Major databases are Balibase, PREFAB, and Homstrad.

Balibase is a hand-written reference sequence alignment as desired by biologists. It contains 142 reference alignments with over 1000 sequences. Of the 200,000 residues in the database, 58% are defined within the core blocks. The remaining 42% are in ambiguous regions, which cannot be reliably aligned. There are four hierarchical reference sets. Reference 1 provided the basis for construction of the following sets. Each of the main sets may be sub-divided into smaller groups, based on sequence length and percent similarity. (THOMPSON, JD, Plewniak, F, Poch, O, 1999).

a.

	A	C	G	T
A	1			
C	0	1		
G	0	0	1	
T	0	0	0	1

b.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W

Figure 10: Scoring Matrices. a) DNA Identitiy Matrix b) BLOSUM62 amino acid scoring matrix

- **Reference 1:** contains alignments of (less than 6) equi-distant sequences, i.e. the percent identity between two sequences is within a specified range. All the sequences are of similar length, with no large insertions or extensions.
- **Reference 2:** aligns up to three "orphan"" sequences (less than 25% identical) from reference 1 with a family of at least 15 closely related sequences.
- **Reference 3:** consists of up to 4 sub-groups, with less than 25% residue identity between sequences from different groups. The alignments are constructed by adding homologous family members to the more distantly related sequences in reference 1.
- **Reference 4-5:** both 4 and 5 contain alignments of up to 20 sequences. It includes N/C-terminal extensions (up to 400 residues), Reference 5 internal insertions references include up to 100 residues.

PREFAB is a test set constructed from a pair-wise structural alignment, where each sequence is used to query a database to collect high scoring hits, then the 2 sequences and their hits are combined and aligned using an MSA program (EDGAR, RC, 2004). It contains an average of 49 sequences of length 242. Accuracy is assessed on the original pair alone, by comparison with their structural alignment. Prefab uses a set of pair-wise structural alignments selected from 500 families at random from the FSSP database. Within each family, three pairs of structures were chosen at random from the sequence identity ranges 0-

15%, 15-30% and 30-97%, giving a total of 1,484 pairs. Then, each full-chain sequence is used (not restricted to its aligned region) to make a PSIBLAST search of the NCBI non-redundant protein sequence database, keeping the locally aligned regions of hits with Expectation value (e-values) below 0.01. E-value is the number of different alignments with scores equivalent to or better than the raw score (S - calculated as the sum of substitution and gap scores) that is expected to occur in a database search by chance. Hits are filtered to 80% maximum identity and 24 selected at random. Finally, the original pair and their remaining hits were combined to make a set of $\leq$ 50 sequences. The limit of 50 was arbitrarily chosen to make the test tractable for some of the more resource-intensive methods, in particular T-Coffee (NOTREDAME, C, Higgins, DG, Heringa, J, 2000).

HOMologous STRucture Alignment Database (Homstrad): is an online benchmark for annotated structure based alignments for homologous protein family (common ancestor). It is not used for the experimentation in this thesis, because structure alignment was not implemented. However, it is often studied in other MSA methods discussed in this Chapter.

Alignment Tools Performance Evaluation: After an MSA is produced, the performance is measured by comparing how good the constructed alignment produced by the method used. These measures are mainly the sum-of-pairs score and the Shannon Entropy score. Sum-of-pairs score is usually used to assess the performance of MSA methods. This score increases as the program succeeds in aligning more matching residues in each column in the final alignment, with minimum gap insertions all over, assuming statistical independence between columns (PEVSNER, J, 2003). Shannon entropy is a simple quantitative measure of uncertainty in a data set. In the context of drug resistance as conferred from single mutations, knowledge of the frequencies of different amino acids in the mutation position as drawn from resistant and sensitive populations, will enable us to guess the amino acids responsible for the resistance. This is because these amino acids were certain (low entropy) in the sensitive population, versus uncertain (high entropy) in the resistant population (LABORATORY, Los Alamos National web site).

Another measure that is of biological relevance is the weight of each sequence in the produced alignment. This measure is used to avoid taxonomic bias due to evolutionary relationships, and can be used to cluster the sequences into species. There are a number of methods to calculate the weight. The tool used to calculate the weights in the experimentation section from the sequence alignments using different methods as calculated

by MASH (http://timpani.bmr.kyushu-u.ac.jp/~mash/weight_ex.html). The tool offers different weight calculations methods like: No Weight, Difference Weight, Henikoff & Henikoff Weight, and Voronoi Weight.

From a constructed alignment, a distance matrix can be generated using neighbour joining, maximum likelihood, or any of the methods mentioned in the next subsection. From the distance matrix, a phylogenetic tree (dendrogram) is constructed to visualize the evolutionary relationship between the sequences. The distance matrix is generated using dnaDist tool available in the Phylib package (FELSENSTEIN, J, 1989).

As a visual evaluation of the performance of the MSA method used, there is the similarity regions plots generated by tools like plotcon from the European Molecular Biology Open Software Suite (EMBOSS). The tool plots the quality of conservation of a sequence alignment. The similarity along the sequences is calculated by moving a window of a specified length, and the similarity of any one position is taken to be the average of all the possible pair-wise scores of the bases or residues at that position within the window (RICE, P, Longden, I, Bleasby,A, 2000).

Another visual tool is developed using Matlab images of the distance matrix calculated from the constructed MSA. Sorting the distance matrix and generating a heat-map with the sequence names can identify clusters of high similarity where the colour is too cold (blue for instance), or low similarity where the colour is too hot (red for instance based on the colour scheme used).

2.4.2 Multiple Sequence Alignment

MSA is a combinatorial optimization problem, with an objective of finding the best or optimal solution among a finite or countably infinite number of alternative solutions. Methods of Pair Wise Sequence Alignment can be obtained by Dot Matrix analysis, a Dynamic Programming algorithm, or Word, or k-tuple methods, such as used by FASTA (LIPMAN, DJ, Pearson, WR, 1985) and BLAST (ALTSCHUL, SF, Gish, W, Miller, W, Myers, EW, Lipman, DJ, 1990). The Dot matrix method aligns 2 sequences in a matrix where the characters of the first sequence are in the first row in the matrix, and the characters of the second sequence are in the first column. The method places a dot on each identical character from both sequences in the intersection cell as shown in Figure 11. This method shows the

alignments as diagonals of dots on the matrix, revealing insertions, deletions, direct and inverted repeats, but it doesn't show an actual alignment (XIA, X, 2002).

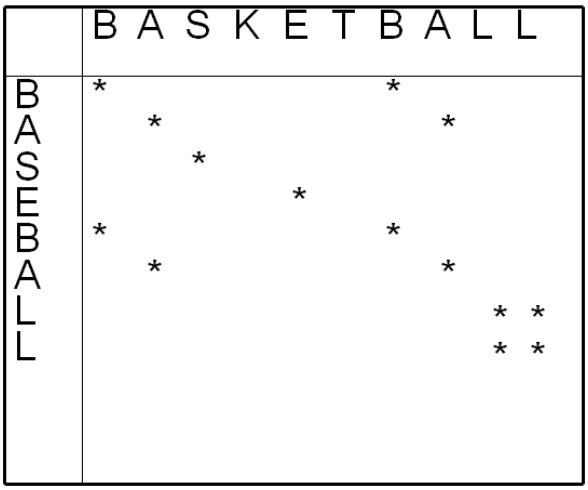


Figure 11: Dot Matrix Alignment Method

Integer, linear and non-linear programming and dynamic programming have all been used for solving optimality and proximity combinatorial optimization problems. MSA with a modest (fixed) number of sequences (dimensions), the problem is barely tractable by the dynamic programming exact solution, i.e. N^2 for two sequences as in Smith-Waterman algorithm (the dynamic programming will be further explained in the next sub-section). However the MSA problem with increasing dimension belongs to the hard NP-complete class of problems; that is, there are no known algorithms with solution time bounded by a polynomial in the number of dimensions as proven in (WANG, L, Jiang, T, 1994) and (JUST, W, 2001). Indeed the exact dynamic programming algorithm requires time which grows exponentially with dimension. To solve large NP-complete problems, one has to choose between going for optimality at the risk of very large, possibly intractable computation time or going for faster solutions at the risk of sub-optimality. The first approach yields optimization problems that are solved by enumeration methods like cutting plane, branch and bound, and dynamic programming. The second approach uses approximation (heuristic) algorithms, like local search and randomization algorithms. Some methods can be configured to be an optimization or an approximation algorithm, for example by introducing heuristic bounding rules to a branch and bound algorithm.

2.4.2.1 Formal Problem Definition:

Let S_1, S_2 . . . S_k be the input set of strings (sequences), where k is at least 2, of lengths vector of natural numbers (shape) $\rho = \{\rho_1, \rho_2, ..., \rho_k\}$. The sequences are sets of symbols of a finite set of alphabets N (such as 20 amino acids in case of protein sequences). The minimum edit distance is defined as d (S_1, S_2 ... S_k).

$\sum$ is the alphabet not containing the gap character "-". A multiple sequence alignment A is defined as 2-dimensional character array over the alphabet $\sum = \sum \cup \{-\}$. A has k rows, each row A_i is the alignment of the string S_i' corresponding to S_i without the gaps, and a pair of rows A_i and A_j is the pair-wise alignment of the sequences S_i' and S'_j with respect to the whole sequences set. A has l columns (alignment length) such that:

$max(\rho_1, \rho_2, ..., \rho_k) \leq l \leq (\rho_1 + \rho_2 + ... + \rho_k)$ .This is illustrated in Figure 12.

Input Sequences:	INDUSTRY INTERESTING IMPORTANT
Output Alignment:	IN-DU-STRY INTERESTING IM-POR-TANT

Figure 12: MSA Input and Output

2.4.2.2 Classes of Search Techniques

Figure 13 illustrates the search techniques found in the literature, among which we will be interested in the Dynamic Programming for the implementations of this thesis. Dynamic programming for MSA employs a scoring matrix for two sequences, or a k-dimensional tensor for k sequences. This tensor will be implemented using MoA and partitioned for parallel processing and optimized for search space reduction. However, due to the extensive use of the other methods in the previous solutions of MSA, we will review these techniques in the following sections as well. Generally, the MSA algorithms in the literature are categorized into exact, progressive and iterative methods (ABDESSLEM, L, Soham, M, Mohamed, B, 2006) and (NOTREDAME, C, 2002).

The existing MSA algorithms include the following:

- Exact Methods
    - Dynamic programming algorithm.
    - Simultaneous Alignment algorithms.
- Progressive alignment algorithms.
- Iterative alignment techniques.
    - Deterministic
    - Stochastic
        - Hidden markov model algorithms.
        - Simulated annealing algorithm.
        - Genetic algorithms.
- FFT algorithm.
- Integer Programming.

2.4.2.3 Dynamic programming Algorithm

Dynamic Programming (DP) is a technique that solves optimization problems. It is like divide and conquer in combining solutions to sub problems, but it also allows for dependence between the sub problems, and saving the sub solutions, so that they don't keep getting re-computed as in the case of divide and conquer algorithms. Dynamic programming requires the following steps:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution in bottom-up fashion
4. Construct an optimal solution from computed information.

The focus of this thesis is on the dynamic programming method in producing optimal (the very best or highest scoring) alignment between sequences. It is proven mathematically to produce the optimal global alignment using the Needleman and Wunch algorithm (NEEDLEMAN, SB, Wunsch, CD, 1970), and for local alignment using the Smith and Waterman algorithm (SMITH, TF, Waterman, MS, 1981). The idea is to start from the ends of both sequences by attempting to match all possible pairs of characters by following a scoring scheme for matches, mismatches and gaps, generating a matrix of numbers that

represent all possible alignments. Then, by following the highest scores on the matrix, the optimal alignment can be found. The substitution scores are defined as a matrix of substitutions over the string alphabet as mentioned in section 2.4.1.

Global Alignment

The pair-wise dynamic programming global alignment algorithm steps are as follows (GUSFIELD, D, 1997):

1. Create an Alignment Matrix of n+1 columns and m+1 rows, where n is the size of the first sequence, and m is the size of the second sequence.
2. Initialize the Alignment Matrix first row and first column, based on the type of comparison required.

Fill in the alignment matrix with scores based on the recurrence shown in Equation 2.

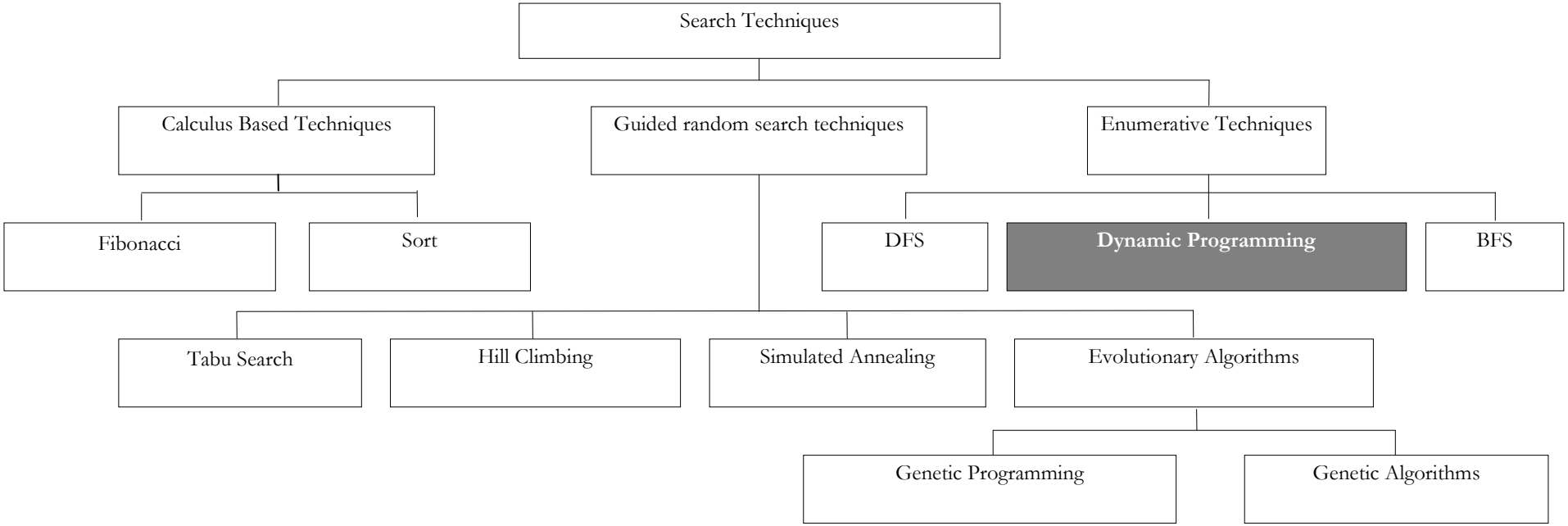


Figure 13: Search Techniques

Equation 2: Dynamic Programming Global Alignment Scoring Recurrence Equation

$$\xi_{i,j} = \max\begin{pmatrix} \xi_{i-1,j-1} + sub(a_i, b_j) \\ \xi_{i-1,j} + g \\ \xi_{i,j-1} + g \end{pmatrix}$$

where ξ_{ij} is the cell in the matrix with row index i (first sequence), and column index j (second sequence), and a_i is the character in the first sequence at position i, and b_j is the character in the second sequence at position j, and g is the gap penalty value, and $sub\ (a_i, b_j)$ is a function that returns the score of substituting a_i with b_j based on the substitution scoring matrix used.

3. Trace back: starting from the bottom right in the scoring matrix (the ends of both sequences), following the highest scores cells paths (up, left and diagonal moves), to find the optimal alignment. Moving diagonally means accepting the substation of the corresponding residues at both sequences. Moving upwards, inserts a gap in the horizontal sequences, while moving left, inserts gap in the vertical sequence.

Step 2 initializes the values of the edge cells of the alignment matrix. Three types of initialization methods can be used. To compare a sequence to sequence, the matrix is initialized using the following formula:

$$\xi_{i,0} = \sum_{0}^{i} g \quad for \quad 0 \le i < M \quad and \quad \xi_{0,j} = \sum_{0}^{j} g \quad for \quad 0 \le j < N$$

Other initialization formulas are used to compare sequence to subsequence and to compare subsequence to subsequence. The interpretation of the scoring matrix is that any given cell $\xi_{i,j}$ has a score that represent the best alignment of a suffix of sequence $a_{(1..i)}$ and a suffix of sequence $b_{(1..j)}$.

For example: aligning Sequence 1: GTGACGACGT, and Sequence 2: TGAGCAGTAC, using linear gap penalty of −2, and match of 2 and mismatch of 0, and sequence to sequence comparison type, the alignment scoring matrix will be:

	-	G	T	G	A	C	G	A	C	G	T
-	0	←-2	-4	-6	-8	-10	-12	-14	-16	-18	-20
T	-2	0	↖0	-2	-4	-6	-8	-10	-12	-14	-16
G	-4	0	0	↖2	0	-2	-4	-6	-8	-10	-12
A	-6	-2	0	0	↖4	2	0	-2	-4	-6	-8
G	-8	-4	-2	2	2	↖4	4	2	0	-2	-4
C	-10	-6	-4	0	2	4	↖4	4	4	2	0
A	-12	-8	-6	-2	2	2	4	↖6	←4	4	2
G	-14	-10	-8	-4	0	2	4	4	6	↖6	4
T	-16	-12	-8	-6	-2	0	2	4	4	6	↖8
A	-18	-14	-10	-8	-4	-2	0	4	4	4	↑6
C	-20	-16	-12	-10	-6	-4	-2	2	6	4	↑4

The resulting alignment is:

G	T	G	A	C	G	A	C	G	T	-	-
-	T	G	A	G	C	A	-	G	T	A	C
-2	2	2	2	0	0	2	-2	2	2	-2	-2

The sum-of-pairs score of this alignment is: 4, which is the summation of the column scores as per the scoring scheme used. This is mathematically proven to be the optimal alignment (highest score) based on the scoring scheme used.

Time Complexity Analysis

Initialize matrix values: O(n), O(m)
Filling in rest of matrix: O(nm)
Trace back: O(n+m)
If strings are same length, total time $O(n^2)$

DP is efficient since there are $\frac{2n!}{(n!)^2} = O(2^{2n})$ possible alignments that are all searched to find the optimal alignment with the highest score. However, its complexity grows exponentially with the number of sequences k being aligned since each fill-in step requires maximizing over 2^k-1 scores.

k-D Dynamic Programming Algorithm:

Using the Dynamic Programming algorithm described above to align more than 2 sequences will require computational steps and memory space that is exponential with the number of sequences to be analysed. This creates a dimensionality problem, and makes the algorithm

applicable only to a limited number of sequences. Filling a tensor of alignment scoring values will provide the alignment of a combination of the sequences, and the internal values will be the alignment of all sequences together.

For any given set of k sequences S_1, S_2 ... S_k having lengths vector $\rho=\{\rho_1,\rho_2,...,\rho_k\}$ respectively, can be associated with a lattice, $L(S_1, S_2 \ldots S_k)$ in n-dimensional space (from here on, referred to as a tensor). This lattice consists of $(\rho_1\times\rho_2\times...\times\rho_k)$ k-space cells. Each cell corresponds to a set of k symbols, where each symbol belongs to the corresponding sequence. The cell corresponding to the first symbol of all of the sequences is called the origin (where the index is a vector of all zero elements) and the cell corresponding to the last symbol of all the sequences is called sink, where the index is equal $\{\rho_1-1,\rho_2-1,...,\rho_k-1\}$. The resulting alignment can be associated with a path $\gamma(c_1,c_2,..c_l)$ from origin to sink in the lattice L that is passing through cells c_1, c_2 … c_l, where l is the length of the produced alignment. Each c_i represents an index vector in the lattice, of elements equal to k, and the element j in c_i represent the residue on the j^{th} position in the i^{th} sequence.

For 3 sequences (ACGTGCTA, AGGT, and AT), the alignment 3D matrix will be a cube as shown in Figure 14.a and flattened for scoring as shown in Figure 14.b. Figure 14.c shows the needed seven lower indexed neighbours of a cell to be scored.

So for *2* dimensions, there are three possible paths to the lower indexed neighbours in the trace back. With *3* dimensions, there are seven possible paths as shown by the arrows in Figure 14.c. In general, the number of paths to search is $2^k - 1$, which is exponential in k.

Figure 15 shows a *3D* scoring cube, and Figure 16 shows a *5D* scoring hyper-cube. The thick black arrow diagonal is the alignment of all sequences together. Other surface diagonals are shown in red lines, which represent the alignment of pairs of the sequences corresponding to the surface axis of the respective diagonal. Since with the addition of every extra sequence, all existing cells will be multiplied by the length of the new sequence to form the number of cells of the new hyper cube, a dimensionality problem is created. Accordingly, heuristic methods were developed to trade off optimality for speed of execution and reduction of memory space.

The algorithm for *3* sequences is as follows (GUSFIELD, D, 1997), where ξ is the scoring cube, S_i is the sequence number *i* from the three input sequences of lengths n_i:

Initialize boundary cells as:

$\xi(0, 0, 0) = 0$

$\xi(i, j, 0) = \xi(i, j) + (i + j) * \text{gap score}$

$\xi(i, 0, k) = \xi(i, k) + (i + k) * \text{gap score}$

$\xi(0, j, k) = \xi(j, k) + (j + k) * \text{gap score}$

Recurrence for a non-boundary Cell:

For i=0 to n_1 do
  For j=0 to n_2 do
  For k=0 to n_3 do
    If $S_1(i) = S_2(j)$ then c_{ij} = match score else c_{ij} = miss score;
    If $S_1(i) = S_3(k)$ then c_{ik} = match score else c_{ik} = miss score;
    If $S_2(j) = S_3(k)$ then c_{jk} = match score else c_{jk} = miss score;
    d1 = $\xi(i-1, j-1, k-1) + c_{ij} + c_{ik} + c_{jk}$;
    d2 = $\xi(i-1, j-1, k) + c_{ij} + 2 * \text{gap score}$;
    d3 = $\xi(i-1, j, k-1) + c_{ik} + 2 * \text{gap score}$;
    d4 = $\xi(i, j-1, k-1) + c_{jk} + 2 * \text{gap score}$;
    d5 = $\xi(i-1, j, k) + 2 * \text{gap score}$;
    d6 = $\xi(i, j-1, k) + 2 * \text{gap score}$;
    d7 = $\xi(i, j, k-1) + 2 * \text{gap score}$;
    $\xi(i, j, k) = \text{Min}(d1, d2, d3, d4, d5, d6, d7)$;
  Next k
  Next j
Next i

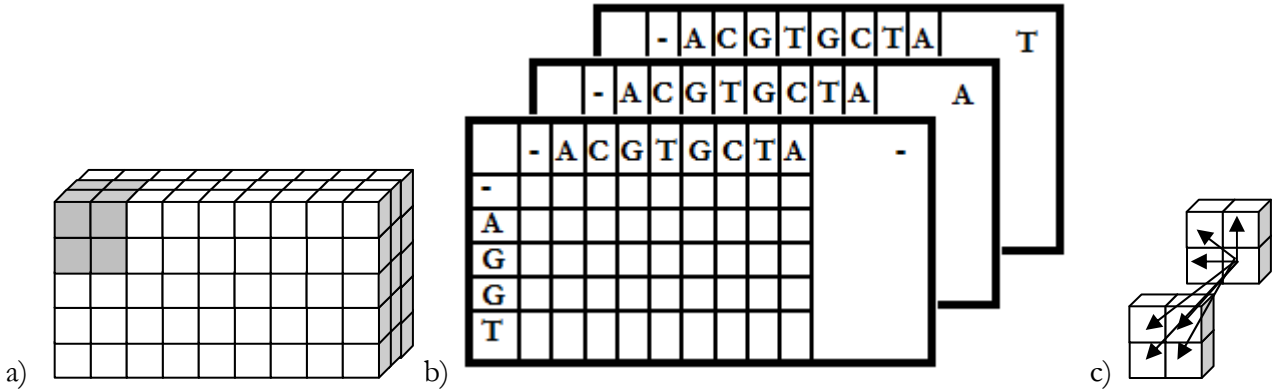


Figure 14: (a) 3D Scoring Matrix. (b) Cube Shape, (c) Flattened for Scoring with Arrows Going to Neighbours.

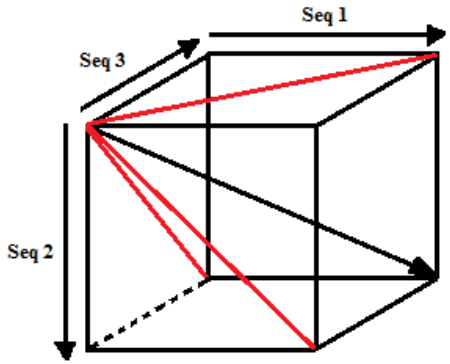


Figure 15: 3D Scoring Cube

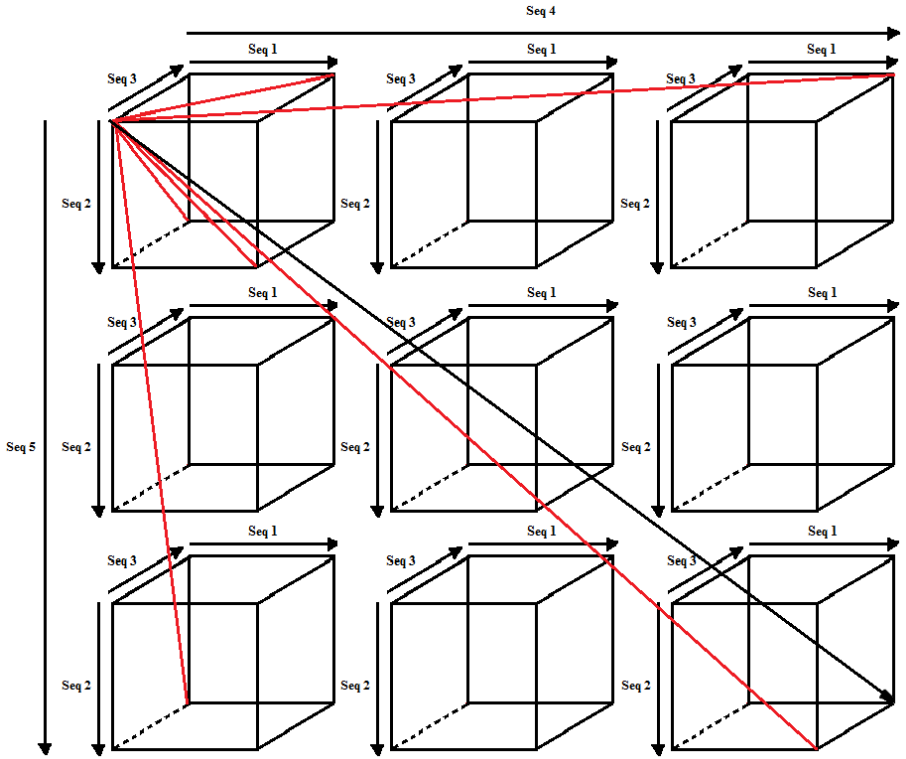


Figure 16: 5D Scoring Tensor

2.4.2.4 Which Method Constructs a Quality Alignment?

This subsection surveys the other heuristic approaches used to construct an MSA and some surveys comparing them to identify a method that constructs a quality alignment. Progressive methods are discussed first, followed by iterative methods, HMMs, Consistency based, SA, and GA, FFT and simultaneous alignments approaches respectively.

Progressive algorithms: build the alignment gradually, by aligning the closest sequences first then successively adding more distant sequences to the results until a final alignment is constructed (DO, CB, Katoh, K, 2000). This method is very popular. Below is the list of algorithms that use this method. Generally the global alignment methods perform better than the local alignment methods. The performance in general is affected by the number of sequences, the degree of identity of the sequences, and the number of insertions/deletions in the alignment (MOUNT, DW, 2004).

One of the objectives of MSA is to produce the phylogenetic tree for a given sequence. It is not possible to provide a sequence and get its fully annotated phylogenetic tree for two reasons. First, an algorithm that considers all possible multiple sequence alignments and then, for each alignment, all possible phylogenetic trees and picks out the best one, would be computationally prohibitive. That is why most phylogenetic programs work on previously aligned sequences. Second, the result is always strongly influenced by the criteria that are used to define the best tree. There are 3 main classes of phylogenetic methods for constructing phylogenies from sequence data:

- Methods directly based on sequences:
    - Parsimony: Find a tree that requires a minimum number of changes to explain the data.
    - Maximum likelihood: Find a tree that maximizes the likelihood of data
- Methods indirectly based on sequences :
    - Distance matrices (UPGMA, Neighbour Joining) find a tree that accounts for estimated evolutionary distances.

Parsimony Based approaches are provided with character-based data, and find a tree that explains the data with a minimal number of changes. Parsimony is the most popular method for reconstructing ancestral relationships (KOLACZKOWSKI, B, Thornton, JW, 2004). It

allows the use of all known evolutionary information in a tree. The disadvantages of Parsimony Methods are (FELSENSTEIN, J, 1978):

- If the evolutionary clock is not constant, the procedure generates results which can be misleading.
- Within practical computational limits, this often leads to the generation of tens or more "equally most parsimonious trees" which make it difficult to justify the choice of a particular tree.
- Long computation time to construct a tree.

Maximum Likelihood Methods require that sequences at each site be considered independently and the log-likelihood of having these bases is computed for a given topology by using a particular probability model. This log-likelihood is added for all sites, and the sum of the log-likelihood is maximized to estimate the branch length of the tree. This procedure is repeated for all possible topologies, and the topology that shows the highest likelihood is chosen as the final tree (FELSENSTEIN, J, 2004).

UPGMA (Unweighted Pair Group Method with Arithmetic Mean) is the simplest and most straightforward method of tree construction that uses arithmetic averages (LEGENDRE, P, Legendre, L, 1998). Originally, it was deigned to construct taxonomic phonograms, which are trees that reflect the phenotypic similarities between Operational Taxonomic Units (OTUs). It is generally not considered a good algorithm for construction of phylogenetic trees as it relies on the rates of evolution among different lineages to be approximately equal. This means that when one of the OTUs has incorporated more mutations over time, than the other OTU, one may end up with a tree that has the wrong topology. It shouldn't be used in the study of bacterial population biology since this is likely not to be the case in that domain. The method uses a sequential clustering algorithm, in which local homology between OTUs is identified in order of similarity, and the tree is built in a stepwise manner. The two OTUs that are most similar to each other are first determined and then these are treated as a new single 'composite' OTU. Subsequently from among the new group of OTUs (composite and simple), the pair with the highest similarity is identified and clustered. This continues until only two OTUs are left. Slightly different clustering may also be seen when the data is presented to the algorithm in a different order.

Clustering works only if the data has an ultra-metric. Ultra-metric distances are defined by the satisfaction of the three-point condition, which means for any three taxa: dist AC <= max (distAB, distBC) or in words: the two greatest distances are equal, or UPGMA assumes that the evolutionary rate is the same for all branches(MILLIGAN, GW, 1979).

Alternatively, Neighbour Joining (NJ) is a very popular method. It does not make a molecular clock assumption; therefore a modified distance matrix is constructed to adjust for differences in evolution rate of each taxon. It produces an un-rooted tree (ATTESON, K, 1997). It also assumes an additive property, i.e. the distance between pairs of leaves = sum of lengths of edges connecting them. Like UPGMA, constructs tree by sequentially joining sub-trees (GASCUEL, O, Steel, M, 2006).

The advantages of distance matrix methods are that they are easy and quick to perform, and fit sequences having high similarity scores. However, their disadvantages are that the sequences are not considered as such (loss of information), all sites are generally treated equally (they do not take into account differences of substitution rates), and they are not applicable to distantly divergent sequences.

The guide tree used by pileup and CLUSTAL (described below) should never be used to infer phylogeny. It has been derived from the distances between pair-wise aligned sequences and these distances are not necessarily the same as the distances between sequence pairs taken from the multiple sequence alignment. Progressive global alignment MSA tools include:

- **CLUSTAL W** (THOMPSON, JD, Higgins, DG, Gibson, TJ, 1994) performs the best in alignment of orphans against a closely related family (reference 2 in (THOMPSON, JD, Plewniak, F, Poch, O, 1999) where the comparison is affected by the existence of other orphans in the family and the size of the family. CLUSTAL W is the second highest with SAGA and after PRRP, with Global alignment performing much better than local alignment in the twilight zone (reference 1 in (THOMPSON, JD, Plewniak, F, Poch, O, 1999)) of evolutionary relatedness. It is worth noting that all programs aligned better with medium length and long sequences than short DNA sequences. The only exception was CLUSTAL W algorithm that improved traditional progressive methods, but for long sequences. This phenomenon can be explained by the usage of an alternative

Neighbouring-Joining algorithm for a guide tree construction, with sequence weighting, as well as by using position-specific gap-penalties. It offers the choice of a residue comparison matrix depending on the degree of identity of the sequences. CLUSTAL X offers a GUI for CLUSTAL W(THOMPSON, JD, Gibson, TJ, Plewniak, F, Jeanmougin, F, Higgins DG, 1997).

- **MUSCLE** (Edgar, 2004) aligns 5,000 sequences of average length 350 in 7 minutes on a current desktop computer, requiring less time than all other methods. MAFFT, MUSCLE and T-Coffee produce, on average, the most accurate alignments, with 6% more positions correctly aligned than CLUSTAL W. It calculates the evolutionary distance between each pair of sequences. Then, it uses a resulting distance matrix to cluster the sequences using UPGMA (SNEATH, PHA, Sokal, RR, 1973), giving a binary tree. The tree is then used to construct a progressive alignment (HOGEWEG, P, Hesper, B, 1984), (FENG, DF, Doolittle, RF, 1987) by aligning profiles of the two sub-trees at each internal node. The study in (ESSOUSSI, N, Boujenfa, K, Limam, M, 2008) shows that MUSCLE version 3.6 scored higher in aligning balibase data sets of identity equals 20% and 30% and Homstrad datasets of identity equals to 10% and 30%, as compared to ClustalX version 1.81, Align-m version 2.3, T-Coffee version 3.93, SAGA version 0.95, ProbCons version 1.08, MAFFT version 5.743 and DIALIGN version 2.2.1.
- **T-Coffee** (NOTREDAME, C, Higgins, DG, Heringa, J, 2000) allows the combination of a collection of multiple/pair-wise, global or local alignments into a single model. It is a greedy progressive method, it allows for much better use of information in the early stages, to rectify the problem with progressive methods of having errors happening early in the alignment and not being able to rectify it later. It also allows estimating the level of consistency of each position within the new alignment with the rest of the alignments.
- **Multalign**, **Pileup**, and **Multal** are global progressive alignment programs. Multalign (BLANCHETTE, M, Kent, WJ, Riemer, C, Elnitski, L, Smit, AFA, Roskin, KM, Baertsch, R, Rosenbloom, K, Clawson, H, Green, ED, Haussler, D, Miller, W, 2004) and Pileup (DEUPREE, JD, Scofield, MA, Bylund, DB, 2000) construct a guide tree using UPGMA method. Multal as defined in (TAYLOR, WR, 1987) and (TAYLOR, WR, 1988) doesn't work for alignments of orphan to closely related family, a closely related equidistant divergent families, N/C terminal extensions, or

internal insertions (ref 2, 3, 4, 5 in (Balibase Website) respectively), because it frequently excludes the divergent orphans as unrelated sequences(THOMPSON, JD, Plewniak, F, Poch, O, 1999). This program uses a sequential branching method and then adds the next closest sequences. Pileup8 is the only global alignment program that ranks equally with local alignment programs in aligning sequences with large N/C terminal extensions (different lengths) - reference 4 in (Balibase Website).

- **Match-Box** is a progressive method that proposes protein sequence alignment tools based on strict statistical criteria. It assembles the alignment in a sequence-independent manner by combining segment pairs ordered by their score until all residues are included in the alignment. It is similar to DiAlign method described in the next section (DEPIEREUX, E, Baudoux, G, Briffeuil, P, Reginster, I, De Bolle, X, Vinals, C, Feytmans, E, 1997).
- **PIMA** is a local progressive alignment program that offers two alignment methods: Maximum Linkage (MLPIMA) and Sequential Branching Algorithms (SBPIMA) to decide the order of alignment (SMITH, RF, Smith, TF, 1992).

Iterative techniques: often offer improved alignment accuracy, successfully learning and improving the alignment if enough information is provided in the dataset. Iterative strategies are based on (GOTOH, O, 1996) and (NOTREDAME, C, Higgins, DG, 1996). The main disadvantage is that the iteration process may sometimes be unstable in the presence of a bias in the sequence set, such as in a single orphan sequence, the iteration may diverge away from the correct alignment. The choice of the fundamental algorithm implemented at each iteration is very crucial in the performance of the iterative methods. Iterative methods also incur heavy time penalty. ClustalX require 161 seconds of CPU time to align 89 histone sequences consisting of 66-92 residues, while Dialign requires 13649 seconds, and PRRP 13209 seconds. Iterative MSA tools include:

- **DiAlign** (MORGENSTERN, B, Dress, A, Wener, T, 1996) is the best local iterative alignment program for gap-free segment alignment. It achieves best performance in aligning sequences with high N/C terminal extensions (reference 4 in (Balibase Website)) and internal insertions (reference 5 in (Balibase Website)). It constructs multiple alignments based on segment-to-segment comparison rather than the

previously used residue-to-residue comparison. Then, it incorporates the segments into multiple alignments using iterative procedures.

- **PRRP** is the highest scoring program in the twilight zone of evolutionary relatedness, where there is only 20-35% sequence identity (reference 1 in (Balibase Website)). It correctly aligns 72% of the total residues. It performs the highest on aligning approximately equidistant divergent families (< 25% ID), composed of highly related sequences (> 25% ID) – reference 3 in (Balibase Website). It is not successful in aligning sequences with high N/C terminal extensions (ref 4 in (Balibase Website)). PRRP uses a hill-climbing algorithm to optimize its MSA alignment score, and iteratively corrects both alignment weights and locally divergent regions of the constructed MSA. It optimizes an already constructed progressive global alignment by a faster method, by iteratively dividing the sequences into two groups that are realigned using a global group-to-group alignment algorithm (GOTOH, O, 1996).
- **Kalign** (LASSMANN, T, Sonnhammer, EL, 2006) applied the same progressive method with the difference in the distance calculations which are based on the Wu-Manber approximate string-matching algorithm, which is an extension to the exact Baeza-Yates-Gonnet algorithm.

Hidden Markov Model (HMM) Algorithms: HMMs are statistical probabilistic models for linear sequence labelling problems (EDDY, SR, 1995). HMMs are used for MSA for global and local alignments, and are considered to be of significant speed-up in computation, especially for sequences that contain overlapping regions. The MSA is represented as a partial ordered graph, consisting of a series of nodes representing possible entries in the columns of the MSA that are well-preserved, i.e. being closer than some threshold, under some metric. Common metrics are: Pair-wise Hamming (edit) distance, multiple alignment score, and Model-based p-value. Some programs use HMMs to construct an MSA as follows:

- **SAM** method lets you create or improve a multiple sequence alignment using hidden Markov models and the SAM system. It is used for protein structure prediction (HUGHEY, R, Krogh, A, 1995).
- **MUMMALS** (MUltiple alignment with Multiple MAtch state models of Local Structure) build an MSA using probabilistic consistency. It is based on pair-wise alignment Hidden Markov Models (HMMs) with multiple match states that

describe local structural information without exploiting explicit structure predictions (PEI, J, Grishin, NV, 2006).

- **HMMER**: is the core utility for **Pfam** and used for protein sequence analysis and for sensitive database search using profile-HMMs (EDDY, SR).
- **HMMT** is a global iterative alignment program. It is excluded from aligning N/C terminal extension or internal insertions (ref 4, 5 in the in (Balibase Website) respectively) because they often include a small number of sequences. It doesn't perform well for global alignment for up to 25 sequences. Also, with up to 100 sequences, it doesn't perform better than other global alignment methods. HMMT produce different scores for the same sequences each time the program runs. It is a statistical model of the primary structure consensus of a sequence family. It uses a simulated annealing method to maximize the probability that an HMM represents the sequences to be aligned (MCCLURE, MA, Vasi, TK, Fitch, WM, 1994).

Probcons is a consistency-based progressive alignment while accounting for all suboptimal alignments with posterior-probability-based scoring based on hidden Markov models (HMMs). Consistency-based MSA approaches use shared homology with out-group sequences as a guide for distinguishing between coincidental and real sequence conservation (DO, CB, Mahabhashyam, MSP, Brudno, M, Batzoglou, S, 2005). It demonstrates a statistically significant improvement in accuracy compared to several leading alignment programs like CLUSTAL W, DIALIGN, and T-Coffee, while maintaining practical running times. It uses a double affine insertion scoring. It uses guide tree calculation via semi-probabilistic hierarchical clustering. It optionally uses iterative refinement, and unsupervised Expectation- Maximization (EM) parameter training. Probcons version 1.08 was shown in the study in (ESSOUSSI, N, Boujenfa, K, Limam, M, 2008) to have higher SP scores than ClustalX version 1.81, Align-m version 2.3, T-Coffee version 3.93, SAGA version 0.95, MAFFT version 5.743, MUSCLE version 3.6 and DIALIGN version 2.2.1 for all datasets tested from Balibase and Homstrad benchmarks.

Simulated Annealing: Annealing is the process of heating up a solid until it melts, then cooling it down until it crystallizes into a perfect lattice state, during which the free energy of the solid is minimized. An analogous process in combinatorial optimization is defined as finding a solution with the minimum cost, among a potential very large number of solutions. Simulated annealing is the process of establishing a correspondence between the cost

function and the free energy, and between the solutions and the physical states. This method is able to obtain a solution arbitrary close to an optimum, where increasing the quality of the solution, will require large computational efforts. Using massively parallel execution reduces the computational efforts required. Using homogenous and inhomogeneous Markov Chains, the method is analysed to prove that it converges to globally optimal solutions. However, in practical implementation, the convergence to optimal solution only holds asymptotically and the algorithm becomes an approximation algorithm, based on the cooling schedule, which is a set of parameters determining the efficiency of the algorithm (AARTS, E, Korst, J, 1989). **MSASA** (Multiple Sequence Alignment by Simulated Annealing) uses SA methods to construct an MSA solution. It is better used to refine an existing MSA produced by another method by a series of rearrangements designed to find more optimal regions of alignment space than the one the input alignment already occupies (KIM, J, Pramanik, S, Chung, MJ, 1994).

Genetic Algorithms: A genetic algorithm maintains a population of candidate solutions for the problem at hand, and applies the evolutionary cycle. It evolves by iteratively applying a set of stochastic operators such as: selection (replicates the most successful solutions found in a population at a rate proportional to their relative quality), recombination (decomposes two distinct solutions and then randomly mixes their parts to form novel solutions), and mutation (randomly perturbs a candidate solution). **SAGA** is an MSA tool using a genetic algorithm (NOTREDAME, C, Higgins, DG, 1996). It is considered the second highest with ClustalX and after PRRP, with Global alignment performing much better than local alignment in the twilight zone of evolutionary relatedness (reference 1 in Balibase), and in aligning orphans to closely related families (ref. 2 in Balibase). It comes first in aligning families of related sequences (ref 3 in Balibase). It is not successful in aligning sequences with high N/C terminal extensions (ref 4 Balibase). N/C terminals in a sequence are the left (start) and right (end). It initially uses a genetic algorithm to select from an evolving population the alignment which optimizes the COFFEE objective function (OF). T-Coffee Replaces the COFFEE objective function that had been implemented in SAGA. RNA equivalent to SAGA is implemented in RAGA (NOTREDAME, C, O'Brien, EA, Higgins, DG, 1997).

MAFFT is a multiple sequence alignment based on Fast Fourier transform (KAZUTAKA, K, Hiroyuki, T, BMC Bioinformatics). It offers three different levels of sensitivity, 1) FFT-

NS-2, which is rough and is a progressive method; 2) FFT-NS-i, which is accurate and good for sequences of variable lengths, and it is an iterative method; 3) G-INS-i and F-INS-i, which are the most accurate and good for sequences of similar lengths.

Figure 17 shows a schematic diagram of the iterative and progressive methods as surveyed in (THOMPSON, JD, Plewniak, F, Poch, O, 1999). As shown in the figure, some methods overlap different algorithmic classes.

Simultaneous Alignment: These techniques are extremely CPU and memory-intensive approaches. One of the techniques used to decrease the complexity of the dynamic programming algorithm for multiple sequence alignment is the "sum-of-pairs" algorithm. This reduces the computation steps to the alignment of every pair of sequences (on the surfaces of the cube), and then sums the alignment scores to align all the sequences on an internal diagonal that crosses all the dimensions producing the total alignment. The sum-of-pairs score ς is defined as shown in Equation 3.

Equation 3: Sum-of-Pairs Score

$$\varsigma(\gamma) = \sum_{\forall 1 \le i < j \le k} \varsigma(\tilde{\gamma}_{ij})$$

Where $\varsigma(\gamma_{ij}) = \sum_{l=0}^{|\gamma_{ij}|} \begin{cases} f(S'_i(l), S'_j(l)) & S'_i(l) \quad and \quad S'_j(l) \in \mathrm{N} \\ g & S'_i(l) \quad or \quad S'_j(l) \in \{-\} \\ 0 & S'_i(l) \quad and \quad S'_j(l) \in \{-\} \end{cases}$ and $|\gamma_{ij}|$ is the length of the path γ_{ij}, and S'_i, S'_j are the aligned *i* and *j* sequences corresponding to that path, with $S'_i(l), S'_j(l)$ representing the l^{th} column of the alignment.

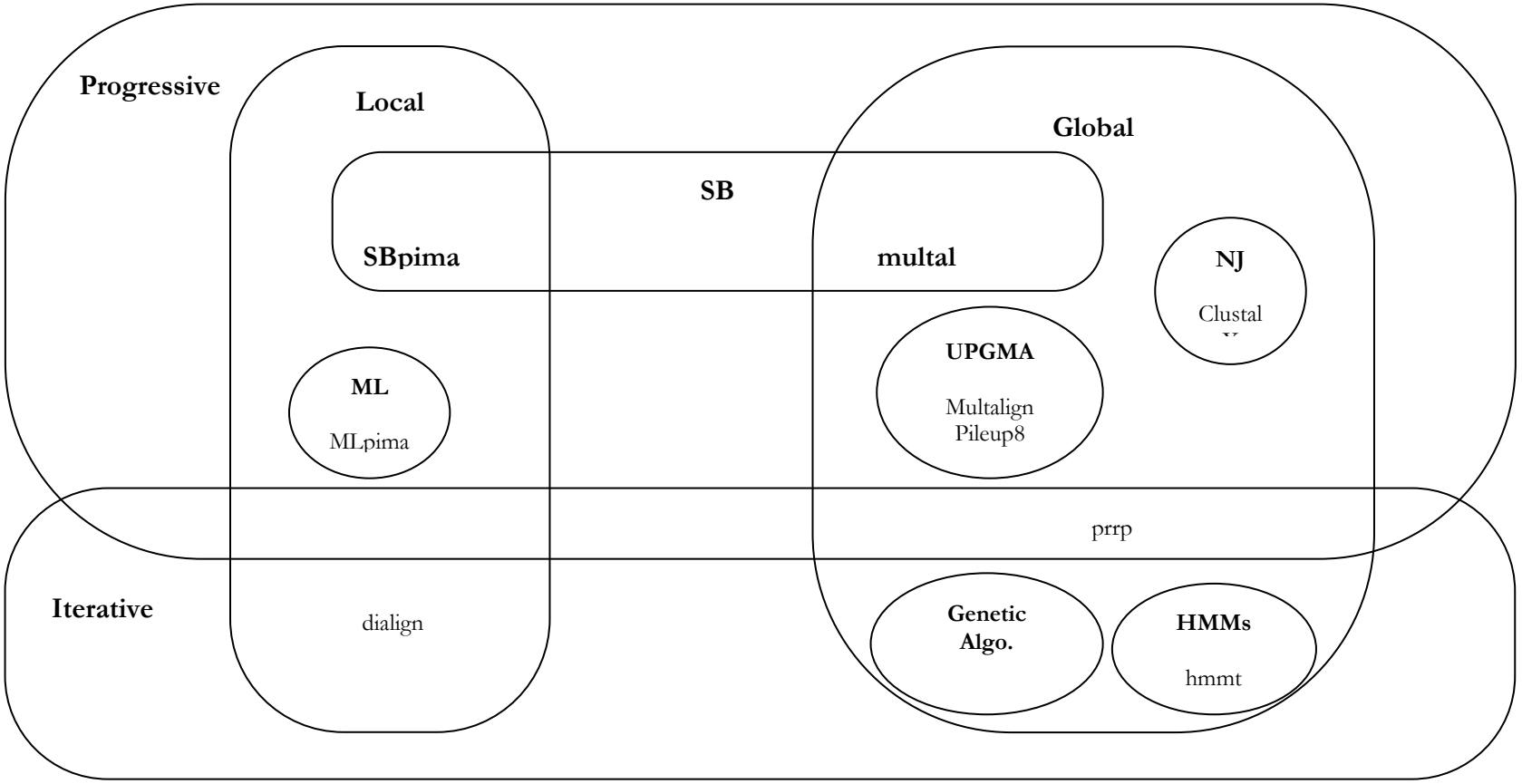


Figure 17: Schematic showing the relation between the progressive and iterative alignment methods and algorithm

The problem of this algorithm is that it calculates a weighted sum of its projected pair-wise alignments. Other methods fit biological intuition more closely. Some Simultaneous Alignment tools are described below:

- **MSA:** is a heuristic based on (LIPMAN, D, Carrillo, H, 1988) that is explained further in Chapter 5. It is a near optimal multiple sequence alignment program that is using tighter bounds than those used in (LIPMAN, D, Carrillo, H, 1988), and does not guarantee a mathematical optimum. The program works best when it is given sequences of approximately equal length that are thought to be globally related. Using MSA to align sequences with only local similarities (e.g. three full length sequences and one fragment) is likely to result in unacceptable time and memory usage (LIPMAN, DJ, Altschul, SF, Kececioglu, JD, 1989),.
- **DCA**: is a simultaneous divide and conquer algorithm that aims to produce multiple alignments which minimize the sum-of-pairs alignment score based on the (Lipman, 1988) algorithm. It recursively cuts the alignment matrix vertically between any two consecutive alignment sites j_0 and j_{0+1}, producing two shorter score-optimal alignments. Then maximize sum-of-pairs score by optimally (using dynamic programming) alignments for the prefix and for the suffix sequences. Finally, we construct the original sequences by concatenation (STOYE, J, Moulton, V, Dress, AWM, 1997).

Integer Programming MSA Methods: The work in (KECECIOGLU, J, 1993) has formulated the MSA into an integer programming method to optimize a Maximum Weight Trace (MWT) objective function. The MSA is represented by a complete k-partite graph where each vertex represents a point in the hyper-cube lattice connecting the residues from the input sequences, and edges connect each residue from one sequence to all other residues in other sequences, forming the complete k-partite graph, where the k sets are the k sequences. Refer to section 4.3 for more on the k-partite graph. The path for a final alignment is formed by connecting the edges from the first residue in the first sequence (first vertex in the first set) to a residue in the second sequence, till the last sequence, and repeat the process to all other residues in the sequences, while skipping some residues, or shifting in one direction, i.e. no lines in the final path crosses. The branch and bound algorithm presented in (KECECIOGLU, J, 1993) maximized the score of each vertex included in the

alignment, through sum-of-pairs score assigned as weights to the edges included in the final alignment.

2.4.2.5 Carrillo-Lipman Search Space Reduction Method:

This section considers a search space reduction technique for the dynamic programming approaches to MSA. Carrillo and Lipman (LIPMAN, D, Carrillo, H, 1988) observed that the score of the projection of an optimal MSA into any of its sequence pairs must be no greater than the score of their pair-wise alignment. Therefore, the dynamic programming optimal MSA is based on the scores of its pair-wise projections on the surfaces of the hyper-cube of the scoring lattice as explained in section 2.4.2.3. The idea is to define bounds on the region around the hyper diagonal by the projections of each pair-wise alignment on the hyper-diagonal. Figure 18 shows three pair-wise alignments of the sequences stretched on the axes of the common surface. These pair-wise alignments are marked with dashed red lines on the surface of the cube as reflected by the sources of light labelled as (l_1, l_2, l_3). The figure shows the projected optimal multiple sequence alignment marked with solid red path through the middle (diagonal) of the scoring cube. The sources of light create shadows of the pair-wise alignments on the optimal multiple sequence alignment.

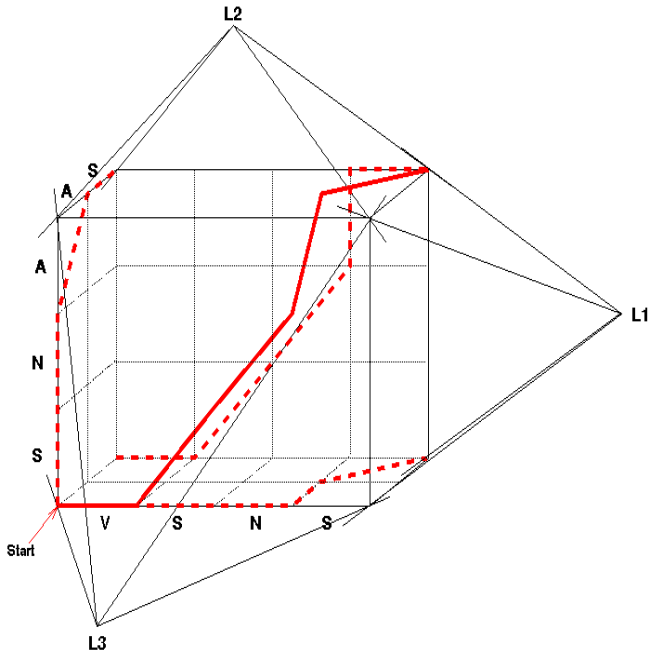


Figure 18: Three Pair-wise Projections of the Alignment

The Carrillo-Lipman bounds are defined as shown in Equation 4.

Equation 4: Carrillo-Lipman Bounds Inequality

$$\sum_{\forall 1\le i<j\le k} \varsigma(\tilde{\gamma}_{ij}^{h}) - \sum_{\substack{\forall 1\le i<j\le k \\ (i,j)\neq(k,l)}} \varsigma(\pi_{ij}^{0}) \le \varsigma(\tilde{\gamma}_{kl}^{0})$$

where ς is the sum-of-pairs, which is introduced in Equation 3 in the previous subsection, π_{ij}^{0} is the optimal alignment of any pair of sequences S_i and S_j, and $\tilde{\gamma}_{ij}^{h}$ is a heuristic path for the same sequences. It can be expressed to define the lower and upper bounds as follows:

$$L - U + \varsigma(\pi_{kl}^{0}) \le \varsigma(\tilde{\gamma}_{kl}^{0})$$

where $L = \sum_{1\le i<j\le k} \varsigma(\tilde{\gamma}_{ij}^{h})$ is the sum of scores of all projected heuristics alignments and $U = \sum_{1\le i<j\le k} \varsigma(\pi_{ij}^{0})$ is the sum of all pair-wise optimal alignments.

The Carrillo and Lipman bounds are extended to MSA using bounds on the scores of its projections into any lower dimensional space with two or more dimensions and less than the original dimension. This lower dimension projection defines an n-space polytope where the optimal path can be found.

Carrillo and Lipman bounds have been observed to be over-estimated. Different methods were proposed in the literature to optimize Carrillo and Lipman bounds. The MSA tool in (LIPMAN, DJ, Altschul, SF, Kececioglu, JD, 1989) presented a simultaneous method that is a heuristic variant of the Carrillo and Lipman method, where alignments are scored as the cost of an evolutionary tree instead of the standard sum-of-pairs scoring scheme. The work in (KECECIOGLU, J, 1993) proposed a branch and bound algorithm for a maximum weight trace that merges weighted pair-wise alignments to form an MSA, using the minimum sum-of-pairs alignment problem as a special case. The guaranteed error bounds presented by (GUSFIELD, D, 1993) are efficiently computed by sum-of-pairs bounds and minimum spanning tree bounds methods. The study in (GUPTA, SK, Kececioglu, JD, Schaffer, AA, 1995) explored the lattice efficiently by means of Dijikstra single-source shortest path algorithm. Using the work in (KECECIOGLU, J, 1993), (REINERT, K, Lenhof, HP,

Mutzel, P, Mehlhorn, K, Kececioglu, JD, 1997) formulated an Integer Linear Programming (ILP) branch and cut algorithm. The methods in (STOYE, J, Moulton, V, Dress, AWM, 1997) applied divide-and-conquer techniques by slicing the input sequences into segments that are later aligned using smaller lattice spaces. Using the goal-directed unidirectional search (the A* algorithm), the methods in (LERMEN, M, Reinert, K, 2000) transformed the edge weights without losing the optimality of the shortest path and was able to exclude more nodes from computation than the Carrillo and Lipman bounds. Then (REINERT, K, Stoye, J, Will, T, 2000) combined the divide-and-conquer approach of (STOYE, J, Moulton, V, Dress, AWM, 1997) with the bounding strategies of (REINERT, K, Lenhof, HP, Mutzel, P, Mehlhorn, K, Kececioglu, JD, 1997). The work presented in (ALTHAUS E, Caprara A, Lenhof HP, Reinert K, 2002) generalized the ILP formulation of (REINERT, K, Lenhof, HP, Mutzel, P, Mehlhorn, K, Kececioglu, JD, 1997) using arbitrary gap cost in a branch-and-cut algorithm for MSA. The later method studied the gapped trace polytope structure, namely the convex hull of the incidence vectors of the ILP solutions. Then presented classes of valid inequalities and reduced the number of variables in the ILP model they presented.

The Carrillo and Lipman approach uses the known topologies for a high-dimensional space which all describe the external points of the space, and only projection on these surface points can be measured. The approaches to be described in Chapter 5 utilize MoA-based indexing of the tensor internals, and attempt to provide measures that can be used to define the bounds on the band that can be scored to reduce the search space.

2.4.3 Are Existing Methods Enough?

The most popular MSA methods are mostly progressive or iterative. These methods are fast, especially the iterative methods. However, most of them assume minimum percent identity of approximately 40% for proteins and approximately 70% for DNA. The resulting alignments are acceptable for families of moderately diverged sequences. Otherwise, a much higher likelihood of errors in the alignment will be expected since these methods can easily run into local optima like any hill-climbing bottom-up methods. Moreover, the discussed global methods require the sequences to be related over their whole length or at least most of it, with the exception of DiAlign (SUBRAM, AR, Kaufmann, M, Morgenstern, B, 2008). Some methods like PRRP (GOTOH, O, 1996), HMMs (EDDY, SR, 1995) and SA (KIM, J, Pramanik, S, Chung, MJ, 1994) are used to refine an MSA produced by another method.

In addition, these methods are sensitive to the order in which the sequences are input. A different ordering of sequences will produce different alignments. This is due to the fact that they calculate a guide tree based on that order. Progressive methods depend on pair-wise alignments, which is less sensitive than simultaneous alignment. This is because pairs of already aligned subfamilies (or closely related sequences) are calculated first, and there is usually more than one optimal alignment of the pairs and the choice of one of them might not be the optimal for the other pair-wise alignments nor have the highest biological relevance. Furthermore, the pair-wise alignments are dependent on the parameters used in the calculations and the parameter changes will not be reflected in the resulting MSA optimal alignment. The pair-wise alignments cause bias (Golubchik, 2007) in the positioning of gaps and statistical uncertainty in the produced conclusions.

Moreover, existing MSA methods assume some knowledge about the MSA outcome, like using annotations and biological knowledge in the objective function, or adding homologs or profiles to the sequences dataset. These methods also assume conserved order of aligned residues, with the exception of ABA (RAPHAEL, B, Zhi, D, Tang, H, et al., 2004), ProDA (PHUONG, TM, Do, CB, Edgar, RC, et al., 2006), TBA (BLANCHETT, e M, Kent, WJ, Riemer, C, et al., 2004), and MAUVE (DARLING, AC, Mau, B, Blattner, FR, et al., 2004). In all these methods care must be exercised in choosing scoring matrices and penalties.

Simultaneous alignments on the other hand do not have problems with local optima. These methods are more robust against parameter changes. The quality of alignment increases as more members of the same family are aligned with sequences from outside the family.

Chapter 3: Data Partitioning for MSA: A Master/Slave Approach

In this Chapter the MSA problem, and more specifically, the dynamic programming algorithm for its solution is expressed using the MoA formalism (MULLIN, LR, 1988). The rationale for using MoA here is that it allows the solution to be expressed independent of the number of sequences, k. Furthermore, with such a formulation, we can attempt to parallelize the solution to distribute processing on HPCs or computer clusters. The key issue here is how to partition the problem sensibly amongst the available processors. The major contribution of this Chapter is a scheme for achieving this data partitioning which does it independent of k, the problem dimensionality. By using the MoA code library described in the last Chapter, we achieve a general-purpose, sequential implementation for MSA which does not depend on the dimensionality of the problem. The key properties that we achieve with this formulation are:

1. Invariance of shape and dimension. For example, there are no static loops, or static nesting of loops, that depend on a fixed dimension k.
2. All operations are mapped to index transformations. In particular, the scoring recurrence used by the dynamic programming algorithm requires the expression of the neighbourhood of a point in a k-dimensional lattice. This is an original contribution of this thesis.

In order to achieve a distributed processing scheme, we need to achieve a partitioning of the problem, and a means to control the allocation of sub-problems to processors, and to coordinate the required data communication for the dynamic programming recurrence. In this Chapter, we formulate a particular approach to partitioning the problem based on our MoA solution. We then analyse the dependencies and communication requirements for this partitioning scheme. We also present a master/slave based implementation for scheduling the work on different processes and for controlling the inter-process communication.

This Chapter is organized as follows. The first section describes the MoA library implementation. The neighbourhood relationships are described here as well. The second section illustrates the scoring scheme based on the MoA neighbourhood. The third section reveals the partitioning operations. The fourth section describes the dependency model. The

fifth section explains the details of the trace back for constructing the alignment for the distributed implementation. The sixth section explains the scheduling and load balancing methods. The seventh section exposes implementation details and the complexity analysis. The eighth section presents the simulation experimental results. The ninth section concludes the work presented in this Chapter.

3.1 MoA Library Implementation

The MoA library is implemented in ANSI C as a set of functions. It is compiled under both Linux and UNIX. There is an equivalent Microsoft © MFC C++ library that runs under Windows© (HELAL, M, 2001). The MoA structure is composed of the fields described in Table 1; the right hand side column highlights how the tensor is interpreted for the MSA problem.

Table 2: MoA Structure Fields for a ξ tensor

Field Name	Description	MoA Symbol	MSA meaning
dimn	Tensor dimension *k*	$\Delta\xi$	K = Number of Sequences
shape	A vector of length *k*, describing the upper bound of each dimension	$\varrho\xi$	sequence lengths
elements_ub	The upper bound of the total elements in the whole tensor, which is calculated as the product of the elements of the shape vector.	$\tau\xi = \pi\varrho\xi$, where π is the product of the vector elements	Number of cells to be scored
elements	A vector of length elements_ub of all tensor elements by row major order.	rav ξ	The actual cells to be scored and all related data. Defined as another structure.
indices	A vector of length elements_ub of all tensor elements' multidimensional indices (where each element is a vector of length *k*) by row major order as the elements vector.	$\gamma'(i, \varrho\xi)$, where i is an arbitrary index vector.	For partitioning references, and position analysis.

The following are a number of examples to illustrate the usage of the library for variable dimensions and shapes.

A 2D tensor:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \end{pmatrix}$$

MoA Structure representation:

Field Name	Value
dimn	2
shape	<4, 5>
elements_ub	20
elements	<1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20>
indices	<<0,0>,<1,0>,<2,0>,<3,0>, <0,1>,<1,1>,<2,1>,<3,1>, <0,2>,<1,2>,<2,2>,<3,2>, <0,3>,<1,3>,<2,3>,<3,3>, <0,4>, <1,4>,<2,4>,<3,4>>

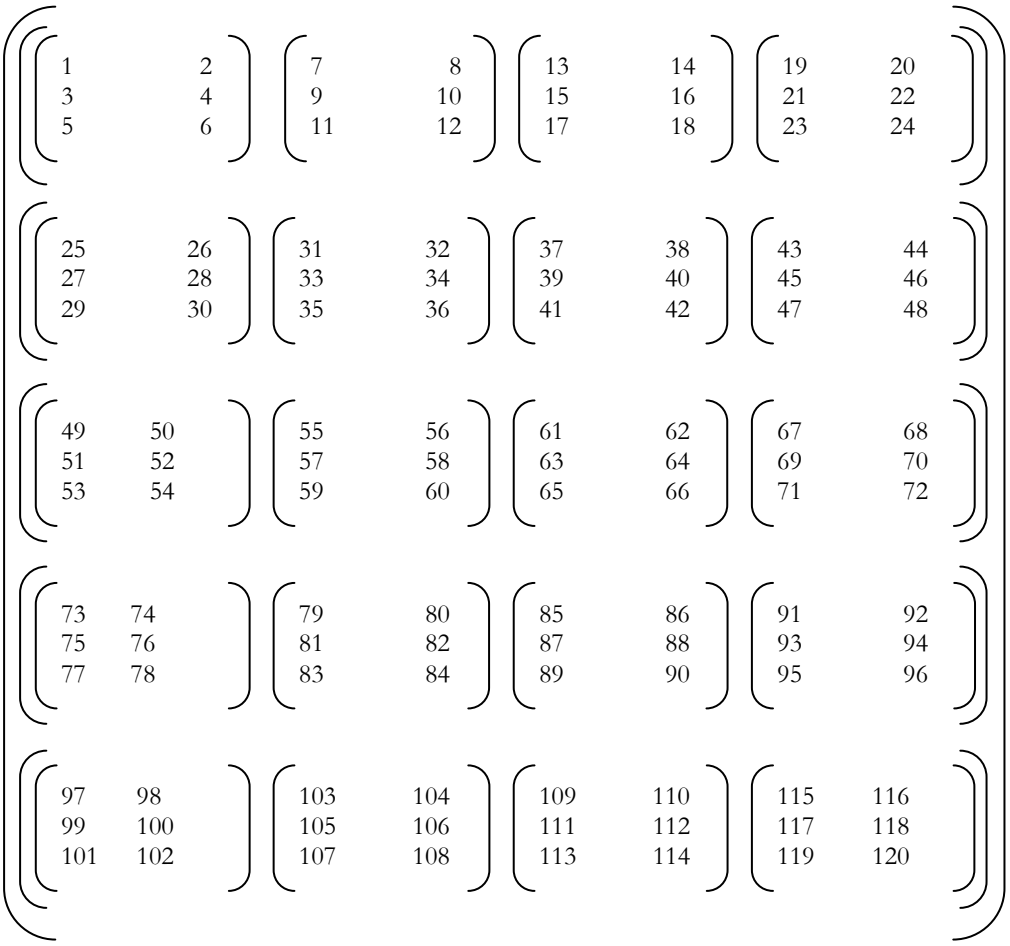
A 3D tensor:

$$\left(\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}\begin{pmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{pmatrix}\right)$$

MoA Structure representation:

Field Name	Value
dimn	3
shape	<3, 3, 2>
elements_ub	18
elements	<1, 2, 3, .., 17, 18>
indices	<<0,0,0>,<1,0,0>,<2,0,0>, <0,1,0>,<1,1,0>,<2,1,0>, <0,2,0>,<1,2,0>,<2,2,0>, <0,0,1>,<1,0,1>,<2,0,1>, <0,1,1>,<1,1,1>,<2,1,1>, <0,2,1>, <1,2,1>,<2,2,1>>

A 4D tensor:


1 2 7 8 13 14 19 20
3 4 9 10 15 16 21 22
5 6 11 12 17 18 23 24
25 26 31 32 37 38 43 44
27 28 33 34 39 40 45 46
29 30 35 36 41 42 47 48
49 50 55 56 61 62 67 68
51 52 57 58 63 64 69 70
53 54 59 60 65 66 71 72
73 74 79 80 85 86 91 92
75 76 81 82 87 88 93 94
77 78 83 84 89 90 95 96
97 98 103 104 109 110 115 116
99 100 105 106 111 112 117 118
101 102 107 108 113 114 119 120

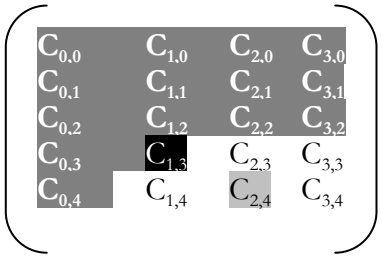
MoA Structure Representation:

Field Name	**Value**
dimn	4
shape	<2, 3, 4, 5>
elements_ub	120
elements	<1, 2, 3, 4, 5, ..., 119, 120>
indices	<<0,0,0,0>,<1,0,0,0>,<0,1,0,0>,<1,1,0,0>,<0,2,0,0>,<1,2,0,0>, <0,0,1,0>,<1,0,1,0>,<0,1,1,0>,<1,1,1,0>,<0,2,1,0>,<1,2,1,0>, <0,0,2,0>,<1,0,2,0>,<0,1,2,0>,<1,1,2,0>,<0,2,2,0>,<1,2,2,0>, <0,0,3,0>,<1,0,3,0>,<0,1,3,0>,<1,1,3,0>,<0,2,3,0>,<1,2,3,0>, <0,0,0,1>,<1,0,0,1>,<0,1,0,1>,<1,1,0,1>,<0,2,0,1>,<1,2,0,1>, <0,0,1,1>,<1,0,1,1>,<0,1,1,1>,<1,1,1,1>,<0,2,1,1>,<1,2,1,1>, <0,0,2,1>,<1,0,2,1>,<0,1,2,1>,<1,1,2,1>,<0,2,2,1>,<1,2,2,1>, <0,0,3,1>,<1,0,3,1>,<0,1,3,1>,<1,1,3,1>,<0,2,3,1>,<1,2,3,1>, <0,0,0,2>,<1,0,0,2>,<0,1,0,2>,<1,1,0,2>,<0,2,0,2>,<1,2,0,2>, <0,0,1,2>,<1,0,1,2>,<0,1,1,2>,<1,1,1,2>,<0,2,1,2>,<1,2,1,2>, <0,0,2,2>,<1,0,2,2>,<0,1,2,2>,<1,1,2,2>,<0,2,2,2>,<1,2,2,2>, <0,0,3,2>,<1,0,3,2>,<0,1,3,2>,<1,1,3,2>,<0,2,3,2>,<1,2,3,2>, <0,0,0,3>,<1,0,0,3>,<0,1,0,3>,<1,1,0,3>,<0,2,0,3>,<1,2,0,3>, <0,0,1,3>,<1,0,1,3>,<0,1,1,3>,<1,1,1,3>,<0,2,1,3>,<1,2,1,3>, <0,0,2,3>,<1,0,2,3>,<0,1,2,3>,<1,1,2,3>,<0,2,2,3>,<1,2,2,3>, <0,0,3,3>,<1,0,3,3>,<0,1,3,3>,<1,1,3,3>,<0,2,3,3>,<1,2,3,3>, <0,0,0,4>,<1,0,0,4>,<0,1,0,4>,<1,1,0,4>,<0,2,0,4>,<1,2,0,4>, <0,0,1,4>,<1,0,1,4>,<0,1,1,4>,<1,1,1,4>,<0,2,1,4>,<1,2,1,4>, <0,0,2,4>,<1,0,2,4>,<0,1,2,4>,<1,1,2,4>,<0,2,2,4>,<1,2,2,4>, <0,0,3,4>,<1,0,3,4>,<0,1,3,4>,<1,1,3,4>,<0,2,3,4>,<1,2,3,4>>

As illustrated in these examples, the multidimensional index value is used to define the position of the element in the multidimensional space. It is retrieved by the gamma inverse MoA function. The latter is applied on the flat index iterator from 0 to *elements_ub-1*. In case of partitioning, the indices field is used to store the global multidimensional index in local partitions for referencing. In sequential processing, where position analysis or dependencies are not required, the indices field becomes unnecessary to be calculated.

MoA Neighbourhood and Partitioning

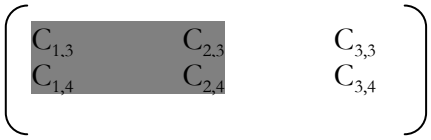
Many array functions are based on a cell's neighbourhood. Using the MoA constructs, direct neighbours are retrieved by decrementing or incrementing the multidimensional index by one in all possible combinations and permutations of its *k* elements. A MoA nested function is developed and presented in this section. Also a simple counting on all dimensions is presented as well. For example:



Neighbours for cell $C_{2,4}$ (highlighted in light gray) have multidimensional index vector of <2, 4> are: $C_{1,3}$, $C_{2,3}$, $C_{1,4}$. Therefore, the MoA function to retrieve the neighbours of the cell at <2, 4> in a 2-dimensional tensor ξ is represented as:

<2, 2> ↑ ((<2, 4> - 1) ↓ ξ)

This nested function drops (↓) the lower indexed cells that are not of interest by subtracting one from the current cell index, thus targeting the cell highlighted in black. The index of the black cell is the amount of cells to drop in each dimension starting from the origin. The *drop* function removes the dark gray shaded region of the tensor that is written in white font, returning the following partition:



The *take* function (↑) retrieves two steps from each dimension from the resulting partition. This returns a matrix with the following cells:

$$\begin{pmatrix} C_{1,3} & C_{2,3} \\ C_{1,4} & C_{2,4} \end{pmatrix}$$

The same function is expanded to 3D to get the neighbours of cell $C_{1,2,1}$ as follows:

$$\left(\begin{pmatrix} \mathbf{C_{0,0,0}} & \mathbf{C_{1,0,0}} & \mathbf{C_{2,0,0}} \\ \mathbf{C_{0,1,0}} & C_{1,1,0} & C_{2,1,0} \\ C_{0,2,0} & C_{1,2,0} & C_{2,2,0} \end{pmatrix} \begin{pmatrix} C_{0,0,1} & C_{1,0,1} & C_{2,0,1} \\ C_{0,1,1} & C_{1,1,1} & C_{2,1,1} \\ C_{0,2,1} & \mathbf{C_{1,2,1}} & C_{2,2,1} \end{pmatrix}\right)$$

$$<2, 2, 2> \uparrow ((<1,2,1> - 1) \downarrow \xi)$$

The *drop* from index <0, 1, 0> returns:

$$\left(\begin{pmatrix} C_{0,1,0} & C_{1,1,0} & C_{2,1,0} \\ C_{0,2,0} & C_{1,2,0} & C_{2,2,0} \end{pmatrix} \begin{pmatrix} C_{0,1,1} & C_{1,1,1} & C_{2,1,1} \\ C_{0,2,1} & C_{1,2,1} & C_{2,2,1} \end{pmatrix}\right)$$

Then taking 2 from each dimension will return the shaded area as follows:

$$\left(\begin{pmatrix} C_{0,1,0} & C_{1,1,0} \\ C_{0,2,0} & C_{1,2,0} \end{pmatrix} \begin{pmatrix} C_{0,1,1} & C_{1,1,1} \\ C_{0,2,1} & C_{1,2,1} \end{pmatrix}\right)$$

The expansion to k-dimension is straightforward as follows:

Equation 5: Lower Neighbours MoA Equation

$$\text{Neighbours}_{\text{lower}} = <2_0, 2_1, ..., 2_{k-1}> \uparrow ((<i_0, i_1, ..., i_{k-1}> - 1) \downarrow \xi)$$

The *getLowerNeighbours* function returns a MoA structure of 2^K cells, where the highest indexed cell is the input cell. Similarly, the *getHigherNeighbours* function works in the opposite direction; however, no subtraction of the input cell index is needed. It can be defined as shown in Equation 6.

Equation 6: Higher Neighbours MoA Equation

$$\text{Neighbours}_{\text{higher}} = < 2_0, 2_1, ..., 2_{k-1} > \uparrow ((< i_0, i_1, ..., i_{k-1} >) \downarrow \xi)$$

The *getHigherNeighbours* function returns a MoA partition having a first cell index equal to the input cell index. In contrast to the *getLowerNeighbours*, it returns a MoA partition having a last index equal the input cell index. The implementation of both functions has to take care of the tensor shape boundaries. The first input to these functions is a vector of size k (the dimensionality) and all its elements are equal to the neighbourhood distance, or the stride. Using these functions to partition the tensor into smaller partitions can be done by utilizing different strides or partition sizes, which will be referred to as *S*.

The MoA tensor is generally too large to be fully processed in memory. In such cases, partitioning allows loading parts of the whole tensor, whether sequentially in one processor, or simultaneously in multiple processors. Partitioning keeps the positions of the parts in the whole tensor known. Hence, the dependency modelling between partitions can be defined. A MoA partition is defined based on a partition size. The partition size is the number of cells to take from each dimension starting from a particular index.

For example, to partition the 2D tensor shown in Figure 19 using partition size $S = 3$, the following procedure is applied: starting from the origin, the first partition is retrieved using the *getHigherNeighbours* function with the origin taken as input. Higher indexed corner cells are defined to have a cell index with one or more (up to all dimensions *k*) elements with a value that is:

- equal to a multiple of *S-1*,
- should be more than the partition index in the corresponding dimension, and
- less than the upper bound at the corresponding dimension.

These cells are used as the indices of the partitions in the next level that are dependent on the current one. Partitions retrieval takes place in breadth first order: all corner cells for all partitions in the same level are traversed in the next level. This method starts a partition at each cell with multidimensional index values of multiples of *S-1* as highlighted in the illustration. Of course, this excludes the cells in the upper bound at any dimension. An

example is cell $C_{0,4}$: this is in the fifth row which is the upper bound of the second dimension. The partition is identified by its starting index, known as the partition index.

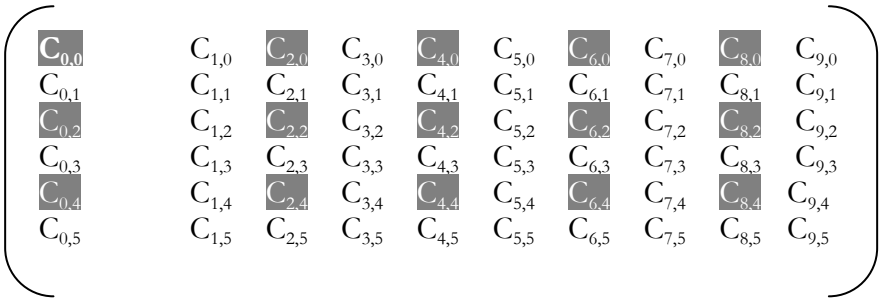


Figure 19: MoA 2D Partitioning Example

Equation 5 and Equation 6 for MoA retrieval of the lower and higher neighbours respectively, are found to be very slow in situations where the whole tensor is too large to be created in the memory of a single processor. Although the whole tensor itself is not created in these functions, the drop function traverses the cells outside the area of interest. In large scale computations and with each step of the scoring, this becomes very expensive. Therefore, the following faster *getLowerNeighbours* and *getHigherNeighbours* functions are found more feasible.

```
getLowerNeighbours (in: cellIndex; in: stride; out: MoAlowerNeighbours)
        MoAlowerNeighbours.dimn = MoAWhole.dimn
        For i from 0 to MoAlowerNeighbours.dimn -1 do
                firstIndex[i] ← cellIndex [i] – stride +1
                If firstIndex [i] < 0 then
                        firstIndex [i] ← 0
                MoAlowerNeighbours.shape[i] ← cellIndex [i] - firstIndex [i] + 1
        End for
        MoAlowerNeighbours.elements_ub ← τ (MoAHigherNeighbours)
        For i from 0 to MoAlowerNeighbours.elements_ub – 1 do
                cellIndex [i] ← γ' (i, MoAlowerNeighbours.shape)
                For j from 0 to MoAlowerNeighbours.dimn -1 do
                        MoAlowerNeighbours.Indicies[i][j] ← MoAlowerNeighbours.Indicies[i][j] + cellIndex [i]
                End For
        End for
End getLowerNeighbours
```

```
getHigherNeighbours (in: cellIndex; in: stride; out: MoAHigherNeighbours)
        MoAHigherNeighbours.dimn = MoAWhole.dimn
        For i from 0 to MoAHigherNeighbours.dimn -1 do
                lastIndex[i] ← cellIndex [i] + stride -1
                If lastIndex[i] > MoAWhole.shape[i] then
                        lastIndex[i] ← MoAWhole.shape[i] – 1
                MoAHigherNeighbours.shape[i] ← cellIndex [i] - lastIndex[i] + 1
        End for
        MoAHigherNeighbours.elements_ub ← τ (MoAHigherNeighbours)
        For i from 0 to MoAHigherNeighbours.elements_ub – 1 do
                cellIndex [i]← γ' (i, MoAHigherNeighbours.shape)
                For j from 0 to MoAHigherNeighbours.dimn -1 do
                        MoAHigherNeighbours.Indicies[i][j] ← MoAHigherNeighbours.Indicies[i][j] + cellIndex
[i]
                End For
        End for
End getHigherNeighbours
```

As previously defined, the function γ' returns the multidimensional index from the flat index *i,* and the dimension and shape vector defined in the MoA structure. Calling the *getHigherNeighbours* with input *cellIndex* = *<2,2>* , stride = 2 (default for direct neighbours), and partition size S = 3, returns the following *MoAHigherNeighbours*:

$$\begin{pmatrix} C_{2,2} & C_{4,2} \\ C_{2,4} & C_{4,4} \end{pmatrix}$$

Similarly a *getPartition* function is defined to be:

```
getPartition (in: partitionIndex, S; out: MoAPart)
        getHigherNeighbours (partitionIndex, S, MoAPart)
End getPartition
```

Calling the *getPartition* with input *partitionIndex <2,2>* and partition size $S = 3$, returns the following *MoAPart*:

$$\begin{pmatrix} C_{2,2} & C_{3,2} & C_{4,2} \\ C_{2,3} & C_{3,3} & C_{4,3} \\ C_{2,4} & C_{3,4} & C_{4,4} \end{pmatrix}$$

Similarly, the *getHigherPartitions* function specifies the dependent partitions and their corresponding processors; it is defined as follows:

```
getHigherPartitions (in: cellIndex; out: MoAHigherPartitions)
        getHigherNeighbours (cellIndex, 2, MoAHigherPartitions)
        For i from 0 to MoAHigherPartitions.elements_ub – 1 do
                Valid_Partition[i] ← 1
                For j from 0 to MoAHigherPartitions.dimn -1 do
                        If cellIndex[j] <> MoAHigherPartitions.Indicies[i][j] and
                        MoAHigherPartitions.Indicies[i][j] + (S-1) – (MoAHigherPartitions.Indicies[i][j] mod S)
                        < MoAWhole.shape[i] then
                            MoAHigherPartitions.Indicies[i][j] ← MoAHigherPartitions.Indicies[i][j] + (S-1) –
                          (MoAHigherPartitions.Indicies[i][j] mod S)
                        Else if cellIndex[j] = MoAHigherPartitions.Indicies[i][j] then
                            MoAHigherPartitions.Indicies[i][j] ← MoAHigherPartitions.Indicies[i][j] -
                           (MoAHigherPartitions.Indicies[i][j] mod S)
                        Else
                          Valid_Partition[i] ← 0
                End For
        End For
End getHigherPartitions
```

The *getHigherPartitions* function calls the *getHigherNeighbours* function. Each cell index is therefore processed in order to add the required strides. This gives the neighbouring partition in the direction of this neighbouring cell. The function returns a MoA structure as defined in Table 2, which describes the dimension, shape, a list of neighbours scores, and the original indices in the scoring tensor. This MoA structure contains the partition indices of all higher neighbouring partitions to the input cell index, labelling the valid and invalid ones. As mentioned in Section 2.2, and will be further explained in Sections 3.3 and 3.4, and also in Sections 4.1 and 4.2, a wave of computation is a collection of partitions that can be processed simultaneously on different processors (with minimum inter-dependency for this chapter, and without dependency across partitions at all in chapter 4). The partition indices are used to get the wave number where it is processed, then the partition number in this wave, and consequently the processor ID.

For the remainder of this thesis, the indices vectors are used in two contexts: 1) to label cells in a tensor, 2) or a partition index (the first cell in a partition of size *S*). In case of a partition indexing, the labels retain the same indexing numbers as the cell indexing in most cases. This means that partitions are drawn or indexed as if they are one cell, where the cell index elements are all multiplied by *S-1*, to be mapped to the actual cell indices including the hidden cells within the partitions. This helps to visualize the neighbourhood of partitions, as if the starting cell of a partition is the only cell it contains.

3.2 MSA Scoring Recurrence

Tensor scoring should be based on index relationships in order to make it invariant of shape and dimension (HELAL, M, Mullin, LM, Gaeta, B, El-Gindy, H, 2007). The dynamic programming recurrence described in Chapter 2 is generalized for K-dimension and arbitrary sequence lengths (shape), using the recurrence shown in Equation 7.

Equation 7: k-D Scoring Recurrence.

$$\xi(i_0, i_1, i_2, ..., i_{k-1}) = \max \begin{pmatrix} G_1 + TS(G_1) \\ G_2 + TS(G_2) \\ \vdots \\ G_{2^k-1} + TS(G_{2^k-1}) \end{pmatrix}$$

where: $\boldsymbol{G_i}$ is neighbour *i* of the current cell being scored; we may have up to 2^k-1 neighbours. Each neighbour index is a vector of *k* elements; each corresponding to an input sequence.

***TS* (G_i)** is the Temporary Score function assigned to each neighbour based on how many multidimensional indices are decremented to get to this neighbour. It is calculated as shown in Equation 8. This equation calculates the sum of the pair-wise substitution scores based on the used scoring scheme, for all pairs of residues on the corresponding axes positions of the neighbour's index. The substitution score is summed only when the pair of the corresponding indices (*j* and *l*) is decremented from the current cell index being scored.

Equation 8: Neighbours Temporary Score

$$TS(G_i)=\left(\sum_{j=0}^{k-2}\sum_{l=j+1}^{k-1}\begin{pmatrix} sub(b_j, b_l) & if\ j\ and\ l\ are\ decremented \\ 0 & otherwise \end{pmatrix}\right)+(gs\times(k-D))$$

Where $sub(b_j, b_l)$ is the substitution score of the residues at dimensions j and l respectively at position of index vector i.

D is the number of decremented indices elements from the current cell index to get to this particular neighbour.

gs is the Gap Score Value. It is multiplied by (K-D). In other words, the Gap Score Value is multiplied by the number of indices that remained unchanged (i.e. they were not decremented to get this neighbour). .

The solution avoids temporary constructs by getting the neighbours of each cell in the neighbours MoA structure. The created MoA structure contains a list of neighbours scores, which are mapped directly to the MoA partition being scored. The approach of creating temporary constructs consumes more memory as compared to having another MoA neighbour's structure that is created once, and sweeps across all cells to retrieve their neighbours, refilling it with the values of the new neighbours of each new cell being computed.

The solution iterates through the partition received by each processor. For each cell, the lower border neighbouring cells scores are retrieved using the *getLowerNeighbours* function described above. These neighbours might be:

- Local cell means it is in the same partition, or another partition computed by the same processor; then its score is retrieved. *TS* is then computed based on how many indices are decremented in the multidimensional index to retrieve this neighbour,
- Remote cell (in another processor), the processor computation thread waits to receive the score from the remote processor

- Lower border cell in the whole un-partitioned scoring tensor; the score gets initialized to the gap score used multiplied by the values in the multidimensional index of the cell.

There are at most 2^k-1 lower neighbours' cells that are required to score a cell. After scoring this cell, there are at most 2^k-1 higher neighbours' cells that depend on this calculated score. Both lower indexed neighbours' cells and higher indexed neighbours can be either local or remote.

3.3 Partitioning MSA using MoA

Parallel processing of sequence alignment dynamic programming algorithm have been limited to pair-wise alignments (2 sequences, or 2 dimensional scoring matrix)(DRIGA, AR, 2002), (CHEN, C, Schmidt, B, 2005) and (CHEN, C, Schmidt, B (2), 2005). In order to achieve parallel processing of MSA dynamic programming algorithm for more than two sequences, we need a partitioning method like the one presented in section 3.1, and we need to study the dependency requirements between partitions. The dependency analysis will decide the appropriate processor scheduling of the partitions. This section contributes a master/slave approach to partitioning and scheduling the k-dimensional scoring tensor of MSA dynamic programming algorithm.

A master process is responsible for 1) partitioning, 2) dependency analysis, 3) scheduling over processors, and 4) managing the trace back processing over the distributed partitions. The partitioning process takes place in breadth first fashion, where partitions that are neighbours on lower indices are retrieved first, then partitions in further depth in the tree is traversed. The partitions that are siblings to each other in the tensor, i.e. on the same breadth level in the tensor graph, forms a wave of computation, that needs to be scored before deeper (higher indexed partitions) can be scored for dependency. The rest of the available processors act as slave processes: 1) they receive partitions, 2) score them, 3) receive dependency requirements, 4) send them to the waiting processors, 5) receive their dependency, and 6) trace back through the partitions. The master process has a partitioning thread, a dependency analysis thread, and a sending thread. The slave processes contain a score computation thread, a receiving thread that buffers all the packets received from the master process and other slave processes, and a sending thread that sends dependency to the waiting slaves' processors.

3.4 Dependency Analysis

To be able to parallelize the score computation, the dependency between the scoring of elements (cells) should be understood in order to communicate the required scores between processors. As analysed in (IYENGAR, AK, 1989), (YAP, TK, 1995), and (CHEN, C, Schmidt, B (2), 2005), the dependency to score each element in the scoring matrix for pair wise alignment, is based on retrieving the calculated score for the top, left, and left-up diagonal, creating a wave-front communication pattern as shown in Figure 23.

So, if every processor scores a row, all processors can initialize the first element of the assigned row. Once the first processor finishes the second element in the first row, the second processor can act on the second element in the second row, and so forth. This will make parallelism increase to the middle diagonal, and then decrease as it approaches the end of the scoring matrix.

The same scheme is followed to partition the alignment tensor in order to maximize parallelism, in a wave-front pattern that can scale with dimension and shape. The MoA higher neighbours function explained in Equation 6 can be used iteratively in a breadth-first traversal, starting from an index vector containing zeros, for the first cell in the tensor (the origin). Then in each retrieved partition, all higher order neighbours partitions can be retrieved to create the next diagonal wave. The first wave will contain one partition starting at the zero-cell, and ending at $<S_0\ S_1\ S_2\ S_3 \ldots\ S_k>$ where S is the specified partitioning size. Then at each higher border corner cell in this partition, the *getHigherNeighbours* function is applied to retrieve the next wave partitions. This procedure in illustrated in the following pseudo code, where the function *doPartitioning* is first called using a wave buffer containing the origin cell, then this fills the next wave buffer and the function is called again, till no more partition is returned in the later wave.

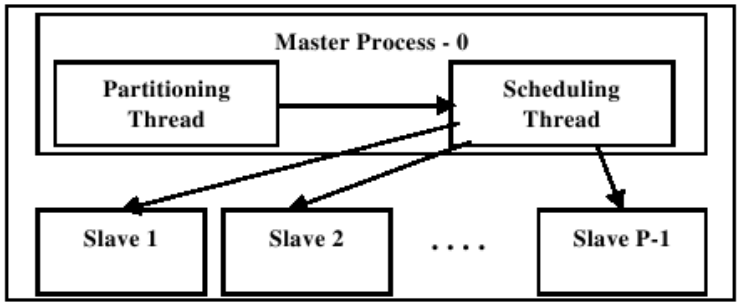


Figure 20: Master/Slave Approach

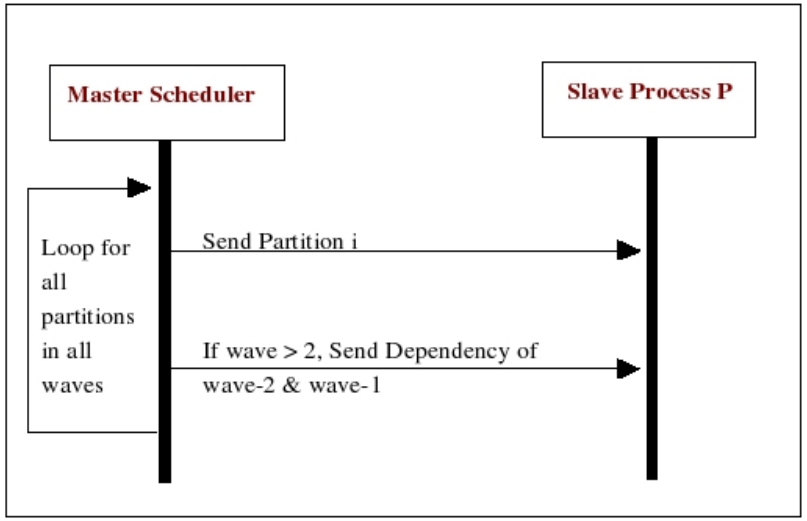


Figure 21: Processes Communication in the Master/Slave Approach

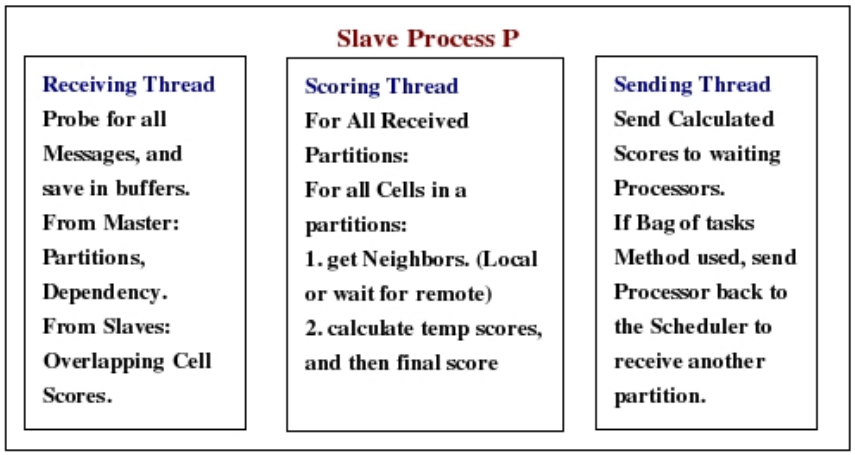


Figure 22: Slave Process

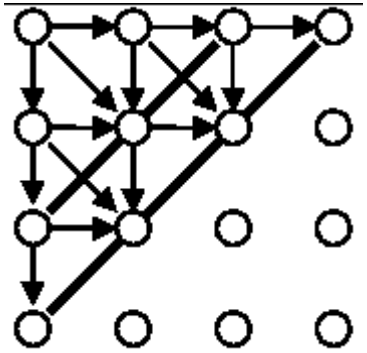


Figure 23: MSA Pair-wise Traditional Wave-Front Dependency.

```
doPartitioning (waveBuffer)
        For each partitionIndex in waveBuffer do
                assignToProcessor (partitionIndex, processors)
                MoANeighb  = getHigherNeighbours (partitionIndex, S)
                CornerCells = getCornerCells(MoANeighb)
                For each cellIndex in CornerCells do
                        If cellIndex is a valid new partitionIndex
                          Add cellIndex to next waveBuffer
                End For
        End For
End doPartitioning
```

A partition of size *S* is then taken from the remaining tensor. The algorithm loops through all partitions created at one wave, and traverses all their higher border corners. A higher border corner is defined to be a cell where one index element is equal a multiple of the partition size *S* that is greater than its correpesponding first element in the partition index, or as the shape boundary of the whole tensor of this particular dimension. The partitions for the next wave are then created, by retrieving all higher neighbouring partitions at these corner cells. The same algorithm repeats for the next wave. The procedure is completed when the last higher order cell in the tensor is reached. In 2D MoA MSA, dependency takes the form of encapsulated squares forming and extra perimeter of partitions around the partition in the previously processed waves. The dependency changes to the illustration shown in Figure 24.

This partitioning process keeps increasing the degree of parallelism from one wave to another. In 3D MSA, dependency takes the shape of enclosed cubes, with inner cubes being scored before the outer ones, as shown in Figure 25. The first dark gray cube is scored first in one wave, and the next wave contains the 2^k-1 neighbouring cubes, coloured in light gray, and then the white wave of cubes. Later waves contain the remaining partitions excluding the ones that were previously partitioned: these are neighbours to other previously traversed partitions.

Generalizing the problem to multiple dimensions requires retrieving the dependency invariant of dimension. In K-dimensions, each internal cell has 2^k-1 lower neighbour cells. This is done using the *lower neighbours* MoA equation described in Equation 5. Then each cell sends its score to the dependent 2^k-1 higher neighbours' cells. This relationship is illustrated

in Figure 26. It is implemented using *OCin (Overlapping Cells incoming)* and OCout *(Overlapping Cells outgoing)* as 2 dimensional arrays of structures corresponding to the lower and higher border cells in all local partitions respectively. These structures are hierarchically indexed by the wave number as the first dimension, where the encapsulating partition belongs, and then the second dimension is the partition index on the second level of the hierarchy, to minimize the search a required cell index within its specific wave index and partition index values.

As shown in Figure 24 and Figure 25, the partitioning is based on retrieving the partitions at distance equals the wave number form the origin. The total number of waves of computations t equals the maximum shape vector element (longest sequence length) divided by one less than the partition size (i.e. S-1), and is calculated as per Equation 9 and Equation 10 The actual number of total partitions P for a specific shape vector all over the waves based on a partition size S is described in Equation 11. This is the product of the total number of partitions created in each dimension using its bound (sequence i length is ρ_i).

Equation 9: Total Number of Waves in the Master/Slave Approach

$$t = \frac{\max(\rho\xi)}{S-1}$$

Equation 10: Number of Partitions per Wave in the Master/Slave Approach

$$P_w = w^k - (w-1)^k$$

Equation 11: Total Number of Partitions all over the Waves

$$P = \prod_{i=0}^{k-1} p_i = \frac{\prod_{i=0}^{k-1} (\rho_i - 1)}{(S-1)^k} \text{ since } p_i = \frac{\rho_i - 1}{S-1}$$

where P_w is the number of partitions in wave number w, and k is the dimensionality or the number of sequences. Table 3 enumerates values from Figure 27; these describe the involved degree of parallelism identified as partitions per wave that can be scored simultaneously (however with some degree of dependence communication among them). In other words, the numbers of partitions that can be scored in parallel, therefore, the number of processors that can be employed effectively.

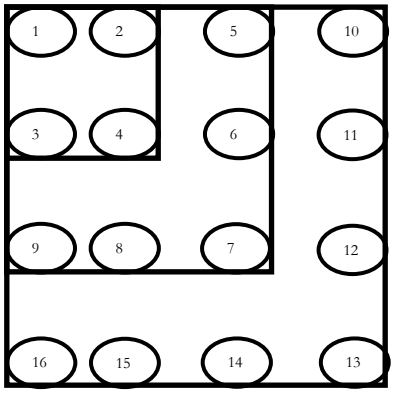


Figure 24: 2D Partitioning

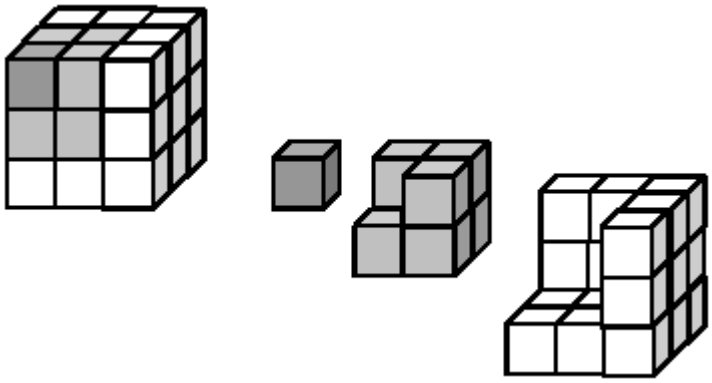


Figure 25: 3D Partitioning

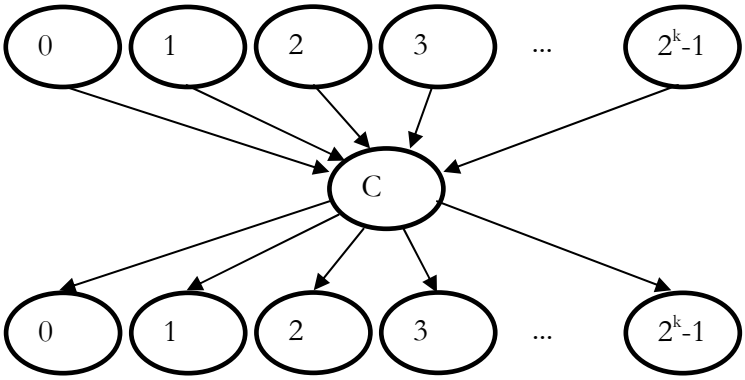


Figure 26: k-D Dependency Model

Table 3: Master/Slave Partitions per Wave for Different Dimensionality.

		Waves									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Dimensions	**2**	1	3	5	7	9	11	13	15	17	19
	3	1	7	19	37	61	91	127	169	217	271
	4	1	15	65	175	369	671	1105	1695	2465	3439
	5	1	31	211	781	2101	4651	9031	15961	26281	40951
	6	1	63	665	3367	11529	31031	70993	144495	269297	468559
	7	1	127	2059	14197	61741	201811	543607	1273609	2685817	5217031
	8	1	255	6305	58975	325089	1288991	4085185	11012415	26269505	56953279
	9	1	511	19171	242461	1690981	8124571	30275911	93864121	253202761	612579511
	10	1	1023	58025	989527	8717049	50700551	222009073	791266575	2413042577	6513215599

This kind of parallelism is exponentially growing in the data size, and allows for a more effective employment of processors. Hence, the complexity of the problem is not reduced by this design. However, with the availability of PC clusters and cheaper high performance machines, the upper bound of the data that can be processed in parallel increases in the magnitude of the available machines. This exponential growth is illustrated in Figure 27.

Based on the selection of the partition size *S*, the amount of computation done per partition and the amount of overlapping cells that should be communicated amongst processors are determined. The optimization of the partition size needs to balance the computation against the communication costs with respect to the metrics of the target machine capabilities. The number of cells in a single partition is calculated as shown in Equation 12.

The cost of dependency communication (overlapping cells between partitions) grows from one wave to another and is a function of $\frac{1}{2}$ the perimeter cells (higher indexed edge cells only in each partition). These cell scores are computed in a partition and sent to up to *k* other processors for dependency. It is calculated as shown in Equation 13. This will return the count of the perimeter cells only encapsulating a partition of size *S-1*.

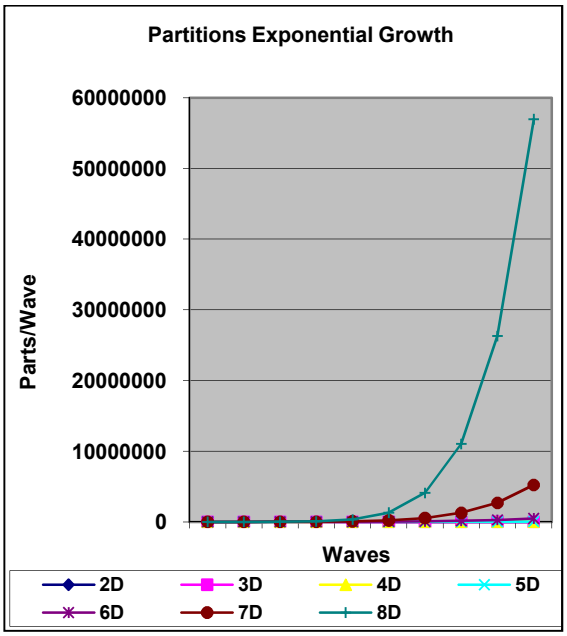


Figure 27: Partitions per Wave Exponential Growth.

Equation 12: Number of Cells per Partition

$$Cells/partition = S^k$$

Equation 13: Number of Overlapping Cells per Partition

$$Cells_{overlapping} / partition = S^k - (S-1)^k$$

Equation 14 calculates the number of overlapping cells C between all partitions over the hyper-cube. This is because on the first dimension (a partitioning of a vector); there is one overlapping cell from each partition in p_0 partitions at this stage, except the last partition, forming C_0. Then as the dimensionality grows, the overlapping cells of the previous dimension is multiplied by the upper bound of the current dimension (shape vector element corresponding to the current dimension). Then, for all the previous partitions of the previous dimensions, there are extra overlapping cells on the intersection of these partitions over the different previous dimensions, as calculated in the second half of the C_i equation. Then all these overlapping cells are summed up together as the dependency all over the scoring tensor. Accordingly the dependency (communication of overlapping cells) varies with the number of partitions created for the tensor, whereas this number is affected by the partition size S chosen.

Equation 14: Communication Cost (Dependency)

$$C = \sum_{i=0}^{k-1} C_i$$

Where

$$C_0 = p_0 - 1$$
$$C_i = C_{i-1} \times \rho_i + \left(\coprod_{j=0}^{i} p_j \right) \times 2^i - 1$$

On the other hand, the computation growth is calculated as the internal cells of the tensor that don't need to be received from another processor. This is shown in Equation 15. It varies with partition size *S* chosen.

Equation 15: Number of Computation Cells in a Partition

$$Cells_{internal} / partition = S^k - (S^k - (S-1)^k) = (S-1)^k$$

Hence, the computation is growing with the dimensionality *k* and partition size *S* exponentially faster than the communication. The numbers of communication and computation cells for 10 dimensions for *S* from 3 to 10 are shown in Table 4 and Table 5 respectively. This growth is illustrated in Figure 28 showing communication and computations cells scores (Comm. and Comp. respectively in the legend) on the y-axis as growing per dimension *k* in the data series and per partition size on the x-axis. It is shown that for small S and k values, the computation and communication costs are almost equal, and as the S and k grow, the computation cost become more than the communication cost. The computation /communication ratio need to be optimized for the targeted high performance machine, as some machine will be faster in either computation or communication.

3.5 Distributed Trace Back

After the partitions have been fully scored and stored in the local disk space of each slave processor, the distributed trace back program starts. Trace back is responsible for searching the scoring tensor to connect the highest scoring cells from the highest indexed cell at the bottom, to the lowest indexed cell (the origin) at the top of the tensor.

Again a master/slave approach is followed. The master process retrieves the largest scoring higher border cell from all higher edge partitions in all processors. This value is then sent to the processor with the highest score to trace back through its partition. If the trace back performed by the slave reaches the lower border edge of this partition, it checks if the next required partition is local. The same procedure repeats till no more local partitions are found. At this point, the slave reports to the master process with the cell index, the path partition found so far. In its turn, the master process sends to the next processor to resume the trace back. The process iterates in the same way till no more partitions in all processors are encountered. At this point, the master process assembles all received partial paths to form the optimal full path and generates the alignment. Figure 16 illustrates the distributed trace back process.

Table 4: Dependency Growth with Partition Size & Dimension

		Partition Size							
		3	4	5	6	7	8	9	10
Dimensions	2	5	7	9	11	13	15	17	19
	3	19	37	61	91	127	169	217	271
	4	65	175	369	671	1105	1695	2465	3439
	5	211	781	2101	4651	9031	15961	26281	40951
	6	665	3367	11529	31031	70993	144495	269297	468559
	7	2059	14197	61741	201811	543607	1273609	2685817	5217031
	8	6305	58975	325089	1288991	4085185	11012415	26269505	56953279
	9	19171	242461	1690981	8124571	30275911	93864121	253202761	612579511
	10	58025	989527	8717049	50700551	222009073	791266575	2413042577	6513215599

Table 5: Computation Growth with Partition Size & Dimension

		Partition Size							
		3	4	5	6	7	8	9	10
Dimensions	2	4	9	16	25	36	49	64	81
	3	8	27	64	125	216	343	512	729
	4	16	81	256	625	1296	2401	4096	6561
	5	32	243	1024	3125	7776	16807	32768	59049
	6	64	729	4096	15625	46656	117649	262144	531441
	7	128	2187	16384	78125	279936	823543	2097152	4782969
	8	256	6561	65536	390625	1679616	5764801	16777216	43046721
	9	512	19683	262144	1953125	10077696	40353607	134217728	387420489
	10	1024	59049	1048576	9765625	60466176	282475249	1073741824	3486784401

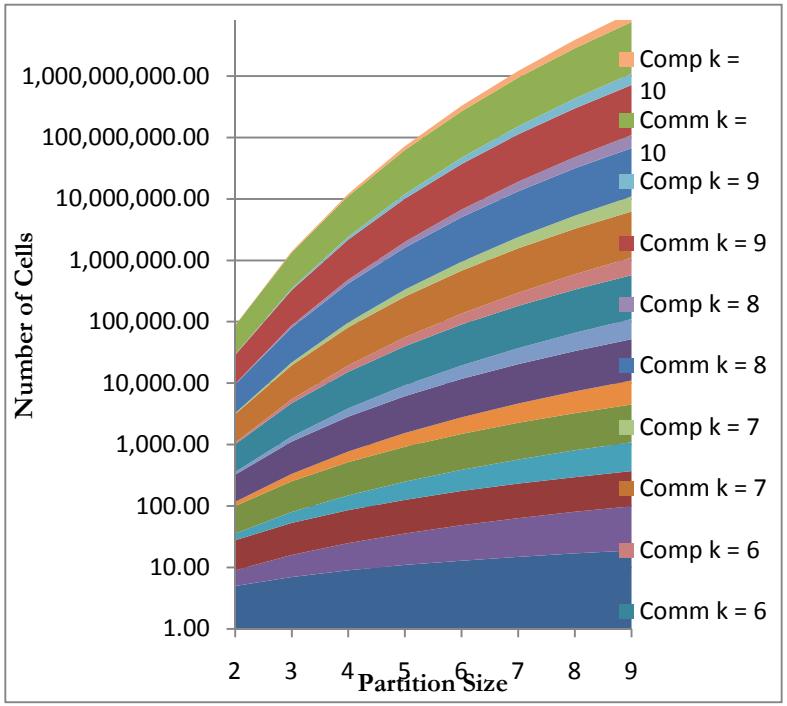


Figure 28: Growth of Communication vs. Computation as per Partition Size and Dimensionality

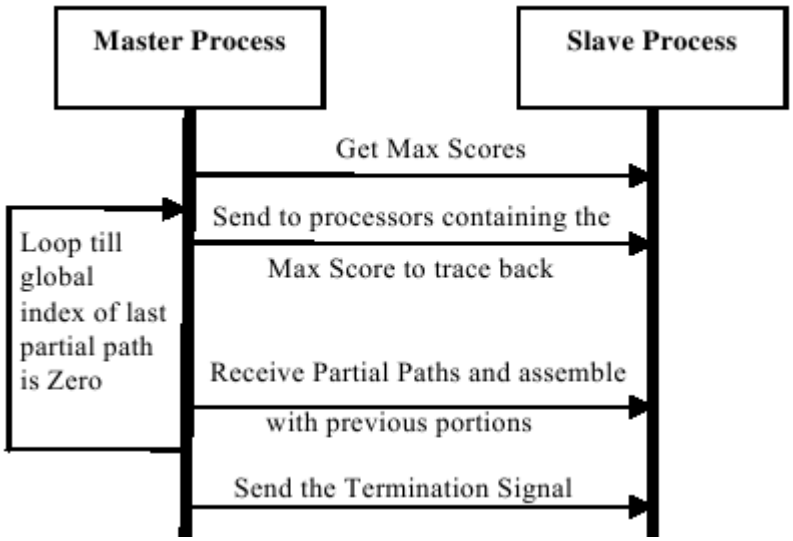


Figure 29: Distributed Trace Back

The trace back function processed by each slave process starts from the cell index received from the master process. Two checks are therefore done:

- The process checks if there is a stored cell index in the previous cell buffer as the maximum neighbour score used to lead to the score of the current cell. If one is found, it is taken as the current cell index.
- Otherwise, all lower border neighbouring cells are checked for the highest score to become the current cell index. A *locateCellIndex* routine is used to locate the partition index, and the corresponding processor id.

Then the alignment path is decided based on comparing the current cell index with the previous one. A non-decremented index element corresponding to a specific dimension means a gap insertion in the corresponding sequence to that dimension. All decremented index elements in any dimension are considered substitutions in the corresponding sequences. This method matches the scoring scheme described in the scoring section.

3.6 Scheduling Scheme and Load Balancing

Having the scoring tensor partitions in section 3.1 and their dependency requirements analysed in section 3.4, a proper scheduling over processors needs to be designed to achieve the highest performance with minimal communication of dependency and delays. This section studies well-established scheduling methods and their properties with respect to the MSA problem.

Three methods of scheduling are considered: bag-of-tasks, round robin, and dependency-based scheduling. The first two methods were implemented in the master/slave approach. The investigations of the third scheduling method led to the design of the P2P design described in the next Chapter. The Bag-of-Tasks method is most suitable to heterogeneous systems where nodes possess diverse computing powers. The Round Robin method is more suitable to homogeneous computing nodes. It is therefore used in the experimentation on the clusters of homogenous computing nodes that were made available to this project. The Dependency-Based Scheduling is optimized to increase locality and decrease data communication.

The bag-of-tasks scheduling technique is based on pushing processors into a queue. These are popped-up later for assignment. At completion, the processor returns to the scheduler by re-pushing itself into the queue waiting for a new assignment. The advantage of this method is that each processor can finish in its own time. The disadvantage is that the scheduler process might remain idle, waiting for processors to come back from an initial assignment in a previous wave.

In the round robin scheduling technique, the master processor retains the scheduler until partitioning all waves uniformly in all available processors. It then sends all partitions and the dependency to waiting processors. Once completed, it serves as a slave itself instead of remaining idle. The advantage of this technique is that the master process is better optimized. However, the disadvantage is that there is no consideration for data dependency and data locality amongst the processors.

Figure 30 illustrates a 2D scheduling process over 5 slaves' processors. Processors are illustrated with a P_i label where i is the processor number. The 2D matrix is coloured based on the partitions of size 3, and overlapping cells show a grading between the colours of the

neighbouring partitions. The illustrated scheduling figure is based on a round-robin scheduling technique, where processors are queued for assignments of the next available non-assigned partition (because of the overlapping of neighbourhood, a partition that is neighbour to a previous partition can be already assigned, while still coming up in the neighbouring partitions of a later partition). In a round robin method, the scheduler assigns the processors iteratively. The scheduler keeps iterating over the processors till no more partitions remain. For the bag-of-tasks scheduling method, the scheduler assigns the processors on a first in, first out queue bases. As partitions come up in the partitioning buffers, it is assigned to the next available processor. Each processor scores its partition, and comes back to the scheduler for further assignments.

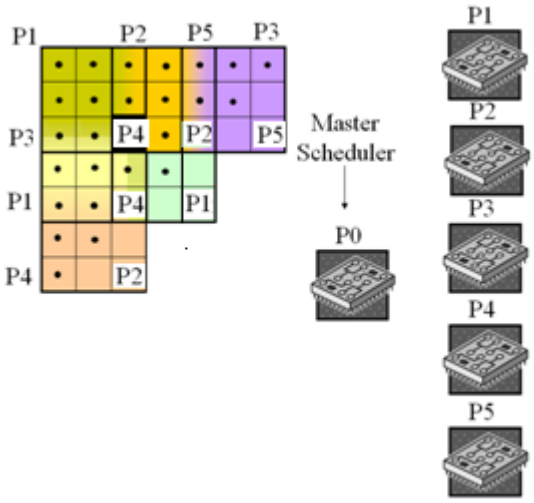


Figure 30: Scheduling in the Master/Slave Approach

The dependency-based scheduling technique optimizes the assignments to processors with the aim of increasing data locality in a single processor; thus reducing communication time. Therefore, idleness is optimized since the processor waiting time to receive the required resources is minimized. The advantage of this technique is a low communication overhead, and high data locality. On the other hand, the disadvantage is in the pre-processing overhead needed to calculate the best assignment based on dependency using the neighbourhood function described in section 3.2. This method was further optimized to eliminate the need for a master process as explained in the next Chapter, and was not used for further tests on the master/slave approach.

3.7 Complexity Analysis

The complexity of the proposed solution is analysed in two levels: the computation and the communication. Both are dependent on the partitioning size *S*, and the load balancing. As mentioned above, the computation complexity of the scoring of the l^{th} cell (ξ_l), where $0 \leq l < S_k$, in a partition (p_{ij}), in a computational wave (w_i), in a processor m, is as follows:

- Process the current cell based on its position:
  - If initial cell, initialize it ≡ $O(1)$
  - If internal cell, compute the score as follows:
    - * get pair-wise scores for each pair of residues ≡ $O(k^2)$
    - * get lower neighbours' scores ≡ $O(2^k - 1)$
    - * compute temporary Neighbour score ≡ $O(3 \times 2^k - 1)$
    - * assign maximum temporary score to current cell score ≡ $O(1)$
  - Otherwise:
    - * If local cell (in another partition), check the (OCout) buffer for the previous wave ≡ $O(p_{(i-1),m} \times (S^k - (S-1)^k)) \equiv O(S^k)$
    - * If not local: check received list (OCin) for the previous wave ≡ $O(p_{(i-1),m} \times (S^k - (S-1)^k)) \equiv O(S^k)$
    - * Otherwise, block wait till received ≡ $O(1)$
- Dependency Analysis ≡ $O(p_{(i+1),m} \times (S^k - (S-1)^k)) \equiv O(S^k)$

The worst case complexity occurs when a cell is scored up to the block wait step. This is calculated as: $O(k)+O(S^k)+O(S^k)+O(1)+O(S^k) \approx O(S^k)$. The blocking time is dependent on the architecture.

After a cell is computed, a dependency analysis is conducted by retrieving the cell's higher indexed neighbours. Then each neighbour's partition index is retrieved; thus, the computing processor is specified. If the neighbour's processor is different from the current processor, then the cell scores are added to the *OCout* buffer. The current processor is the one that currently computes the cell.

The complexity of the sequential processing of the design remains $O(n^k 2^k)$, where n is the average length of sequences to the power of the dimensionality k to get the number of the tensor cells. The number of cells is multiplied by the number of operations required to process the score of each cell, which is the number of lower indexed neighbour cells 2^K-1. The exact complexity would be $O(\tau \xi 2^k)$ where τ is the number of elements for the tensor ξ. The division over the available processors V will divide the complexity and hence the total completion time (distributed time) over processors, subject to the partition size chosen

effects over communication cost. The maximum distributed time is of the order of the total computation and communication time as shown in Equation 16.

Equation 16: Asymptotic Complexity in the Master/Slave Approach

$$O(((\sum_{w=0}^{t}\left\lceil \frac{P_w}{V} \right\rceil)\times S^k \times 2^k) + ((S^k - (S-1)^k)\times V^2)),$$

Equation 16 is composed of two components, the computation plus the communication. The computation cost is the sum of the maximum assignment of partitions over processors in each wave, multiplied by the number of cells in each partition, multiplied by the cost of scoring each cell. The communication cost is of the order of the overlapping cells between partitions (the highest edge indexed ones) multiplied by the square of the number of processors as worst case scenario when all partitions dependencies are remote.

3.8 Simulation Results

Testing started with handwritten small data sets, using different machines. Initial experiments were carried out on a single processor machine (Intel Pentium M Processor 740 – 1.73 Ghz, 1 GB RAM, 70 GB HDD), with simulated distributed processes using the *mpich* library version 2-1.0.4-rc1. Then, scalability testing was carried out on an SGI Altix 3700 Bx2 cluster of APAC with configuration described in Chapter 2, section 2.2.2.

Initial results are summarized in Table 6. M1 is the single machine, and M2 is the SGI Altix, described above. The abbreviations used are: "S" for the partition Size, "P" for the number of processes created, "K" for the number of sequences aligned, and "L" for their corresponding lengths, "C" for the total computation performed, "M1 CPU" and "M2 CPU" are the CPU time in the first and second machines respectively measured in seconds, then "P Mem"& "V Mem" are the Physical and Virtual Memory consumption. The testing was continued to identify the optimization chances based on varying the partitioning size, the number of processors, and the best scheduling techniques. A partition size of 3 was used in all tests, except the last one which uses a partition size of 20. Figure 31 plots the various performance measures, where CPU time of both machines is measured in seconds. Physical and Virtual Memory are measured in MB. The Total computation is calculated with all tensors cells of k dimensionality and average sequence length *n*, multiplied by computations

per cells, which is total neighbours of 2^k-1. The exponential consumption of resources (memory and computation time) is shown in Figure 31. In Section 4.7.1 further comparisons with sequential processing show that thy79e parallel processing using the master/slave partitioning approach presented in this chapter, consume less resources only for small data size. This is due to the fact that sequential processing doesn't include the partitioning and communication cost. However, as the data size increased, the parallel processing started to consume fewer resources than the sequential processing.

Table 6: Mater/Slave Simulation Experiments Results:

S	P	K	L	C	M1 CPU	M2 CPU	P Mem	V Mem
3	3	3	4, 3, 2	192	0.20	0.00	15	122
3	3	4	5, 4, 3, 2	1800	0.50	0.00	15	122
3	3	3	7, 8, 9	4032	257.00	3.00	48	281
3	4	5	6, 5,4,3,2	17280	60.25	8.00	62	355
3	4	6	7, 6, 5,4,3,2	176400	79.60	10.00	76	429
20	3	3	90,80,85	4896000	2370.57	7783.00	371	606

3.9 Conclusion

This Chapter contributed a master/slave approach to the partitioning/scheduling of MSA scoring tensor using the dynamic programming algorithm. The approach used MoA and Ψ-Calculus methods in indexing the tensor. The MoA methods allowed the following:

1. MSA tensor scoring recurrence that is invariant of dimension and shape, and based on index transformations;
2. Neighbouring functions that is invariant of dimension and shape that is used for scoring and partitioning.

The dependency between partitions was analysed and the scheduling over processors has been studied. The initial simulations results have been presented. It is noted that the performance can benefit from a dependency-based scheduling that keep partitions local to their other dependent partitions. The partitioning methods presented in this Chapter created dependency between partitions within the same wave of computation and across waves of computations. This is due to the cubical partitioning that divided the computations into waves, where each wave of computation included partitions in the same multidimensional

perimeter from the origin, but still dependent on each other. This was illustrated in Figure 24 and Figure 25, where the same wave was shown to include partitions that are two or more steps away on the same row or column in 2D, or the same axes on 3D and similarly on higher dimensions. Further work on studying the partitions neighbourhood and new partitioning/scheduling methods will be presented in the next Chapter.

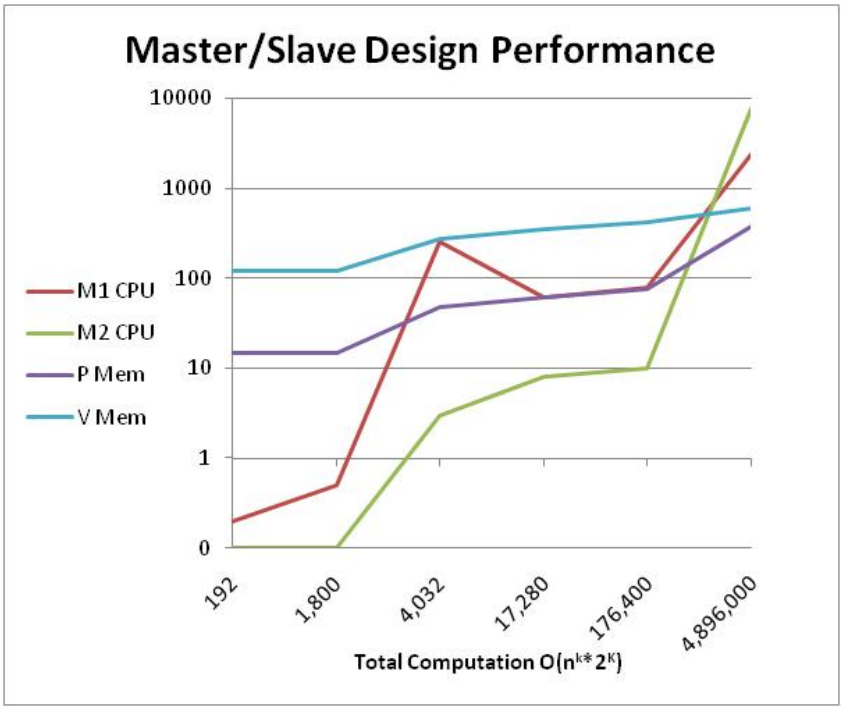


Figure 31: Performance Measures of the Master/Slave Approach.

Chapter 4: Peer-to-Peer Optimized Approach

As explained in the conclusion of Chapter 3, the master/slave approach required a refinement to increase both data locality and parallelism. The master/slave approach presented a parallel approach to MSA, where dependency was found to be within the same wave of computation and across waves of computations. To further increase parallelism, dependency within the waves needs to be eliminated. Two conclusions were made; 1) partitions needed to be assigned to processors based on their neighbourhood to reduce communication costs of nearby partitions; 2) it is better to redefine the waves so that this dependency is considered from the beginning, while assigning partitions to waves, so that the assignment to processors becomes easier.

The study of the scheduling approaches led to dependency based scheduling that assigns partitions to processors that are the closest to each other both within and across the waves. This was achieved by employing the neighbourhood function in the scheduling. This added extra computation cost that made the solution infeasible even for small datasets. Hence, a new wave definition is needed to complete the dependency scheduling approach without further computation of neighbourhood while assigning to processors. The approach discussed in this chapter eliminates the dependency within the same wave. This is by redefining the computation wave to be the group of partitions that are only one step away from each other partition in the previous wave on all axes. This approach allows more processors to work simultaneously without being blocked for dependency of partitions that are being computed in the same wave but assigned to another processor. The new wave definition results in a function where a processor ID can be retrieved from the partition index and vice versa. This eliminates the need for a scheduler master process, as each processor can fetch its own assigned partitions, calculate the partition index of the dependent partitions, and hence calculate the dependant processor ID of the dependant partition and send to the waiting processors.

In this Chapter, we will present how the dependency within the waves has been eliminated by using the wave front technique described in (YAP, TK, 1995) and (YAP, TK, Munson, PJ, Frieder, O, Martino, RL, 1995), to work invariantly of dimension and shape. This Chapter also details an innovative parallel computation wave definition, scheduling and communication optimizations. In more detail, this Chapter will explain the following:

1. Section 4.1 will present an innovative parallel computation wave definition that is different from the one presented in Chapter 3. A wave that is based on distances from the origin cell, and is calculated by generating Integer partitions, or recursively using the neighbourhood functions presented in Chapter 3. This will assign all partitions in one wave to be only one step away on all axes from all partitions in the previous wave. This new definition will be formalized in section 4.2
2. The elimination of the dependency within the partitions in the same wave of computation, ending up with:
    a. fewer partitions per wave,
    b. more waves,
    c. and partitions per wave increases till the middle wave and then decreases.
3. Section 4.3 will introduce an innovative tensor graph representation, and relate it to all existing tree and graph structures in the literature that has been used to represent higher dimensions data.
4. Section 4.4 explains the basic differences between the simplex topology in the literature (the higher dimensional computational technique) vs. the presented Tensor structure using the Ψ-Indexing Techniques.
5. An innovative Tensor abstract diagonalization approach through the new computation wave definition.

Section 4.5 describes the implementation details of the MSA scoring function, scheduling and trace back techniques used in the new approach. Section 4.6 analyses the new design complexity. Section 4.7 presents the simulation results in comparison with the master/slave approach, processor scalability measure and load balancing techniques.

4.1 New Wave Definition

The challenging point lies in defining a MoA function that defines an abstract multidimensional hyper-plane, rather than a multidimensional perimeter. Partitioning is achieved using the MoA functions in Equation 5 and Equation 6 in section 3.1. This section introduces a new definition for the wave of computation that encapsulates the partitions that can be scored simultaneously without dependency within the wave. These are the partitions that fall on the hyper-plane where all lower indexed partitions have already been scored in

the previous wave, like the 2D wave-front parallelism. The new wave definition properties are as follows:

1. In the first wave, the origin cell (index vector of zeros in all dimensions) is retrieved as the partition index, with sum of all indices equal to zero (the wave number).
2. The sum of the index vector elements of the first cell of any partition (partition index) always equals the wave number. This forms a set of independent partitions that are retrieved by increasing a single element in the index vector from one wave to the next.
3. The same procedure repeats till all elements in the shape vector (sequences lengths) are incremented up to the corresponding sequence length minus the partition size; thus, retrieving a single partition in the last wave.

Several algorithms can be implemented to perform step 2 in this process, whether recursively or iteratively until all the scoring tensor is covered. The basic difference between the cubical partitioning and its partitions numbers of Chapter 3 as illustrated in Figure 24, and the diagonal partitioning in this Chapter is illustrated in Figure 32. The new wave definition (described in this thesis as peer-to-peer partitioning due to the elimination of the master process), groups partitions on the exact same distance from the origin, and only one step away on all axes from the partitions in the previous wave. This will be seen later, in Figure 41, where the sums of the index elements of the partition index of all partitions on one wave all equal the wave number. Also, the difference between any partition in one wave and all other partitions in the previous wave is described as one of the following:

1. Only one index position is incremented while the rest remained the same
2. One index element is decremented and another element is incremented twice.
3. One index element is decremented and two other index elements are incremented once.

These three cases guarantee that the sum difference is always one from one wave to another. This creates a gray code ordering among the multidimensional indices of the partitions.

Figure 33 illustrates the dependency in the presented method in 2D: it shows that partition 1 sends interdependent data only to partitions 2 and 3. Partitions 2 and 3 take care of the interdependent data required for partition 5, including what they receive from partition 1. The $S^k-(S-1)^k$ overlapping cells, where S is the partition size, are previously sent to 2^k-1

waiting partitions, whether local (buffered only and not sent between partitions), or remote incurring a communication cost. The new peer-to-peer design reduces the communication overhead of the overlapping cells in the early waves and non-edge partitions to only *k* waiting partitions. The method is extended to 3D cube with shape vector <3, 3, 3> as shown in Figure 34 where the dependency connections are coloured per dimension. Figure 35 shows another 3D dependency network for a cube with shape vector <4, 4, 4>, showing the clustering over processors as different gray shades. All partitions with the same shade are assigned to same processor. Furthermore, Figure 36 illustrates the implementation of the new design on any arbitrary dimension. In Figure 36 each wave has partitions up to the maximum P(w, k) as defined later in Equation 18.

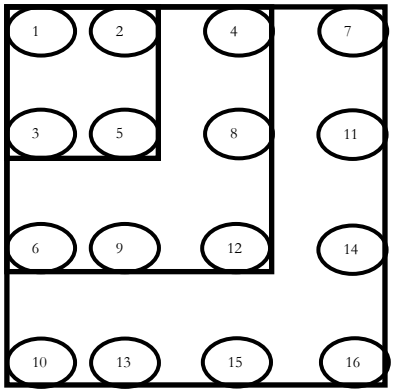


Figure 32: Numbering the Master/Slave Cubical Partitioning in Diagonal Order.

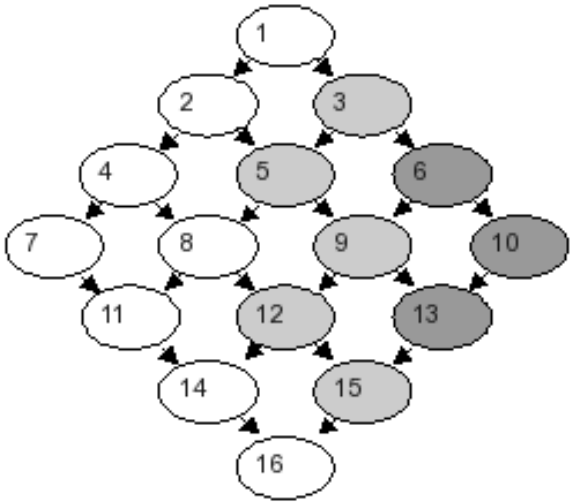


Figure 33: 2D Dependency Networks

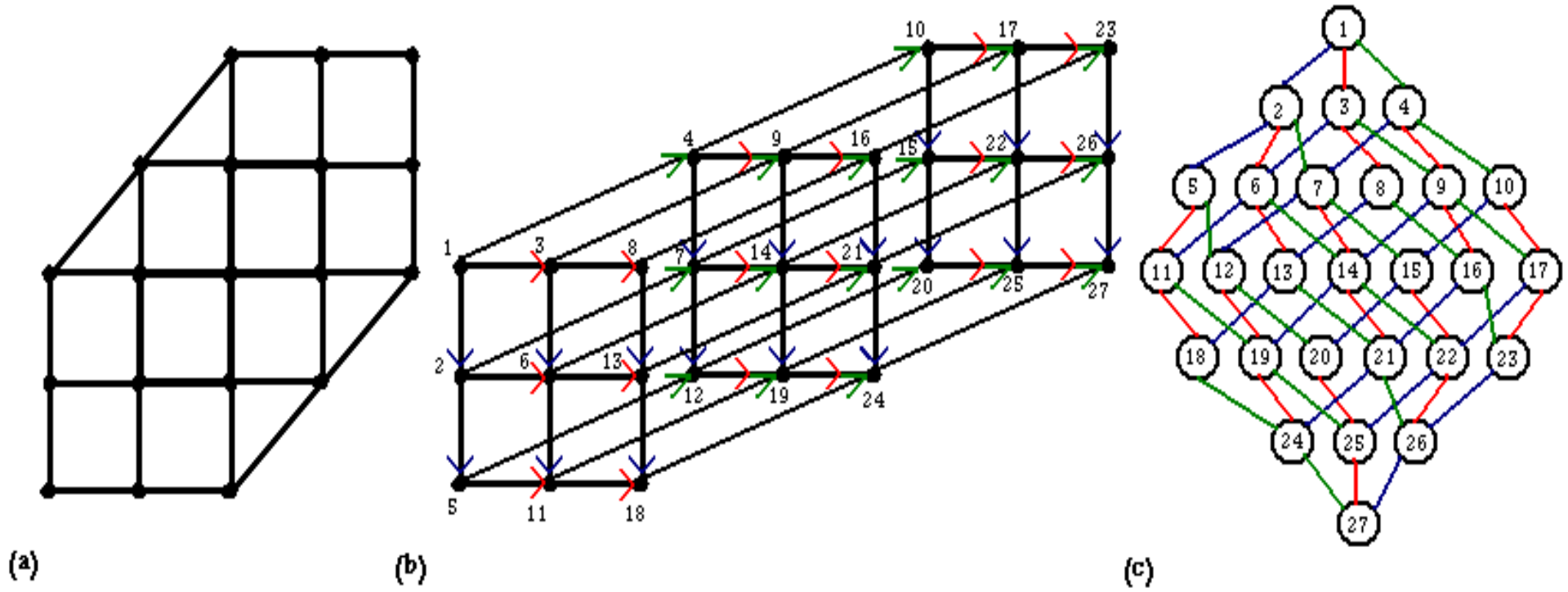


Figure 34: (a) 3D Cube with Shape <3, 3, 3>, (b) Flattened, and (c) Tensor Graph Showing the Dependency Network

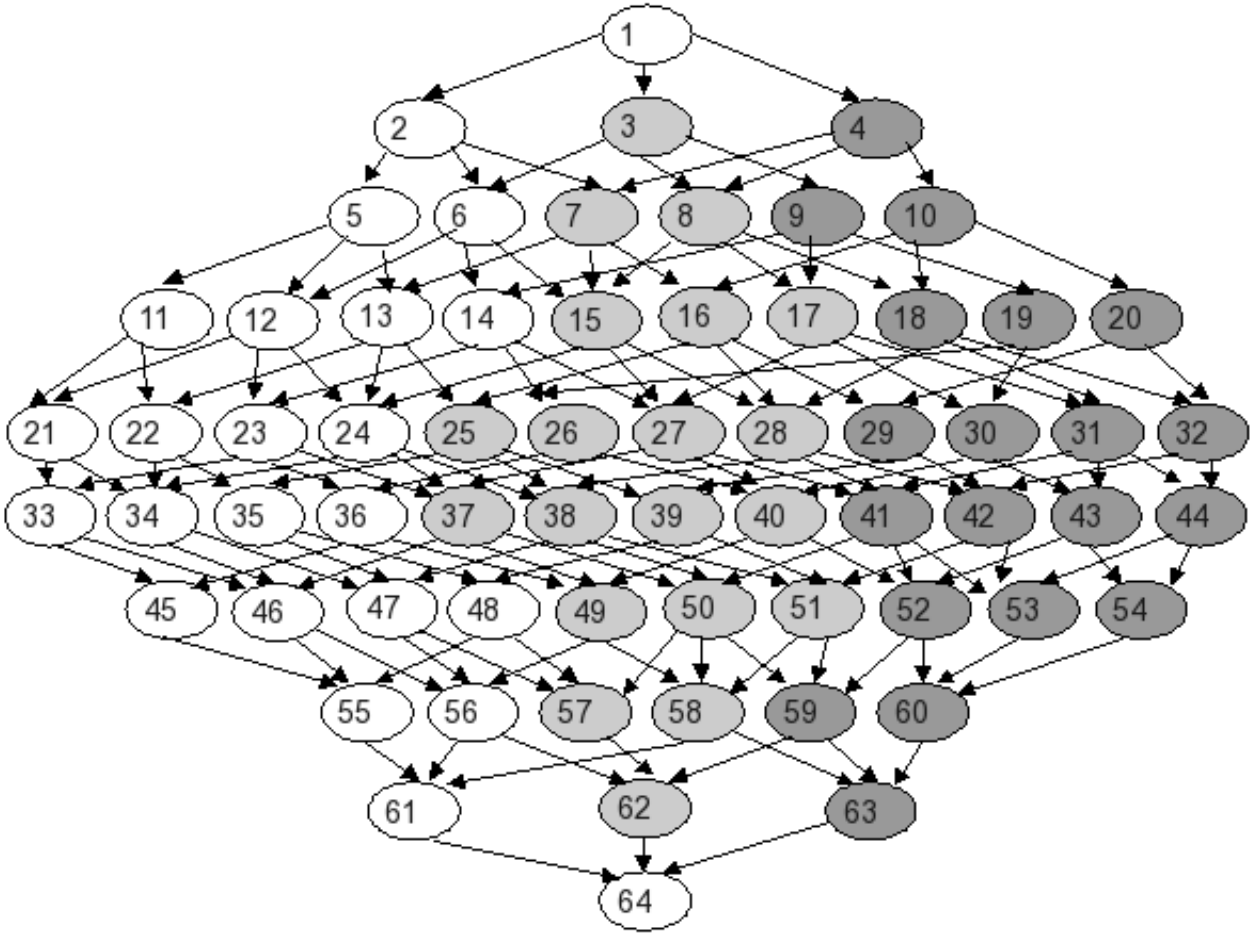


Figure 35: 3D Dependency Network

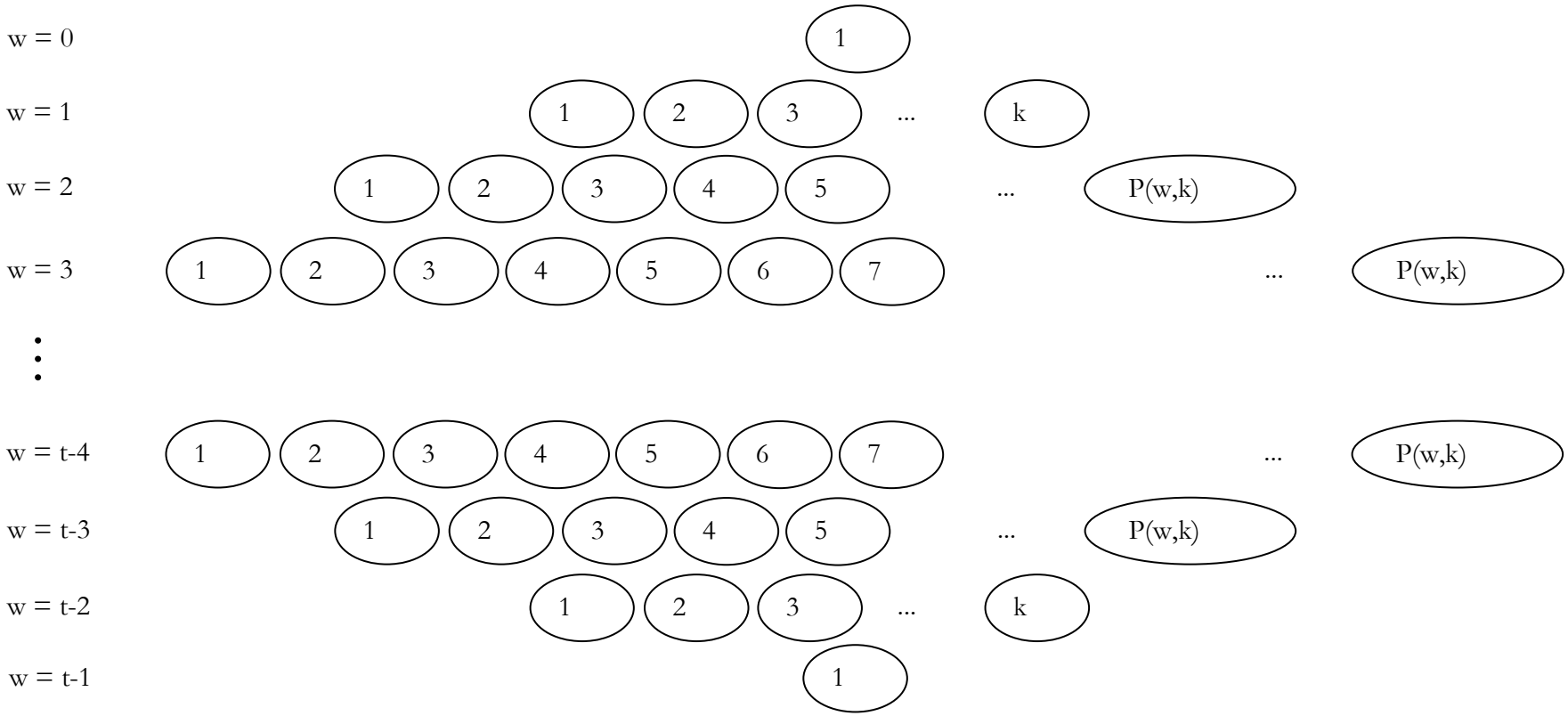


Figure 36: k-D Dependency Networks

Having each cell sending its score to a maximum of k adjacent processors, and only to the next immediate wave, the communication overhead in each partition is reduced. The dependency will be needed in up to k waves; however, will pass from each wave to the next only; then later cells will be responsible for further propagation.

The Peer-to-Peer term used to describe this approach refers to the partitioning/scheduling methods as compared to the master/slave approach explained in Chapter 3. In P2P, the partitioning is pre-processed; each processor fetches its own partitions, and calculates the destination processor ID for the dependencies it scores. Therefore, there is no need for a scheduler master process to partition and manage dependency communication. Thus, the master's computation cost and much communication overhead are eliminated. Each processor is responsible for:

1. fetching its own assigned partitions,
2. defining the neighbouring dependent partition indices,
3. defining the processor computing each neighbouring partition from its partition index using an easy mapping function that will be described below,
4. dependency is sent directly from the computing processor to the dependant processor, without being sent first to the master scheduler to define and resend to the dependant processor.

This optimization redefines the computation wave as stated previously, to be all partitions located at the same distance from the origin and can be executed simultaneously by the available processors. This term "wave" indicates the grouping of independent partitions, and the motion to the next wave.

4.2 P2P Partitioning Formalization

This section will provide a mathematical formalization of the new wave definition. For a given data set of k sequences ($\delta\xi$ - dimensionality), shape $\varrho\xi$: (sequences lengths) ϱ_i length ($0 \leq i < k$), and a partitioning size S, the total number of waves t is calculated by Equation 17. The total number of waves is defined to be the maximum distance from the origin as multiples of the specified partition size. The number of waves created in the P2P diagonal partitioning scheme is more than those created by the master/slave cubical partitioning scheme presented in Chapter 3 and described in Equation 9, which divides the longest

sequence length (dimension upper bound – shape vector element), by the partition size S. On the other side, Equation 17 sums all divisions of shape elements (sequence lengths) by the partition size.

Equation 17: P2P Design Total Number of Waves

$$t = \frac{\rho_0 - 1}{S-1} + \sum_{i=1}^{k-1}\left(\frac{\rho_i - 1}{S-1} - 1\right) \quad \text{or} \quad t = \sum_{i=0}^{k-1}\left(\frac{\rho_i - 1}{S-1} - 1\right) + 1 \quad \text{or} \quad t = \sum_{i=0}^{k-1}\left(\frac{\rho_i - 1}{S-1}\right) - k + 1$$

The same number of total partitions over all the waves is produced using this partitioning scheme as the master/slave cubical partitioning scheme and described in Equation 11 in Chapter 3. Equation 18 calculates recursively the number of partitions P_w that are available for independent computations in the available processors for the wave number w and the dimensionality k. This function grows slower than Equation 10 in Chapter 3, where the partitions per wave are calculated for the master/slave approach. This is due to the fact that Equation 10 raises the number of wave to the power of the dimensionality k, while Equation 18 uses recursive summation to the smallest wave and dimensionality. However, the processors are more efficiently used with less idle time, as there is no dependency between these partitions in the same wave. In addition, all dependency cell scores are grouped into one communication step between each pair of processors in between the waves. Such distribution eliminates the communication blocking in the middle of the wave computation. Moreover, it packs all dependencies in a wave in a single communication message, reducing the TCP/IP communication overhead. This will be further explained in section 4.5.

Equation 18: Number of Partitions per Wave in the P2P Approach

$$P_w(w,k) = \sum_{i=0}^{w} \begin{cases} 1 & if \quad w = 0 \\ w & if \quad k = 2 \\ P_w(w-1, k-1) & otherwise \end{cases}$$

where w is the *current* wave where $0 \leq w < t$. In each w, there are P_w partitions, distributed equally amongst the V available processors (the cluster size). Equation 19 shows the scheduling (processor assignments), as a simple rounded division.

Equation 19: Scheduling Equation

$$m_{p_{j,i}} = \left\lceil \frac{p_i}{V} \right\rceil$$

Where $0 \leq m < V$, and $m_{pj,i}$ is the processor that scores the *jth* partition in the *ith* wave.

4.2.1 Integer Partition Iterative Wave Calculation

This section details one algorithm for calculating the partitions' indices that fall in one wave of computation. In the presented P2P approach, waves, as well as their partitions, are pre-processed (HELAL, M, El-Gindy, H, Mullin, LM, Gaeta, B, 2008). Each processor fetches its partitions in the current computation wave w_i, based on the partition order p_{ij} *which* is the j^{th} partition in i^{th} wave, and the number of processors V in the cluster as formalized in Equation 17 to Equation 19. One way to define partition indices that have the same sum of all its elements is to use Integer partition algorithm, and permutations. The following algorithm assigns partitions indices to the (p_{ij}) 2D array.

```
getIPwavesPartitions (t, p)
  w = 0
      do
                do
                        p[w][j] ← getNextIntegerPartition(w, more_IP)
                        AddPartIndex (p[w][j])
                        do
                             p[w][j] ← permute (p[w][j], more_Perm)
                             AddPartIndex (p[w][j])
                        while more_Perm
                while more_IP
                w ← w + 1
while w < t
end getIPwavesPartitions
```

The algorithm loops for all waves *i* until the maximum wave *t* is reached. For each wave, the integer partitions of the wave number *w* are retrieved. In numbers, the integer partitions are represented by the different combinations of integers (REINGOLD, EM, Nievergelt, J, Deo, N, 1997) that add up to the wave number; whereas the wave is designated by the distance from the origin. For example, wave 2 has (1, 1) and (2, 0) for 2D index; wave 3 has (2, 1), and (3, 0) for 2D index, and (1, 1, 1), (2, 1, 0) and (3, 0, 0) for 3D index. Then, the permutations of these indices are iterated. The function *AddPartitionIndex* multiplies the index by the partition size, checks for the shape constraints, adds the index to the p_{ij} then finally increments *j*. The shape constraints are: 1) not to exceed the sequences lengths; 2) cell

positions (edge cells vs. inner cells); Table 7 provides the number of independent partitions that can be processed in parallel, preferably in different processors for maximum speed-up.

Table 8 illustrates the actual partitioning of 4 sequences with lengths (8, 8, 8, 8); this becomes (9, 9, 9, 9) after including the initial gap character. This results in 6561 cells to be computed. Dividing by a partition size $S = 3$ in each dimension, 256 partitions are obtained in 13 waves according to Equation 17 to Equation 19. Figure 37 plots the data in Table 7 for 10 dimensions. The y-axis is a logarithmic scale of the number of partitions that can be scored in parallel. The x-axis represents the waves.

The complexity of the Integer Partition step is $O(w)$ and the permutation step is $O(w^2)$ (REINGOLD, EM, Nievergelt, J, Deo, N, 1997). The iterations through the t waves of the design, makes the total complexity of the waves calculation in $O(\sum_{w=1}^{t} w^3) \approx O\left(\left(\frac{t}{2}\right)^3\right)$. This algorithm generates partitions that are outside the tensor space of the specific input data shape (sequences lengths). It also generates repeated permutations that are included in an initial Integer Partition set. These invalid partitions are excluded in the "AddPartIndex" function that increments j if successful. Because of these drawbacks, better approaches have been investigated and are presented in the next subsection.

Table 7: Partition Parallelism (Total Partitions per Wave) Showed for 9 Dimensionality in Rows(k), and 9 Waves in Columns.

k	**Waves**								
	1	2	3	4	5	6	7	8	9
2	1	2	3	4	5	6	7	8	9
3	1	3	6	10	15	21	28	36	45
4	1	4	10	20	35	56	84	120	165
5	1	5	15	35	70	126	210	330	495
6	1	6	21	56	126	252	462	792	1287
7	1	7	28	84	210	462	924	1716	3003
8	1	8	36	120	330	792	1716	3432	6435
9	1	9	45	165	495	1287	3003	6435	12870

Table 8: 4D of Shape (9, 9, 9, 9) Partitions (pw) per Wave (w).

w	1	2	3	4	5	6	7	8	9	10	11	12	13
pw	1	4	10	20	31	40	44	40	31	20	10	4	1

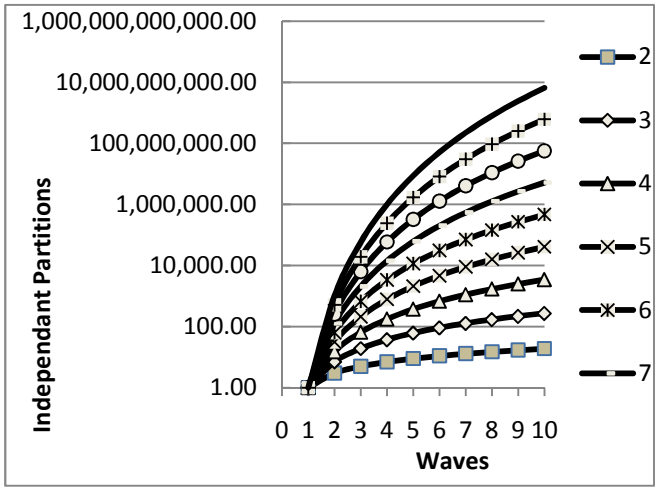


Figure 37: Partitions Parallelism per Wave for 10 Dimensions.

4.2.2 Recursive Retrieval of Neighbour Partitions for Wave Calculation

The integer partition wave calculation method is found to be very slow as the dimension increases and the shape increases specifically with a small partition size. This is due to the large amount of integer partitions and the permutations of each. A hybrid method between the master/slave partitioning method and the P2P one is followed to reduce the complexity. This is performed by recursively traversing through the tensor tree starting from the origin, and at each node, calling the *getHigherPartitions* function described. Then, iterate through all valid partitions and recursively return their higher partitions. This is explained in the following pseudo-code:

```
getwavesPartitions (p)
                getHigherPartitions (p[w][j], parts)
                for all parts do
                        AddPartIndex (p[w][j])
                        getwavesPartitions (p[w][j])

                end for
end getwavesPartitions
```

For every visited partition, the returned higher neighbouring partitions are not all in a single wave, since the MoA neighbouring function -described in Chapter 3- returns all neighbours

across up to *k* waves. However, each partition's wave number is retrieved by the *getwaveNumber* function (as shown below) and stored in the partitioning structure in the correct wave number position.

```
getwaveNumber (in: k, S, partIndex; out: w)
        w ← -1
        For i from 0 to k – 1 do
            Distance[i] ← partIndex[k]/(S-1)
        End For
        w←∑ Distance
End getwaveNumber
```

In the reduced search space run (as will be explained in Chapter 5), only partitions within the specified epsilon distance from the diagonal are included in the partitioning structure, and therefore, only these are recursively called for their higher-order neighbouring partitions. This reduces the partitioning complexity significantly. This recursive function will terminate when no higher partitions are found. There are also repeated partitions indices generated in this function, as the same partition index vertex has adjacency with at most k parents as shown in previous section. However, it is more efficient than the Integer Partition method as it is $O(kP)$ where k is the dimensionality (number of sequences) and P is the total number of partitions generated as shown in Equation 11 which is exponentially growing in k but in rate corresponding to the partition size S chosen.

4.3 Tensor Graph Representation

Figure 33, Figure 34, Figure 35 and Figure 36, the P2P approach defines waves of computations within the tensor as an exponentially growing then decaying graph structure, with a depth equal to the number of waves in the case under consideration as calculated in Equation 17, and nodes are the partitions calculated, and edges are the dependency between partitions. This graph is un-balanced in terms of the number of nodes in each level (wave), and that nodes can have more than one and up to k parents: it starts with 1 node in level 0, then k children in level 1, and keeps on increasing to the middle level. Then the graph decays to k nodes in the level before the last, and again one node on the last level. The intermediate nodes have children from 1 to k based on the corresponding index position and the tensor shape. The children nodes are retrieved by the *"getHigherPartitions"* function, as discussed in section 3.2. The inverse function *"getLowerPartitions"* is used to retrieve the parents of a node in the trace back steps. The maximum number of nodes in each level is calculated as shown in Equation 18; it depends on the shape tensor as discussed in partitions per wave calculation

in section 4.2. Figure 38 illustrates the tensor graphs on binary shapes (upper bounds in each dimension equals $2 \approx 2$ bp sequences) starting from two dimensions in (a) to seven dimensions in (f).

This novel tensor graph is not a simple multidimensional scaling for the binary search tree (BERG, MD, 2008) and its multidimensional scaling to the KD-tree, which is a binary tree in which every node is a k-dimensional point, and every non-leaf node generates a splitting hyper-plane dividing the space into two subspaces. The binary search tree defines a tree with the root as the middle value of some sorted one dimensional array values, where every non-terminal node defines the splitting value of the sub tree it is on top of. The terminal leaves defined the array elements values. The 2-dimensional kd-tree defines some specific points on the 2D plane with the root as the vertical line that splits the points on the plane into two equal subsets. The non-terminal nodes on depth two stores the horizontal lines the further splits the points into equal subsets. All non-terminal nodes stores the splitting lines that are horizontal lines if at even depth from the origin or vertical lines if at odd depth from the origin. The terminal (leaves) nodes are the points on the plane. This doesn't describe the plane itself as much as it describes the specific set of points given. A Kd-tree for n-points uses $O(n)$ storage and can be constructed in $O(n \log n)$ time. This method creates a tree with nodes having a dimension of the tree (1 for the binary search tree), and 2^k children.

An exponential tree is another multidimensional scaling of the binary search tree, where the dimension is not constant at all levels. In this method, the dimension is equal to the depth of the node, having the dimension of the root node equals one and having two children each of dimension two. The nodes at the third level can hold eight children, having dimension equals three, and so forth. This tree was laid out on a hyperbolic space in (LAMPING, J, Rao, R, 1994), with the root at the centre of the circle. The recursive algorithm presented in this study shows that as the radius increases, the circumference of the circle grows exponentially, keeping the distance between parents, children and siblings uniform and giving more space for more data to be stored for large hierarchies.

A K-partite graph (HARARY, F, 1994) connects vertices such that the vertices can be decomposed into k disjoint sets such that there is no adjacency between the vertices in the same set, while there is adjacency between each pair of vertices in different sets as shown in Figure 39. The K-partite graph structure doesn't describe the presented tensor structure, as

there are less adjacency among vertices and only in between one set (wave) and the next set, not among all sets.

The novel tensor graph representation formulated by the work of this thesis can be described as a multidimensional scaling of the Hasse diagram connecting the residues of the sequences. Figure 40 shows a Hasse diagram for a partial order sets by inclusion of {x, y, z}, which is identical to Figure 38(b) the 3D tensor tree representation. To extend the Hasse diagram into an array of residues in each set, like x = $\{x_1, x_2\}$, and $\{y_1, y_2\}$, and z = $\{z_1, z_2\}$ is shown in Figure 41, where the greyed partition in the middle is considered the middle partition in every wave, and all nodes are ordered from left to right on the index values. An edge from one vertex to another denotes one element index single increment, so that the total sum of all elements adds up to the wave number. This is a Gray code in (REINGOLD, EM, Nievergelt, J, Deo, N, 1997) implemented across the cells' index elements to order them based on one single change in the index elements.

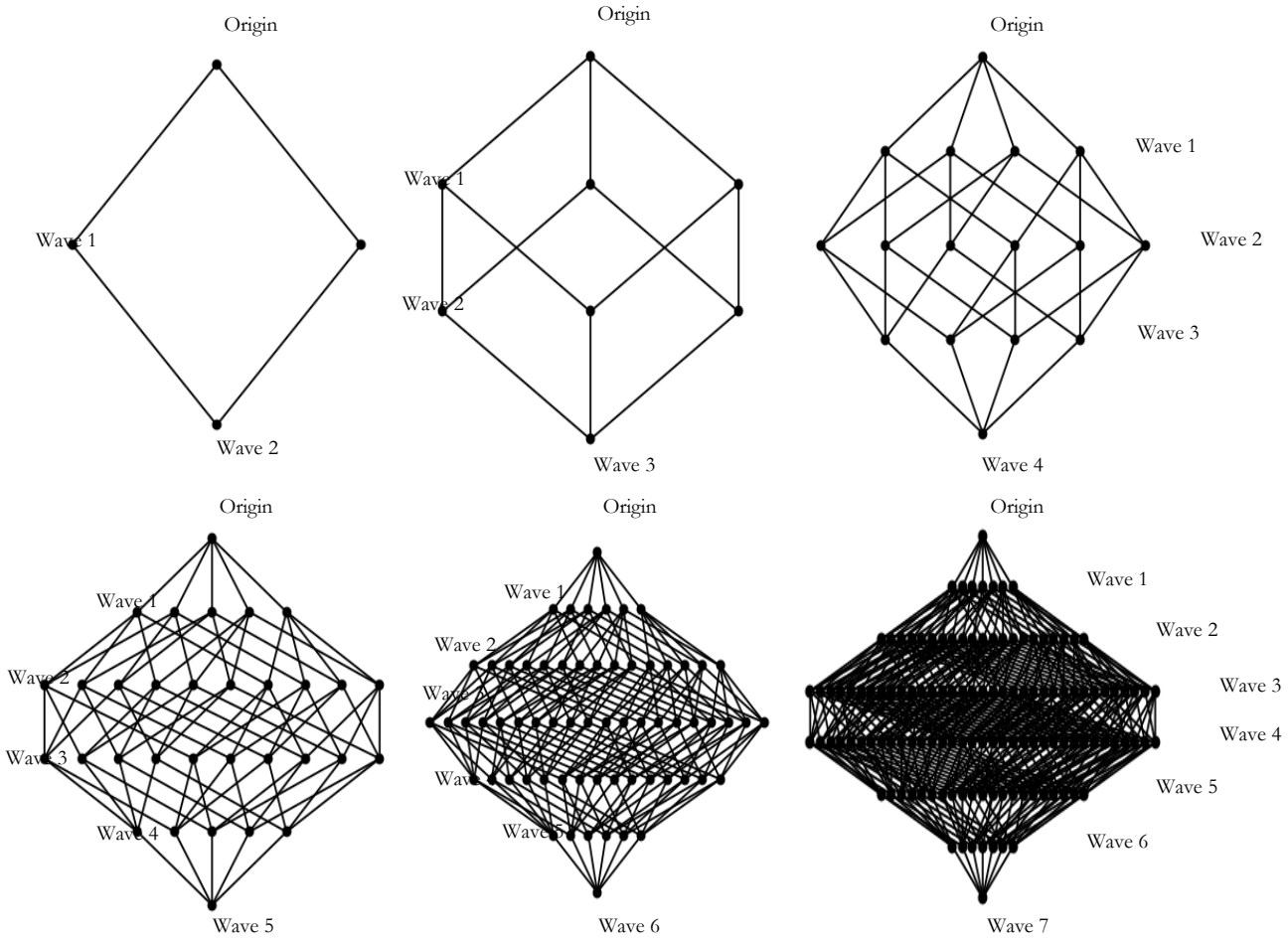


Figure 38: MSA Tensor Graph Representation for Shape Vector of (2, 2 ... 2). (a) 2D Matrix, (b) 3D Cube, (c) 4D Tensor, (d), 5D Tensor, (e), 6D Tensor, (f) 7D Tensor.

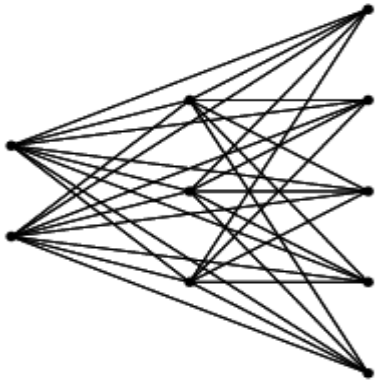


Figure 39: K-Partitie Graph

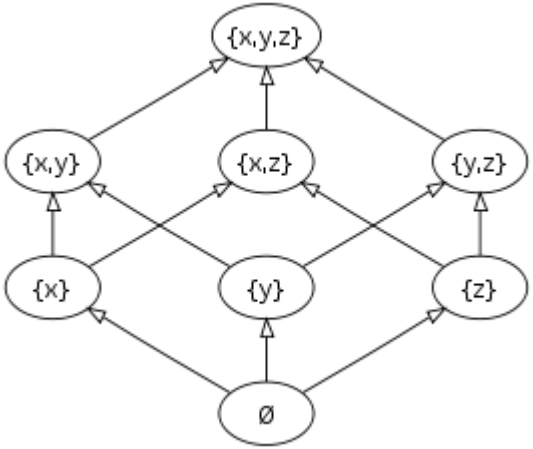


Figure 40: Hasse Diagram

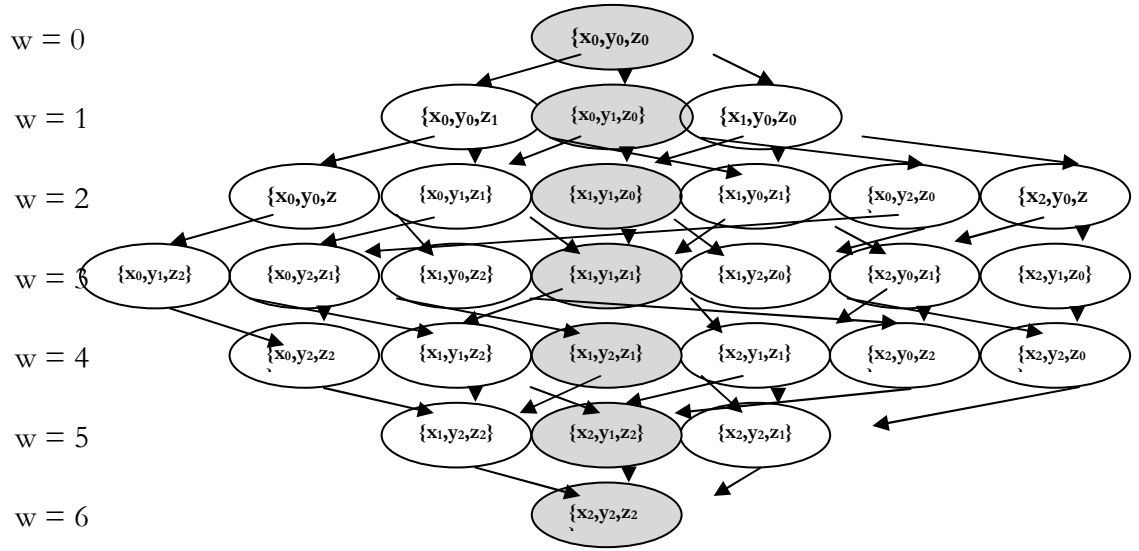


Figure 41: Multidimensional Hasse Diagram

4.4 The tensor points geometric layout vs. the simplex topology

This section will further relate the tensor structure defined in this thesis to the well defined computational data structures for higher dimensional problems, mainly the simplex structure. The geometric layout of the presented tensor graph is illustrated in Figure 42 for 2D, 3D, 4D and 6D. These figures show the points as small circles on the axes (lines) of the dimensionality of the figure and its expansion away from the origin (point of intersection of all axes), forming waves that connect the points on equal distance from the origin. The back planes of the third dimension are projected as shown in Figure 42.b. This figure doesn't show the upwards projections causing the figure to be pyramid-shaped, and not revealing the points inside the pyramid. It only shows the points on the base of the pyramid. Similarly, the fourth dimension back plane is shown in Figure 42.c, which is the 2-D backplane without the remaining two projections forming a cube with internal points uniformly distributed on all intersections of all points on the grid. The same visualization concept is applied to the 6D figure in Figure 42.d, where the 2D back plane only is shown, while the 3D projections are omitted for simplicity.

Table 9 from (RUDIN, W, 1976) lists the geometric simplexes and the corresponding number of vertices, edges, cells and faces. On relating the partitions per wave as shown in the plots in Figure 42 to the simplex definition from geometry as shown in Table 9, we can make some interesting observations. We find that a 2D plot does not have a simplex representation, but its number of partitions per wave corresponds to the Vertices (0-face) column as it grows across the varying simplex dimensionality. The 3D plot is a 2-simplex and its partitions per wave growth across the waves correspond to the edges (1-face) column; a 4D plot is a 3-simplex that relates to the 2-face column.

The growth of the number of faces with the dimensionality of the simplex, is defined to be a triangle read by rows, giving the numbers T(n,m) = binomial(n+1,m+1); or, Khayyam's triangle (Pascal's triangle, Yang Hui's triangle or Tartaglia's triangle) shown in Figure 43 with its left-hand edge removed.

T(n,m) is the number of m-faces of a regular n-simplex. An n-simplex is the n-dimensional analogue of a triangle. Specifically, a simplex is the convex hull of a set of (n + 1) affinely

independent points in some Euclidean space of dimension n or higher, i.e. a set of points such that no m-plane contains more than (m + 1) of them. However, reading the triangle by rows defining the faces of the simplex as it grows with dimension does not define the partitions per wave growth function. The number of partitions per wave for a given dimensionality is growing according to the Khayyam's triangle values diagonally as shaded in Figure 43, rather than by rows for the simplex structure. This defines the basic difference between the presented tensor structure and the simplex structure.

The n-simplex topology defines an n-dimensional triangle for a set of n+1 affinely independent points. The simplex topology relates the n-dimensional space external points onto the 2D space. That's why the Pythagorean Theorem holds true for distance measuring among the points represented by a 2D-simplex, and its 3D scaling measures such as De Gua's theorem (ALVAREZ, SA, 1997). The simplex topology does not provide a complete topology for an indexing system for the tensor (hyper-cube) internal points as shown in the indexing system used in this thesis, and led to the partitioning schemes provided.

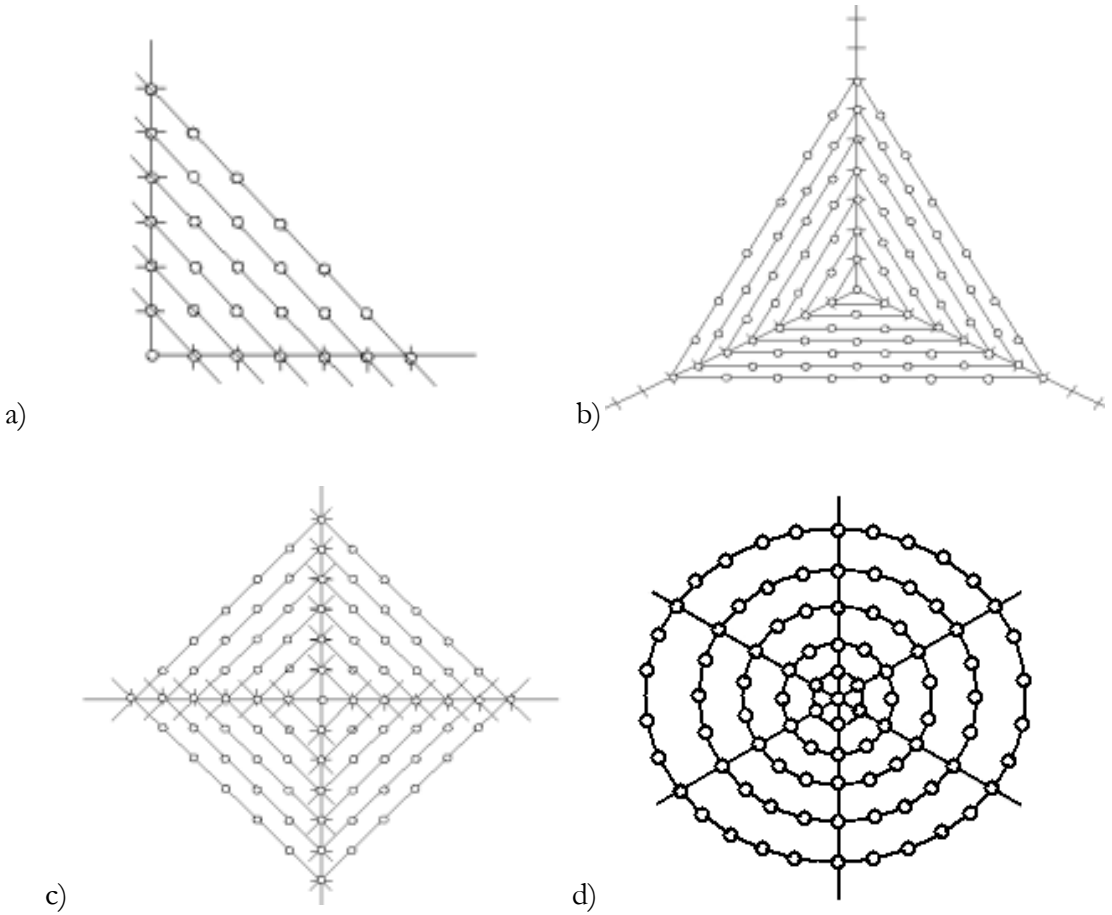


Figure 42: Geometric Layout of the Base of the Internal Points, Higher a) 2D, b) 3D, c) 4D, d) 6D.

4.5 P2P Scoring, Dependency Communication, and Trace Back

This section described the implementation details of the P2P approach. The P2P approach employs the same scoring scheme used in the master/slave approach in Chapter 3. The scoring recursive function was introduced in section 3.2 and the implementation details were explained in section 3.7.

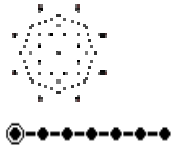
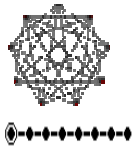
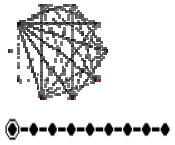
The dependency relationships are computed by retrieving the higher indexed neighbours of the current cell, their partition index and the corresponding processor. If the neighbour partition is found in a different processor, it is added to the *OCout* buffer in order to be sent after all the cells in all partitions in the current computation wave are scored. Then all *OCout* cells are packed to be sent in one buffered MPI communication per processor to reduce the TCP/IP overhead.

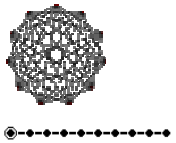
In the presented peer-to-peer design, dependency communication is packed to collect all dependency in all partitions in one wave of computation in one processor to be communicated in one MPI communication transaction. In contrast to the master/slave approach, the packing per wave was not feasible because there was dependency within the same wave. So, each processor after the score computation of each assigned partition would pack this particular partition's dependency by dependant processor id, and an MPI communication cost incurs. Now that the dependency between partitions within the same wave is eliminated, dependency communication can be delayed till all partitions in the same wave are computed, and all dependency from all wave's partitions are packed and sent once.

The implementation of the communication of dependencies is performed using Message Passing Interface (MPI) in two steps. The first step is clock-wise communication where higher ranked processors send to lower ranked ones. The second step is an anti-clockwise where lower ranked processors send to higher ranked ones. Then an MPI barrier is implemented to guarantee that all processors are working on the same computation wave at the same time. All required dependencies for the next wave are communicated in between wave computations. This method avoids deadlocks, since no two processors are involved in a two-way communication at the same time.

Table 9: N-Simplex Elements

Graph	Name	Vertices *0*-faces	Edges *1*-faces	Faces *2*-faces	Cells *3*-faces	*4*-faces	*5*-faces	*6*-faces	*7*-faces	*8*-faces	*9*-faces
	Point *(0-simplex)*	1									
	Line segment *(1-simplex)*	2	1								
	Triangle *(2-simplex)* Tetrahedron	3	3	1							
	*(3-simplex)* Pentachoron	4	6	4	1						
	*(4-simplex)* *Hexateron*	5	10	10	5	1					

	*Hexa-5-tope* (5-simplex) *Heptapeton*	6	15	20	15	6	1				
	*Hepta-6-tope*(6-simplex) *Octaexon*	7	21	35	35	21	7	1			
	*Octa-7-tope*(7-simplex) *Enneazetton*	8	28	56	70	56	28	8	1		
	*Ennea-8-tope* (8-simplex) *Decayotton*	9	36	84	126	126	84	36	9	1	
	*Deca-9-tope* (9-simplex) *Hendeca-10-tope*	10	45	120	210	252	210	120	45	10	1

	(10-simplex)	11	55	165	330	462	462	330	165	55	11

								1												
							1		1											
						1		2		1										
					1		3		3		1									
				1		4		6		4		1								
			1		5		10		10		5		1							
		1		6		15		20		15		6		1						
	1		7		21		35		35		21		7		1					
	1		8		28		56		70		56		28		8		1			
	1		9		36		84		126		126		84		36		9		1	
	1		10		45		120		210		252		210		120		45		10	1
	1		11		55		165		330		462		462		330		165		55	11	1
	1		12		66		220		495		792		924		792		495		220	66	12	1
	1		13		78		286		715		1287		1716		1716		1287		715	286	78	13	1
	1		14		91		364		1001		2002		3003		3432		3003		2002	1001	364	91	14	1
	1		15		105		455		1365		3003		5005		6435		6435		5005	3003	1365	455	105	15	1
	1		16		120		560		1820		4368		8008		11440		12870		11440	8008	4368	1820	560	120	16	1

The P2P model uses the same trace back algorithm used in the master/slave implementation with a slight change. It is adapted to make the master process perform its slave function at the same time while scoring its own partitions. No feasible solution is found to eliminate the role of a master node, since there must be a node responsible for merging the partial paths, and editing the alignments as well as the calculating the full alignment scores.

Sum-of-pairs, Distance Matrix (MOUNT, DW, 2004), Balibase score (THOMPSON, JD, Plewniak, F, Poch, O, 1999) and entropy values (LABORATORY, Los Alamos National web site) are calculated from the resulting alignments. Basic classification of the dataset is done using simple sorting of the scores, a threshold for the needed number of classes, and the distance within one class. The existing clustering techniques are applied on the results. These include, but are not limited to, the following: partitional, agglomerative, & graph-partitioning based; similarity/distance functions: Euclidean distance, cosine, correlation coefficient, extended Jaccard, user-defined; agglomerative merging schemes: single-link, complete-link, UPGMA. Other visualization packages are also used such as Matlab, R, Weblogo amongst others (FIELDING, AH, 2007).

4.6 Computational Complexity Analysis

This section analyses the computational complexity of the presented P2P approach. The complexity analysis of the P2P diagonal partitioning design has reduced the complexity from Equation 16 in Chapter 3 of the master/slave approach complexity to Equation 20.

Equation 20: P2P Design Asymptotic Complexity

$$O(((\sum_{w=0}^{t}\left\lceil \frac{P_w}{V} \right\rceil)\times S^k \times 2^k) + ((S^k - (S-1)^k)\times t \times V))$$

Equation 20 shows the same computation cost elements as Equation 16; however, the number of waves t is more in the P2P design as shown in Equation 17 than in the master/slave approach as shown in Equation 9. Also the maximum assignment of partitions over processors is less in the P2P design as shown in Equation 18 than the master/slave approach as shown in Equation 10. The overall computation cost is similar between the two designs. However, the efficiency was improved in the communication cost and in the time a processor remains idle, due to the elimination of the communication within the same wave

communication and all-to-all worst case scenario communication. The communication cost in the master/slave approach as shown in Equation 16, has reduced the all-to-all communication. This is by changing the communication of the number of overlapping cells multiplied by the square of the number of processors V, to the communication of the edge partitions only, between processors and only in between waves. The simulation results in the next section show a fivefold reduction in resources usage for the small datasets used.

4.7 Simulation Results

This section presents the simulation results. The first subsection compares the P2P simulation results to master/slave approach performance. The second subsection provides processor scalability measures. The last subsection measures the load balancing performance as it varies with execution parameters.

4.7.1 Comparison with Master/Slave Approach

The P2P design is tested on a single processor sequentially, as well as on multiple processors (P2P). The evaluation results in terms of CPU time, elapsed time, physical and virtual memory used are shown in Figure 44, where the x-axis is the exponentially growing data size (product of sequences' lengths – cells to be scored). The P2P implementation achieves up to five times speedup for the datasets attempted as shown in Figure 44, and shows better memory optimization as the data size increases, as compared to the master/slave design. Running the solution sequentially on a single processor -without any partitioning or communication- and in a full search space provides a better performance than the parallel solution for small data sizes. However, when increasing the number and length of sequences being aligned, the P2P version achieves an almost 10-fold reduction in CPU time, and 2-fold reduction in the elapsed time on 4 processors, and 4- to 5-fold decrease in memory usage.

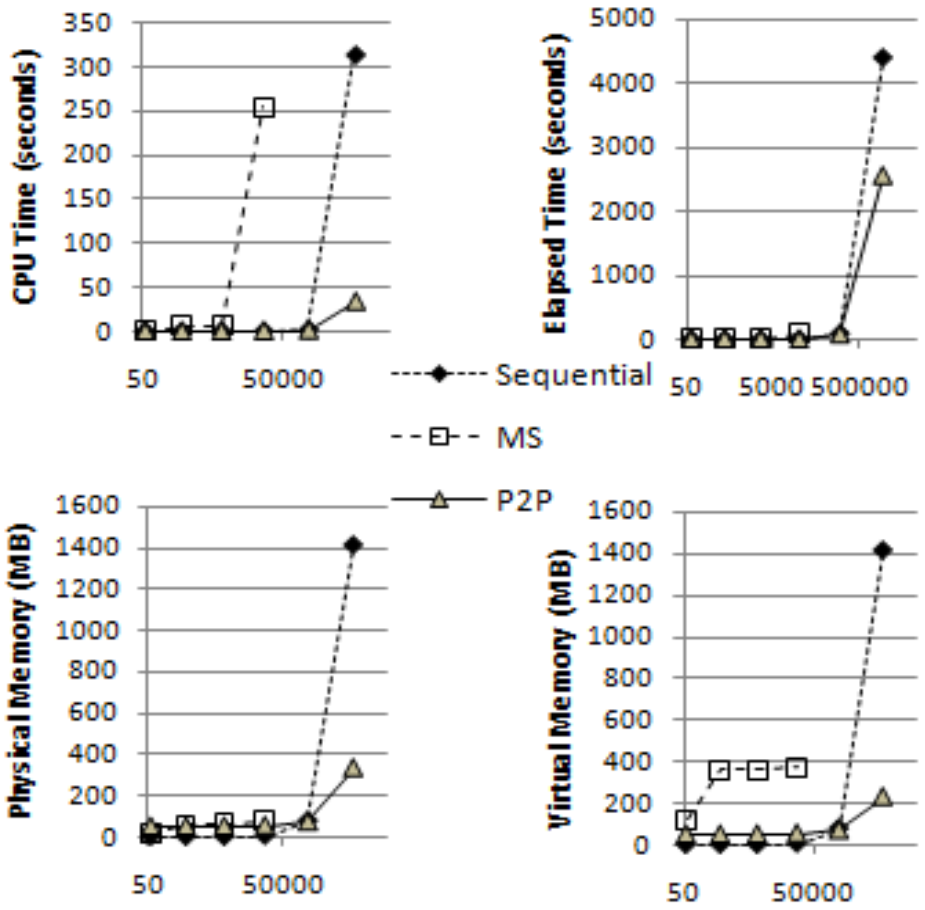


Figure 44: Performance Evaluation of P2P vs. Master/Slave vs. Sequential Implementations

4.7.2 Processor Scalability

In order to measure the processor scalability performance, the same datasets (k = 4 and lengths (11, 16, 17, 18)) using a partitioning size of 5 are used to test the performance on 4, 6, 8, 16, 24, 32 and 64 processors. The resulting elapsed time scalability profile is shown in Figure 45. The elapsed time decreases as the number of processors increases. This type of profile is the desirable one for a distributed application.

Table 10: Processor Scalability Results

CPU	E Time	P Mem	V Mem
0:05:50	0:02:09	81072kb	379296kb
0:07:29	0:01:32	117728kb	537056kb

0:09:07	0:01:23	155632kb	693488kb
0:08:48	0:01:01	320240kb	1321120kb
0:13:04	0:00:58	504912kb	1942752kb
0:17:36	0:00:51	504912kb	1942752kb
0:01:48	0:00:11	1822112kb	5261280kb

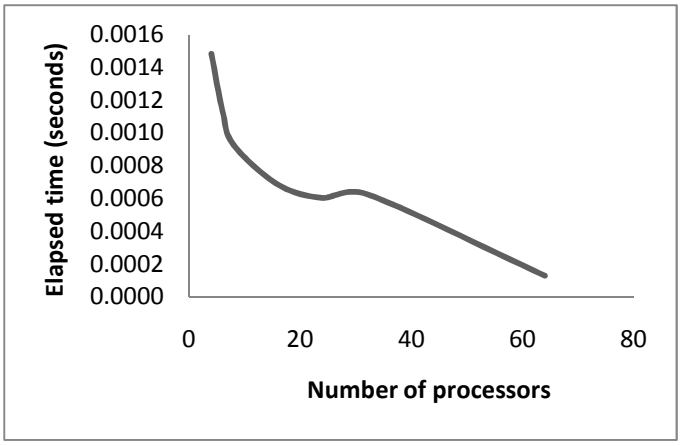


Figure 45: Processor Scalability

4.7.3 Load Balancing

The simple scheduling of partitions over processors is found to have inherent load balancing characteristics. The load imbalance percentage is measured using the following formula:

Load Imbalance= ((largest_load – smallest_load) / largest_load) * 100

By averaging the data collected from the experiments displayed in Table 11 the average Load Imbalance is 15.71%. The higher imbalances can be related to the shape (sequence lengths) imbalance. By sorting the data by the shape standard deviation from the average length, we find that the highest load imbalances (35.53, 37.42, 25.50) are found to be at the highest shape standard deviation of (130.74, 225.57, 549.67) respectively. Another reason that could cause high load imbalance is the waves' number and the total partitions and the partitions per wave ratio. These numbers are based on the shape of the data (sequence lengths), the partition size used *S*, and the search space reduction factor ξ. Figure 46 shows the 8 processors average, max, and minimum loads per processor for the experiments displayed in Table 11, where it is clear that the initial processors are always overloaded with the remaining partitions after dividing the partition per wave over the available processors. This is further magnified as more experiments have been aggregated in this figure. The choice of the data sizes for this experiment is based on randomly created sequences to suit the gradual growth of the data size from one experiment to the next. The partitioning and scheduling over processor varies according to the data size, the partitioning size, the search space reduction factor (ε), and the number of available processors. So, the first two experiments for instance, the data size, the partitioning size, and the number of processors were fixed, and only the ε value changed. The table is sorted by "Load Std Dev" column, to show which factor causes more imbalance for the fair load distribution over processors. So, the experiment with larger data size (4 sequences with lengths {84, 86, 92, 87}), and fixed partition size and number of processors, but smaller ε value, came third in this experiment. Table 11 shows that the average parts/wave is the key to load balance. The highest this average, the better the load balance is, and vice versa.

Table 11: Load Balancing Experimental Results

K	ρ	S	ξ	Shape stdev	Waves	Total Parts	Average Parts/Wave	Load Std Dev	Load Std Dev Normalized	Load Imbalance	Processors Used Partitions Number							
											0	1	2	3	4	5	6	7
4	{31, 31, 31, 31}	3	8	0.00	56	48977	874.59	0.04	0.32	0.80	12.5	12.5	12.5	12.5	12.4	12.4	12.4	12.4
4	{31, 31, 31, 31}	3	7	0.00	56	44967	802.98	0.04	0.32	0.87	12.5	12.5	12.5	12.5	12.4	12.4	12.4	12.4
4	{84, 86, 92, 87}	3	5	3.40	170	127853	752.08	0.06	0.48	1.02	12.5	12.5	12.5	12.4	12.4	12.4	12.4	12.4
4	{31, 31, 31, 31}	3	6	0.00	56	37113	662.73	0.05	0.37	1.05	12.5	12.5	12.5	12.5	12.4	12.4	12.4	12.4
4	{84, 86, 92, 87}	3	4	3.40	170	72662	427.42	0.08	0.62	1.81	12.6	12.4	12.4	12.4	12.4	12.4	12.4	12.4
5	{217, 243, 220, 169, 161}	3	2	35.36	427	150165	351.67	0.14	1.11	2.23	12.6	12.6	12.6	12.6	12.5	12.3	12.3	12.3
3	{2428, 2443, 2521}	17	1	49.93	460	3172	6.90	0.15	1.07	2.61	14.5	14.4	14.3	14.2	14.2	14.1	14.1	
4	{84, 86, 92, 87}	3	3	3.40	170	35477	208.69	0.16	1.24	3.78	12.6	12.5	12.5	12.5	12.5	12.5	12.5	12.1
4	{84, 86, 92, 87}	3	2	3.40	170	13461	79.18	0.62	4.96	9.61	12.9	12.9	12.9	12.9	12.9	11.7	11.7	11.7
3	{31, 31, 31}	3	6	0.00	42	2813	66.98	0.44	3.53	10.51	13.1	12.9	12.6	12.5	12.4	12.2	12.1	11.8
4	{31, 31, 31, 31}	3	2	0.00	56	3945	70.45	0.64	5.10	10.66	13.0	13.0	12.9	12.8	12.8	11.7	11.7	11.6
5	{31, 31, 31, 31, 31}	3	5	0.00	42	2331	55.50	0.74	5.90	13.65	13.5	13.3	13.1	12.4	12.1	11.9	11.8	11.6
5	{217, 243, 220, 169, 161}	3	1	35.36	417	20276	48.62	1.03	8.26	14.69	13.7	13.7	13.7	11.7	11.7	11.7	11.7	11.7
4	{31, 31, 31, 31}	3	4	0.00	42	1773	42.21	0.92	7.33	16.32	13.4	13.3	13.1	12.9	12.8	11.6	11.3	11.2
3	{31, 31, 31}	3	3	0.00	42	1202	28.62	1.42	9.94	21.61	16.5	16.0	13.9	13.6	13.4	13.3	12.9	
3	{1648, 2518, 2665}	17	1	549.67	405	2605	6.43	1.87	13.08	25.50	15.5	15.4	15.4	15.3	15.2	11.5	11.5	
4	{491, 493, 493, 493}	3	1	1.00	979	18498	18.89	2.73	21.82	33.46	15.8	15.7	15.7	10.5	10.5	10.5	10.5	10.5
4	{447, 517, 505, 415}	3	1	48.25	842	15674	18.62	2.74	21.95	33.80	15.8	15.8	15.8	10.5	10.5	10.5	10.4	10.4
4	{391, 402, 438, 457}	3	1	30.78	807	14856	18.41	2.74	21.90	34.26	15.8	15.8	15.7	10.6	10.5	10.5	10.4	10.4
4	{84, 86, 92, 87}	3	1	3.40	170	3092	18.19	2.75	22.04	34.35	15.9	15.7	15.7	10.6	10.5	10.4	10.4	10.4
4	{98, 110, 338, 322}	3	1	130.74	133	3916	29.44	2.76	22.08	35.53	15.8	15.8	15.7	10.9	10.9	10.2	10.2	10.2
4	{105, 121, 457, 541}	3	1	225.57	243	4060	16.71	2.76	22.06	37.42	15.8	15.8	15.7	10.8	10.8	10.5	10.4	9.9

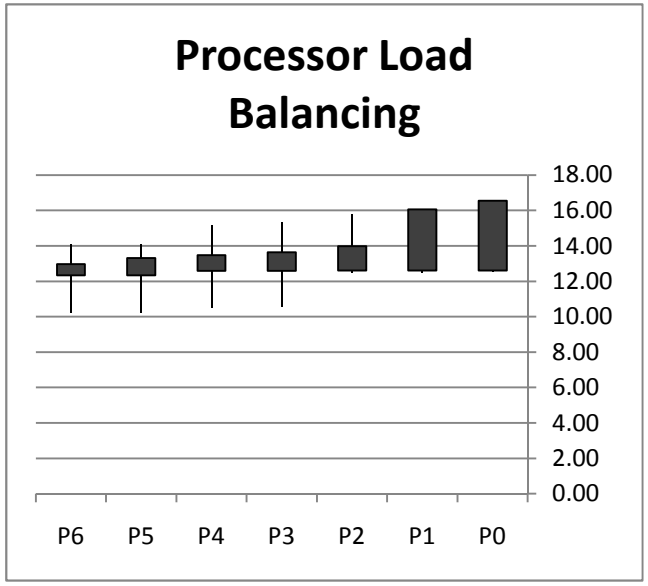


Figure 46: Load Balancing Results

4.7.4 Simulated Comparison with Heuristics Methods

The presented dynamic programming alignment is proven mathematically to be optimal because it searches all possible paths and reports the highest scoring path according the scoring scheme used (GUSFIELD, D, 1997). The innovative scoring recurrence presented in section 3.2 implemented a simultaneous alignment that scales with the dimension and is based on index transformations. The simultaneous alignment approach is proven to be more sensitive than the pair-wise progressive alignment approaches implemented in most available heuristic methods (PERREY, SW, Stoye, J, Moulton, V, Dress, AWM, 1997). Some tests are conducted to compare the alignment performance with the existing heuristic MSA methods. The objective is to compare the alignment optimality in terms of Sum-of-Pairs score. Another objective is to validate the scaling of the scoring scheme used with dimension. Six dissimilar sequences were used as test cases. The sequence alignments using the different methods tested are shown in Table 12. Each sequence alignment is presented in one column, and the alignment of the six sequences is shown under the method name in the first row. The Sum-of-Pairs score of each method is shown in the second row, which is calculated as the sum of each column score (shown as row of alignment here to preserve the space), with +1 for a match, 0 for a mismatch, -1 for a gap and 0 for pairs of gaps.

Using CLUSTAL W (THOMPSON, JD, Higgins, DG, Gibson, TJ, 1994) with default parameters produced a similarity SP score of -36. However, running the same test using the T-Coffee (NOTREDAME, C, Higgins, DG, Heringa, J, 2000) online server produced an alignment similarity score of -22. On the other hand, the MUSCLE (EDGAR, RC, 2004) server on EBI website produced an alignment similarity score of -12. The proposed algorithm results in an alignment with a higher score of -11. As mentioned in Chapter 2, CLUSTAL W, T-Coffee and MUSCLE are progressive methods; they use position-specific gap penalties scoring methods. CLUSTAL W and MUSCLE insert all of the gaps next to each other on one or both ends of the sequences. On the other hand, T-Coffee and the optimal algorithm distribute the gaps more evenly.

Table 12: Sequences Alignments in CLUSTAL W, T-Coffee, MUSCLE, and mmDst

CLUSTAL W							T-Coffee							MUSCLE							mmDst						
						-36							-20							-12							-11
-	C	-	-	-	-	-5	G	G	C	-	-	-	-8	G	C	G	A	A	A	4	G	C	G	A	A	A	4
-	T	-	-	-	-	-5	T	T	T	-	-	-	-6	T	T	T	C	C	T	7	T	T	T	-	-	-	-6
-	G	-	-	-	-	-5	G	G	G	-	-	-	-6	G	G	G	G	T	-	1	G	G	G	-	-	-	-6
-	C	-	-	-	-	-5	C	C	C	-	-	-	-6	C	C	C	T	-	-	-5	C	C	C	C	C	-	5
G	A	G	A	A	A	7	A	A	A	A	A	A	15	A	A	A	-	-	-	-6	A	A	A	-	-	-	-6
T	T	T	C	C	T	7	-	-	-	-	C	C	-7	G	T	-	-	-	-	-8	G	-	-	G	-	-	-7
G	-	G	G	T	-	-5	G	-	-	-	G	-	-7	T	-	-	-	-	-	-5	T	T	-	T	T	T	5
C	-	C	T	-	-	-8	T	-	T	T	T	T	5														
A	-	A	-	-	-	-7																					
G	-	-	-	-	-	-5																					
T	-	-	-	-	-	-5																					

4.8 Conclusion

This Chapter presented another parallel approach for the MSA problem that optimized the master/slave approach presented in Chapter 3. The basic advantages lie in the following:

1. The elimination of the master process work, the scheduler, and employing a peer-to-peer approach in partitioning the assignment to processors.
2. The elimination of the dependency between partition that lie within the same wave of computation, hence optimizing the processor idleness and load balancing.
3. To achieve these implementation optimizations, an innovative tensor graph representation has been implemented, and a description of the basic differences between the simplex structure higher dimensional representations vs. the new presented Tensor approach have been discussed.

The simulation results were shown to out-perform the master/slave approach in all performance aspects (execution time and memory usage). However, the exponential growth as the dataset increases in size requires a further optimization to eliminate the scoring of cells that are known not to participate in the final solution. Further work on search space reduction techniques will be presented in the next Chapter.

Chapter 5: Reduced Search Space Heuristics and Optimization

Chapter 4 presented an optimized peer-to-peer parallel approach to MSA that out-performed the master/slave approach presented in Chapter 3. However, the exponential growth of the tensor space with the addition of extra sequences to the input, limits the usefulness of the design to sequences of moderate length. However, the simulation experiments showed that the method achieves higher scores for simulated sequences than the three established MSA methods CLUSTAL W, T-Coffee, and MUSCLE. Further optimization is required to enable the method to work for higher dimensions (more sequences) and for longer sequence lengths. In this Chapter we investigate an intelligent search space reduction technique that does not decrease the optimality of the final results significantly.

The objective of this Chapter is to define a search space reduction method, where only the area around the expected optimal path for the global MSA is scored (HELAL, M, Mullin, L, Potter, J, Sintchenko, V, 2009). As introduced in Chapter 2, Carrillo and Lipman (LIPMAN, D, Carrillo, H, 1988) noticed that all the exponentially growing number of cells in the dynamic programming scoring hyper-cube is not needed to reach an optimal solution. The literature of the different methods implementing the Carrillo and Lipman search space reduction was discussed. Carrillo and Lipman defined a method of pair-wise projections over the optimal path and to define bounds on these surface projects to trim from the search space.

As opposed to the Carrillo and Lipman methods, the work in this thesis determines the search space by an epsilon value (ε) that measures how much away from the hyper-diagonal we can include in the scoring. The first section in this Chapter develops an approach relying on a novel definition of a hyper-diagonal through the tensor space, around which the global alignment optimal path is expected. The second section considers different distance measures used to identify a band around the hyper-diagonal, and highlights some problems and how they can be addressed. The third section explains some implemented optimization techniques to reduce the complexity. Then the fourth section presents some simulation results. The fifth section concludes this Chapter.

5.1 Hyper-Diagonal Definition

The main objective of this section is to define the area where the optimal global alignment is expected. From the explored literature, it is defined to be the area around the diagonal of the scoring matrix in a pair-wise alignment, or the internal points around the 3D-diagonal in the scoring cube for a 3D alignment and similarly in higher dimensions. A hyper-diagonal in higher dimensional lattice is not strictly defined in literature, and several attempts were made using known methods. We present a definition of the hyper-diagonal which selects a unique diagonal point for each wave number.

The search space reduction technique followed in this thesis determines how far from the hyper-diagonal we need to score, rather than how far from the edges we need to trim, as in the Carrillo and Lipman approach (discussed in Chapter 2).

Only nodes around the middle partition in each wave of computation of the scoring lattice need to be scored. The selected approach is based on the P2P partitioning scheme described in Chapter 4, using the tensor indexing scheme provided by MoA as described in Chapters 2 and 3. The hyper-diagonal is defined as a line that:

1. Connects two points: the origin cell of the hyper lattice and the sink cell as the last point in the lattice that is indexed as the lengths of the corresponding sequences lengths on each axe,
2. Passes through the waves' middle partitions, that is defined in the same way as a 2-D diagonal points; assuming equal lengths at all dimensions, in 2D the middle points in each wave are: (0, 0), (1, 1), (2, 2) ... (l_1, l_2), where l_i is the length of sequence i.
3. In higher dimensions, the diagonal is defined to be a line connecting cells of equal index elements, i.e. (0, 0, 0, ... 0), ..., (1, 1, 1, ..., 1), ... (2, 2, 2, ... 2), ... (l_1, l_2, l_3, ... l_k).

There are variable paths that can be taken to connect these diagonal points across each consecutive wave. The waves are not defined on these points only. Since the sum of each consecutive cell index elements does not equal the sum of the previous cell index elements plus one; therefore there are waves in between these points to comply with the gray code definition of waves as presented in Chapter 4. The intermediate points that define the middle

partitions in the waves in between (0, 0, 0 ... 0) and (1, 1, 1... 1), up to (l_1, l_2, l_3 ... l_k), can be defined in different ways, however, a simple way is selected and described below.

In the distributed waves of computations, as previously discussed in Chapter 4, each wave is defined by all the permutations of the indices for which the sum of elements is equal to the wave number. This refers to all permutations of all integer partitions of the specified wave number, over the dimensionality *k*. Consider that the points connecting the hyper-diagonal line through the tensor constitute the middle partition on each wave. These points are calculated as the fair division of the wave number over the indices; with the highest remainder distributed over the middle indices -starting from the middle dimension index going toward the edge dimensions. The quotient is then ceiled. The middle partition index is calculated in the following procedure:

```
q = floor(w/k)
r = w mod k
for i = 0 to k-1:
if  floor((k-r)/2) <= i < floor((k+r)/2)
                    mp(i) = q + 1,
else
                    mp(i)  = q,
```

where mp(i) is the middle partition index at i^{th} dimension, k is the dimensionality, and w is the wave number. Examples are shown in Table 13.

It is assumed that the sequences are of equal lengths. However, it can be adjusted to consider variable lengths as a function to advance the element values of the diagonal line points as a function of the corresponding sequence length to each element. The hyper diagonal line is visualized as the vertical line connecting the middle partitions in all waves as shown in Figure 38.

Table 13: Middle Partition Index Examples

w	k	mp
5	6	{1, 1, 1, 1, 1, 0}
9	6	{1, 2, 2, 2, 1, 1}
12	6	{2, 2, 2, 2, 2, 2}
35	8	{4, 4, 5, 5, 5, 4, 4, 4}
40	12	{3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3}

5.2 Distance Measures Between Single Wave's Partitions

Now that there is a diagonal line defined, in each wave, the distance between the "middle partition" index (as defined in the previous section as the diagonal points and their connections) to each other partition must be specified. This is necessary in order to decide whether it is within a specified Epsilon value (ε), and therefore, is included in the scoring, and whether its higher neighbour in subsequent waves should be explored. The objective of this section is to find a norm that:

1. Measures the distance between the middle partition and all other partitions in the same wave, which should be:
    a. Metric/norm
    b. Considers the indices symmetrically
    c. Consider the different indices that has equal summation of its elements
2. The norm also needs to be restricted within a specific epsilon value (ε) around the middle partition. The choice of (ε) need to:
    a. Cause gradual controlled increase in the number of partitions included in the search space.
    b. The increase rate needs to be monotonic until it reaches the maximum width of the band that can be scored around the middle partition in each wave; and then should remain fixed in all waves until the final waves; then it decreases in symmetry with the initial increase in the first waves.

Several geometrical methods are discussed, and several problems are highlighted. Finally, we present a reasonable approach for selecting partitions close to the diagonal, but it is still an open area of research to explore.

The main objective is to uniquely discriminate between partitions in the same wave. This is achieved by dividing the partitions into two categories. The first category contains those on the left hand side of the middle partition in the current computation wave. The second category contains those partitions on the right hand side of the middle partition. The partitions are expected to be evenly distributed on a line connecting all partitions in the same wave. Therefore, distance between each partition and the middle partition index is measured.

The traditional 2D and 3D geometry does not scale with dimension; moreover, it does not handle symmetry in the partition indices in an efficient manner. The first method is to calculate the Pythagoras distance between the middle partition index vector and the other partition indices in the same computation wave. Equation 21 measures the Pythagoras distance between two points *x* and y in the tensor of dimensionality *k*.

Equation 21: Pythagoras Distance

$$\sqrt{\sum_{i=0}^{k-1}(y_i - x_i)^2}$$

Other similar distance measures like squared Euclidean distance will only magnify distances between dimensions that are further apart. Table 14 lists some examples. The first two rows show an example of symmetry problems. They measure the distance between the middle partition at wave 6 and two edge vector indices in the opposite directions on the tensor graph. The distance measure is the same. Then the next two rows show the distance between the same middle partition and other non-symmetric vector indices on the same wave, but the sum of their elements are the same. Both still produce the same distance measure because the element differences compensate each other at the final summation. This is because the distance calculation is neither sensitive to the directions in each dimension, nor to element positions.

Another method attempts to measure the angle between the index vectors of the partitions as calculated in Equation 22. However, this method results in non-uniform angle distances (no unit distance between each 2 consecutive partitions, on both sides of the diagonal) as shown in Table 14, and produces the same problems.

Equation 22: Index Vectors Angle Norm

$$\theta = \cos^{-1}\left(\frac{x.y}{\|x\| \quad \|y\|}\right)$$

Where $x.y = \sum_{i=0}^{k-1}(x_i y_i)$ and $\|z\| = \sqrt{\sum_{i=0}^{k-1} z_i^{\,2}}$ for any vector z.

A third method is the Manhattan distance, or city block distance. This method measures a route along non-hypotenuse sides of a triangle, i.e. grid-like layout of points. It is calculated as shown in Equation 23. The resulting distances are also shown in Table 14.

Equation 23: Manhattan Distance

$$\sum_{i=0}^{k-1} |y_i - x_i|$$

Another method attempts to handle all the problems mentioned above by ranking the dimensions of the measured indices. The idea is based on the Gray Codes (GRAY, F, 1947), in which each index change counts. For example, the distance between (0, 0) and (1, 0) is equal to 1, while the distance between (0, 0) and (1, 1) is equal to 2. Furthermore, the sign problem is solved by taking the first different index sign. If the first index element is different in a dimension that is lower than the middle dimension, therefore the sign is positive. If the first different index is at a dimension higher than the middle dimension, then the sign is negative. For example, a 5D tensor has 5 elements in its indices. If the first different element is before the middle one, which is the third element, then the sign is positive. Otherwise, if the first different index is the third, fourth, or fifth, then the sign is negative. The problems of permutations and symmetry are solved by weighting each dimensions differently, using Equation 24.

Table 14: Pythagoras, Angle distances, and Manhattan distance between partition indices on the same wave examples.

First Partition Index	Second Partition Index	Pythagoras distance	Angle distance	Manhattan distance
(1,1,2,1,1)	(6,0,0,0,0)	5.66	1.21	10
(1,1,2,1,1)	(0,0,0,0,6)	5.66	1.21	10
(1,1,2,1,1)	(0,1,2,3,0)	2.45	0.71	4
(1,1,2,1,1)	(0,1,1,3,1)	2.45	0.77	4

Equation 24: Index Ranking Distance Measure

$$\sum_{i=0}^{k-1} \left| y_i - x_i \right| \quad \times \quad i^2$$

Table 15 shows the distances based on Equation 24 for the partition indices examples used before in Table 14.

Table 15: Index ranking Distances Measures Examples.

First Partition Index	Second Partition Index	Index ranking distance
(0, 0)	(1, 0)	1
(0, 0)	(1, 1)	2
(1, 1, 2, 1, 1)	(6, 0, 0, 0, 0)	-34
(1, 1, 2, 1, 1)	(0, 0, 0, 0, 6)	62
(1, 1, 2, 1, 1)	(0, 0, 2, 3, 0)	14
(1, 1, 2, 1, 1)	(0, 1, 1, 3, 0)	-10

The epsilon value (ε) in the index ranking method is a percentage of the distances to include in the search space. However, the unit partition and the full wave length should be calculated. The unit partition is defined as distance between the middle partition and its direct neighbour on the same wave. The full wave length is the distance between the middle partition and the first partition in the wave (the edge partition). The first edge partition is the vector index where the value of the first dimension index is the wave number and the rest of the dimension indices are all zeros. This reduced the number of partitions to be scored as illustrated in Figure 47. The reduced search space area takes the shape of a cone whose slope

increases rapidly close to the origin of the tensor graph space, but decreases more slowly across the waves which are close to the bottom of the tensor graph. However, the distance measure between the partitions is still not uniformly distributed over the wave length. Moreover, no fixed percentage of all partitions in each wave is included in the search space. This is because more edge partitions are scored on the top of the tensor graph. As the shape of the tensor increases, and the number of dimensions increases, the reduced search space area becomes much larger than what is actually required, and doesn't form a fixed band across all waves.

Finally, the method adopted for our experiments, selects a specific constant band around the middle partition in each wave. This requires the Epsilon value (ε) to be a fixed number, designating the maximum absolute distance between each index element value in the partition index to the corresponding index element in the middle partition, as shown in Equation 25.

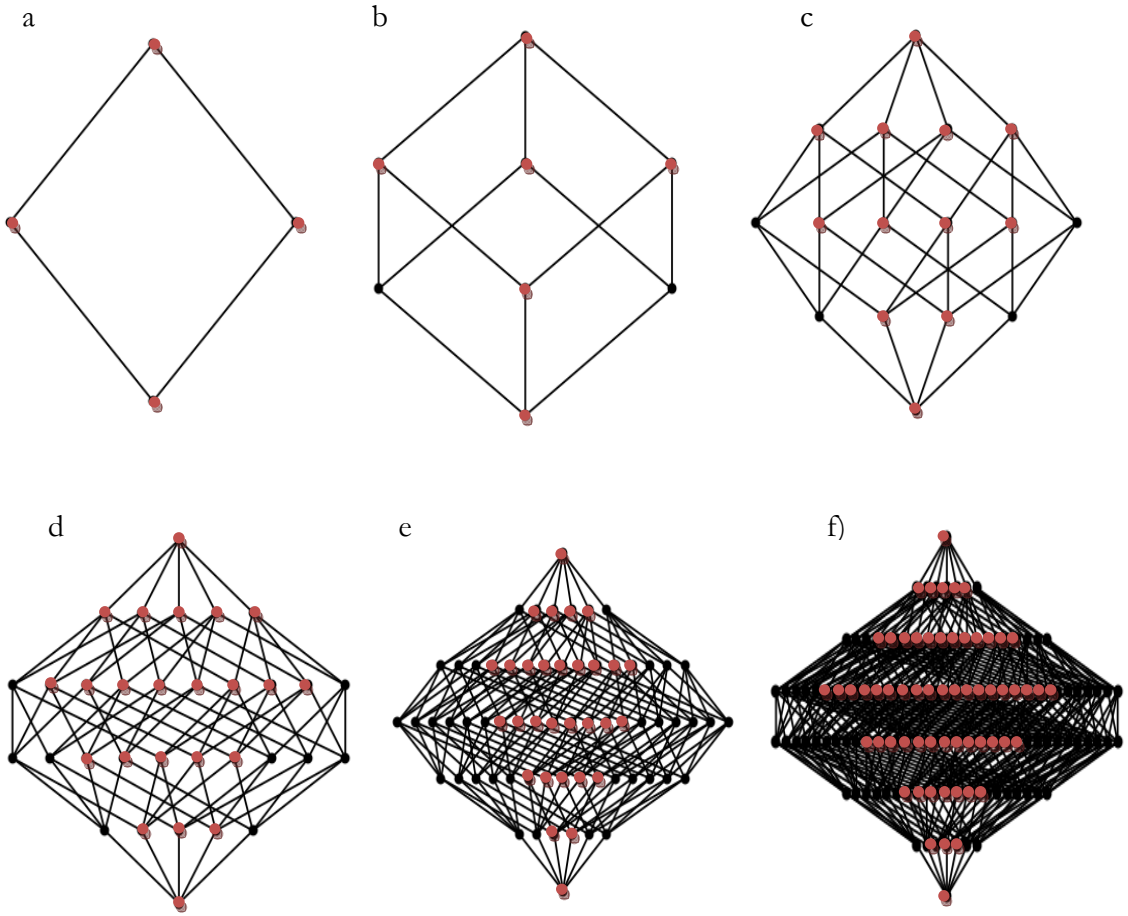


Figure 47: Index Ranking Search Space Reduction on the Tensor Tree Representation

Equation 25 : Fixed Band Distance Measure

$$\max_{i=0}^{k} \left| y_i - x_i \right| \leq \varepsilon$$

This norm is defined in literature as the supremum norm, the Chebyshev norm, or the infinity norm, which is the maximum absolute distance. It produces a fixed band with symmetrical increasing and decreasing curves at both the top and the bottom of the tensor graph (tensor lattice points as vertices of the graph, connected by neighbourhood on single index element change) as shown in Figure 48. The number of partitions per wave grows till it reached a fixed number that is a function of the dimensionality and the Epsilon value(ε), and then it remains constant for all middle waves, till it starts decaying again in the same pace as the initial growth. The maximum number of partitions for the dimension and the Epsilon values are shown in Table 16 and their exponential growth is very fast. This fast growth makes this distance measure not as sensitive as would have been desirable, but it scores very well due to its ability to define the partitions around the middle band through the tensor space efficiently. Table 16 shows the number of multidimensional neighbours of a cell for a given Epsilon value (ε) equals to (((ε*2)+1)k-1), restricted to neighbours that fall on the same wave number, i.e. the sum of the neighbour's index elements equals the wave number of the given middle partition cell index. The neighbours of a cell spans +k and –k waves around the cell's wave number, and the number of the neighbours falling on each of these waves, vary with the shape of the dataset, and the position of the cell itself. So, it is recursively computed without a generic closed formula on the bounds.

This section has presented different distance measures and highlighted some problems associated with them. Two distance measures were proposed, the index ranking and the supremum norm. The last one is adopted in the implementation. It measures a fixed maximum distance between each partition index element and the corresponding element in the middle partition of the same computation wave.

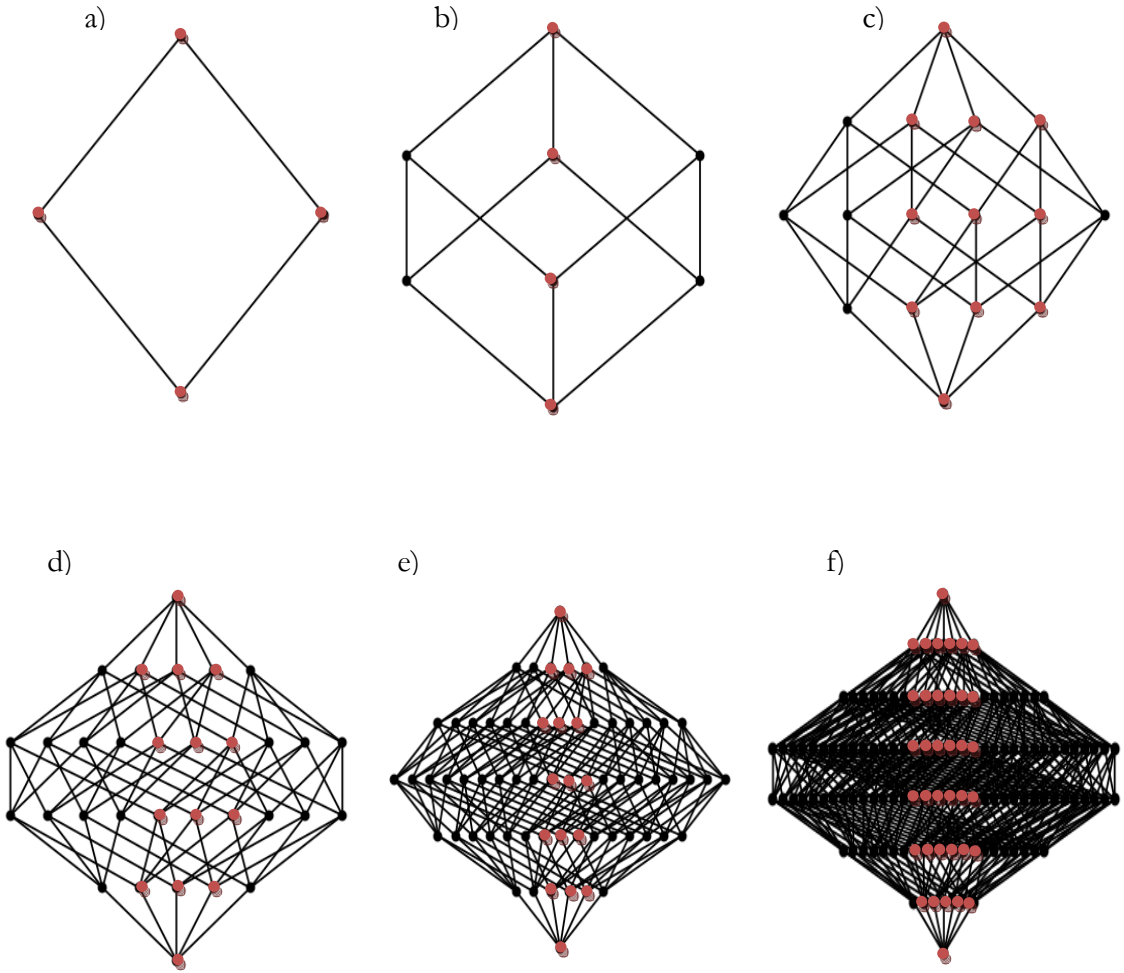


Figure 48: Fixed Epsilon Search Space Reduction on the Tensor Tree Representation

Table 16: Maximum Partitions in all waves per E value and dimensionality k.

	E	1	2	3	4	5	6	7	8	9	10	11	12	13	14
k															
3		7	19	37	61	91	127	169	217	271	331	397	469	547	631
4		19	85	231	489	891	1469	2255	3281	4579	6181	8119	10425	13131	16269
5		51	381	1451	3951	8801	13651	20851	31321	46211	66901	95001	132351	181021	243311

5.3 Simulated Results

The experiments reported here were run on a SunFire X2200 with 2xAMD Opteron quad processors of 2.3 GHz, 512 Kb L2 cache and 2 MB L3 cache on each processor, and 8GB RAM. Randomly created sequences have been selected to test the performance of varying the ε value. Table 17 describes the data selected, where *K* is the dimensionality (number of sequences), *L* is the sequence lengths, *C* is the computation cost represented in the number of cells of the scoring tensor, which is the product of the sequence lengths each incremented by 1 for the initial gap, *T* is the number of waves, and *P* is the total number of partitions all over the waves.

Table 17: Data Sets Used in the Experiments

Set #	K	L	C	T	P
1	2	20,20	441	12	100
2	2	30,31	961	29	225
3	3	21,21,21	9261	28	1000
4	3	31,31,31	29791	43	3375
5	4	31,31,31,31	923521	57	50625
6	4	41,41,41,41	2825761	77	160000
7	5	41,41,41,41,41	115856201	96	3200000

Table 18 shows the performance measure produced for varying the ε value for the chosen data sets (Set #), where ε is the epsilon value, % is the percentage of partitions computed from the total partitions shown in Table 17, ST is the system time (CPU time) in seconds, UT is the user time (wall time) in seconds, M is the process memory size in KB, SP and Ent. are the sum-of-pairs score and Entropy respectively of the produced alignment, and AL is the alignment length.

Table 18: Experiments Performance Results

Set #	ε	%	ST	UT	M	SP	Ent.	AL
1	1	51.00	0.04	0.03	146432	5	93.59	21
1	2	75.00	0.06	0.04	146432	5	93.59	21
1	3	91.00	0.07	0.03	146432	5	93.59	21
1	4	99.00	0.06	0.07	146432	5	93.59	21
1	5	100.00	0.07	0.04	146432	5	93.59	21

2	1	36.00	0.07	0.06	146432	5	164.35	31
2	2	55.56	0.08	0.04	146432	5	164.35	31
2	3	71.56	0.09	0.04	146432	5	164.35	31
2	4	84.00	0.10	0.41	146432	5	164.35	31
2	5	92.89	0.11	0.05	146432	5	164.35	31
2	6	98.22	0.11	0.07	146432	5	164.35	31
2	7	100.00	0.11	0.06	146432	5	164.35	31
3	1	17.20	0.11	0.09	146432	16	165.97	21
3	2	40.20	0.15	0.12	146432	16	165.97	21
3	3	65.40	0.21	0.84	146432	16	165.97	21
3	4	86.60	0.26	1.22	146432	16	165.97	21
3	5	97.80	0.32	1.04	146432	16	165.97	21
3	6	100.00	0.32	2.94	146432	16	165.97	21
4	1	8.21	0.06	0.18	244736	10	308.72	34
4	2	20.36	0.10	0.95	244736	5	345.36	37
4	3	35.82	0.23	0.36	244736	5	345.36	37
4	4	52.77	0.28	7.21	277504	5	345.36	37
4	5	69.30	1.04	3.73	277504	5	345.36	37
4	6	83.59	0.35	1.56	277504	5	345.36	37
4	7	93.72	0.39	2.80	277504	5	345.36	37
4	8	98.64	0.39	6.43	277504	5	345.36	37
4	9	99.94	0.40	4.21	277504	5	345.36	37
5	1	1.94	0.17	1.14	174	-4	538.90	39
5	2	7.81	0.44	15.34	447	-11	575.49	41
5	3	18.81	0.97	66.36	995	-11	575.49	41
5	4	34.68	1.90	616.28	1951	-11	575.49	41
5	5	53.84	2.96	206.47	3034	-11	575.49	41
5	6	73.33	4.11	322.89	4213	-11	575.49	41
5	7	88.84	5.28	473.04	5408	-11	575.49	41
5	8	96.76	6.43	495.34	6587	-11	575.49	41
5	9	99.46	8.54	748.25	8750	-11	575.49	41
5	10	99.98	7.47	579.96	7644	-11	575.49	41
5	11	100.00	7.46	503.38	7643	-11	575.49	41
6	1	0.85	0.30	5.71	277504	3	779.27	52
6	2	3.53	0.92	36.01	278528	-2	816.79	54
6	3	8.84	1.90	95.96	280576	-2	816.79	54
6	4	17.09	1.39	69.15	281600	-2	816.79	54
6	5	28.17	6.31	439.75	285696	-2	816.79	54
6	6	41.56	10.13	756.10	287744	-2	816.79	54

7	1	0.14	1.25	114.12	283648	-41	1143.83	55
7	2	0.97	9.66	907.05	327680	-116	1611.69	72
7	3	3.39	36.27	8344.97	327680	-219	2361.11	96
7	4	8.35	112.41	50208.94	454656	-219	2361.11	96
7	5	16.62	320.66	214003.73	608256	-219	2361.11	96

The results show that in the first 3 datasets, which are fairly small, the smallest search space produced exactly the same alignment scores (SP and Ent.) as the Full Search Space (FSS). As the data sets grows, in the fourth data set, the smallest search space value $\varepsilon = 1$, (which is 0.06% of the FSS), produced 10 SP with 34 AL, which is better than the steady SP of 5 with 37 AL that is produced using $\varepsilon = 2$ (which is 0.1% of the FSS) and up to the FSS. Same observation is shown in dataset 5, where $\varepsilon = 1$ (0.17% of the FSS) produced -4 SP with 39 AL, which is better than the -11 SP with 41 AL from $\varepsilon = 2$ (which is 0.44% of the FSS) and up to the FSS. In the largest dataset used, the $\varepsilon = 1$ (0.14% of the FSS) produced -41 SP with 55 AL; $\varepsilon = 2$ produced -116 SP with 72 AL, which is 0.97% of the FSS, and $\varepsilon=3$ (which is 3.39% of the FSS) up to the FSS produced -219 SP score with 96 AL. The smallest band around the hyper-diagonal forces the alignment to use fewer gaps and more substitutions, which in turn causes shorter alignment length and higher SP score because of the less gap penalty incurred.

As the band around the hyper-diagonal increases, there is more room for gap insertions, and hence, longer alignment lengths, and lower SP score. Based on the size of the dataset, the increase in the search space, doesn't affect the produced alignment. In the first few data sets, there was no change at all; then steady results were reached at $\varepsilon = 2$ for data set 4, 5 and 6, and $\varepsilon = 3$ for dataset 7. These results are plotted in Figure 49, where the graphs' titles are numbered after the dataset they represent. The left hand side of the figure illustrates the alignment scoring variations (Red: Entropy; Blue: Sum of Pairs score) on the y-axis over the change of the ε value on the x-axis. The right hand side illustrates the resource consumption of the experiments (purple: Process Size in Kbytes; Green: User Time in seconds; Orange: System Time in seconds) on a logarithmic scale over the change of the ε value on the x-axis. Each graph is numbered as the dataset of the experiment used as shown in Table 18.

The experiment conducted in (HELAL, M, Sintchenko, V, 2009) show that the proposed method score came third after TCoffee and MUSCLE, where similar sequences were aligned.

For the alignment of the less similar sequences, the current method scored the better than other methods tested. These results show that the presented method scores better when aligning sequences of large dissimilarity, and can be used to identify regions of high dissimilarity along the full length of the input sequences.

For global alignments of sequences of similar lengths, the experiments show that only 1% of partitions around the middle diagonal of the hyper-cube produce the same optimal results as scoring the full search space. For local alignments, other heuristics are to be applied, such as the selection of the starting points (high scoring areas) to score, and the sparse array heuristics. In addition, the rubber band method (TAHERI, J, Zomaya, AY, Zhou, BB, 2008) is used to identify these poles, and score the partitions around them.

5.4 Optimization

This section presents different optimization techniques that are implemented to either reduce the complexity or to increase optimality. The first subsection describes the current implementation of an affine gap scoring technique. The second subsection details the optimization of the execution parameters.

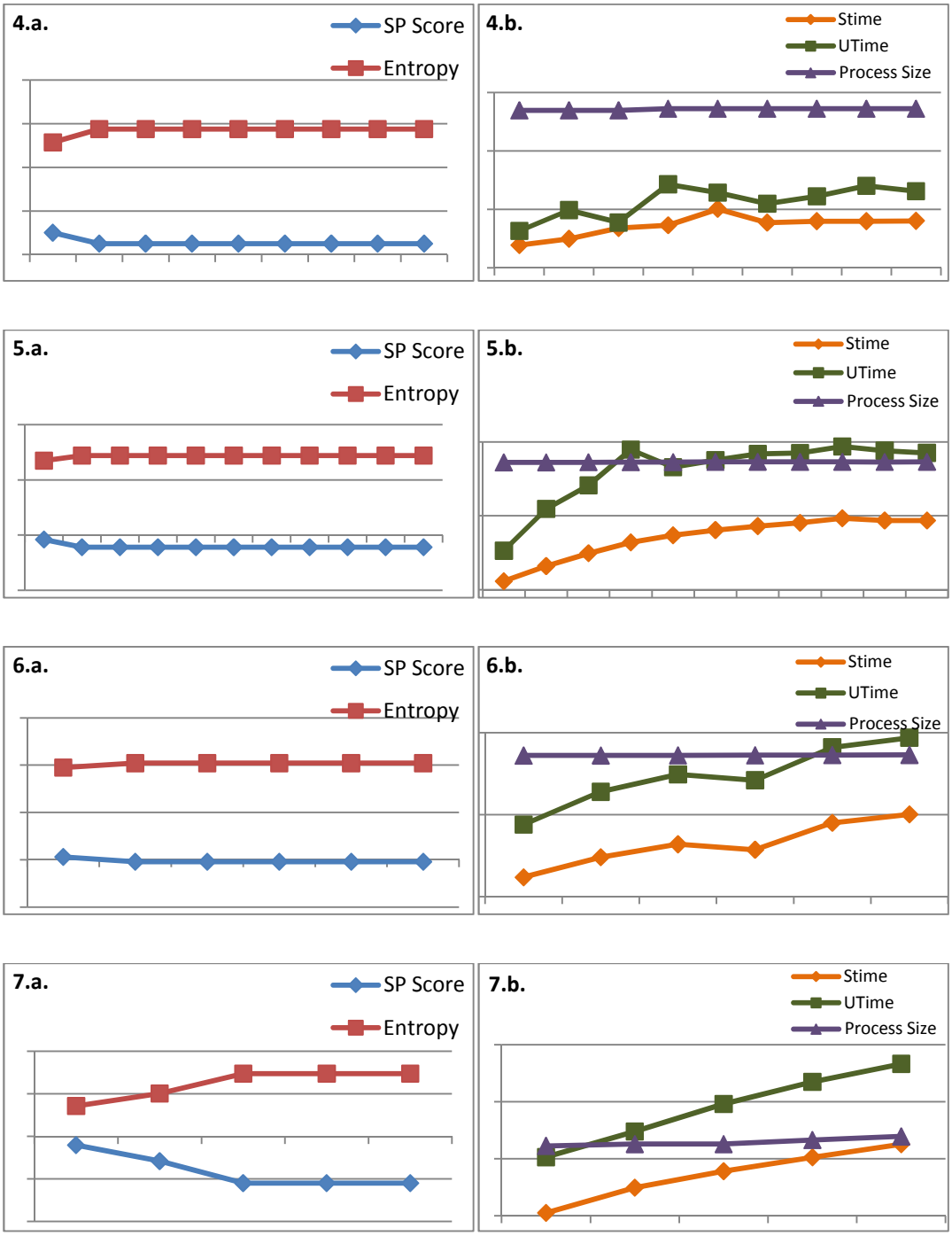


Figure 49: Performance Results for Search Space Reduction Technique Simulations: a) the alignment scoring variations (Red: Entropy; Blue: Sum of Pairs score), and b) resource consumption (System Time, U-Time, and Process Size). In all cases the x-axis plots the size of the diagonal band ε.

5.4.1 Affine Gap Penalty Scoring

A linear penalty score was applied in this thesis for the master/slave experiments, and the initial P2P experiments. An affine gap scoring method was implemented for later experiments described in Chapter 6. The design took care of the different gap insertions on all dimensions. The idea is based on testing the previous chain in each cell, and scoring using a gap extension for the dimension with decremented neighbour index. Otherwise, a gap opening score is used to start a new gap on the corresponding sequence. This is based on the assumptions used in the implemented scoring scheme as described in Chapter 3, where a decremented index element of a neighbour from the current scoring cell means there is a gap insertion in the corresponding sequence to that element's axe.

The initial implementation of affine gap as presented in (GOTOH, O, 1982) for pair-wise alignment required three matrices, one for each sequence to record the score if a gap is inserted vs. if it is not inserted, then records in the third matrix the accumulated result. A naïve generalization of this method to higher dimensions would have been to create a scoring tensor for each dimension, and another one for the accumulated score. The implemented affine gap scoring for MSA in this thesis avoided the use of external buffers for each dimension (sequence) to store the different scores if a gap is inserted vs. if it is not inserted in the corresponding sequence. It assumes that only the highest scoring neighbour that led to the score in this cell will be taken in the trace back for the final alignment. If more alignment paths need to be returned, then all possible scores need to be stored if or if not a gap is inserted in each corresponding sequence. Due to the memory space complexity, this has not been implemented and is left for future work.

5.4.2 Optimizing Execution Parameters

The execution time in a distributed environment is estimated by Equation 26 as defined by (STONE, HS, 1977)

Equation 26 : Distributed Execution Time Equation

$$dT = r \times \max\left(p_m\right) + \left(\left(\frac{c}{2}\right) \times \left(P^2 \sum_{m=0}^{V-1} p_m^2\right)\right)$$

where *dT* is the distributed overall time. This is composed of the maximum computation cost on the highest-loaded processor plus the communication cost of partitions communicating with each other as a mesh.

r is the execution time of a single partition on a processor, which is $O(S_k \times (2^k-1))$ for all cells in each partition, in each wave assigned to a processor *m*. It is multiplied by the maximum processor allocation max(*pm*). This is the allocation of the first processor; it is equal to $\left\lceil \frac{P}{V} \right\rceil + 2$; for simplicity, it is approximately calculated as $\left\lceil \frac{P}{V} \right\rceil$. *c* is the cost of communication between a single task on a processor and all other tasks on the other processors. This is equivalent to $O(S^k - (S-1)^k)$ per partition. This assumes a mesh of communication between all partitions. However, the P2P design shown in Chapter 4 limited each partition's communication to be with a maximum of 2^k-1 other partition. According to the scheduling scheme, only the edge partition in the cluster of partitions assigned to a processor communicate with other processors. For the local partitions, the search cost might be equivalent to the communication cost, based on the hardware design used. Equation 26 shows the worst case in communication. However, due to the operating system scheduling of processors in a single multi-core HPC, or equivalently, a network routing in a distributed cluster of computing nodes, it is hard to estimate a more accurate equation. *P* is the total number of partitions in all waves. The *r* and *c* are attributes of the hardware on which the execution is performed. Let *R* be the total computation time for all partitions assigned to a processor; and let *C* be the total communication done by a processor for all partitions assigned to it. *R* and *C* are easily calculated based on the partition size. The ratio *R*/*C* is the granularity of the design. Minimizing the total execution time takes place by altering these ratios.

The above equation is simplified and substituted by the partitioning variables as:

$$dT = S^k \times (2^k - 1) \times \left\lceil \frac{P}{V} \right\rceil + \left(\frac{\left(S^k - (S-1)^k\right)}{2} \times P^2 \times \left\lceil \frac{P}{V} \right\rceil^2 \right)$$

In order to make the execution time a function of the input variables (sequence lengths ρ of the k sequences), the partitioning size *S,* and the cluster size V only, *P* (number of total

partitions) is eliminated using the calculation for the full search space as shown in Equation 11. The resulting distributed time estimation equation is shown in Equation 27.

Equation 27: Optimization Equation

$$dT = \mathrm{S}^{\mathrm{k}} \times (2^{\mathrm{k}} - 1) \times \left\lceil \frac{\left(\frac{\prod_{i=0}^{k-1} \rho_i - 1}{(S-1)^k} \right)}{\mathrm{V}} \right\rceil + \left(\frac{\left(\mathrm{S}^{\mathrm{k}} - (\mathrm{S}-1)^{\mathrm{k}}\right)}{2} \times \left(\frac{\prod_{i=0}^{k-1} \rho_i - 1}{(S-1)^k} \right)^2 \times \left\lceil \frac{\left(\frac{\prod_{i=0}^{k-1} \rho_i - 1}{(S-1)^k} \right)}{\mathrm{V}} \right\rceil^2 \right)$$

Equation 27 needs to be optimized to minimize the distributed execution time, by altering the partitioning size S and the cluster size V. The main target of the optimization technique is the partition size. As shown in Figure 28, the ratio of the communication cost to the computation cost of a small partition size is higher compared with bigger partition sizes. An identified metric for the communication and computation cost for the target machine, can be used as constraints in the optimization process.

Another target of the optimization technique is the number of processors used. A larger number of processors are not always favourable. Each extra processor imposes an overhead, and therefore, reduces the total execution time. Consequently, the two main objectives are to minimize the number of processors, and to trade-off between the communication overhead and the computation cost. Since the design is well structured, a closed formula can be reached and fed to one of the well-known solvers in the different statistical packages. For this implementation, a simple Excel sheet is used. The data size input is entered, and the partitioning equations are then implemented. The function "*goal-seek*" is used on the target value to be optimized, by changing the partition size value. A specific relationship between the machine cache size and the maximum partition size that can be used should be determined empirically. Similarly, the maximum allowed execution time; the recorded communication speed and any further constraints can be added to the model. The current Excel sheet implements the equations used in Chapter 4 and uses goal seek function to optimize the Partition Size (S) or the Processors (Cluster size – V) as required. An example of this excel sheet is shown in Table 19, for a dataset of k=3 and shape = {1648, 2517,

2665}. It is shown that the minimum communication and computation is achieved at the highlighted row, where the partition size is 224, and the number of processors that can be employed without leaving any idle processors (maximum number of partitions in one wave) is 35.

Table 19: Excel "Goal Seek" Optimisation Example

S - Partition Size	R - Comp.	t -Waves No.	P - Partitions No.	C - Comm.	R/C	V - Cluster Size	Total Comp.	Total Comm.
3	11054462640	3411.5000	1379902716	15181397637	0.0909	404486	260801613324	26218151604
224	**11054462640**	**28.6608**	**1000**	**11195**	**0.0893**	**35**	**78320380993**	**149404129**
132	11054462640	50.0009	4879	54229	0.0900	98	79053825167	254181868
115	11054462640	57.8860	7452	82714	0.0901	129	79325983177	293061583
55	11054462640	124.4259	70107	774539	0.0905	564	81647574081	624717427

5.5 Conclusion

This Chapter has presented a search space reduction method that is suitable for the parallel dynamic programming method presented in Chapter 4. The search space reduction method employed the novel tensor indexing scheme used in this thesis for partitioning the scoring tensor for the parallel processing of the DP MSA problem. This tensor indexing scheme has enabled another view for the internal points of the tensor space, based on a novel formulation of a hyper-diagonal. Accordingly, distance measures have been investigated to identify the norm that can measure the distances between the middle partition and the near-by ones in each wave of computation across the logical diagonalization of the tensor. The bounds created around the middle hyper-diagonal are the tightest presented in literature, and the approach is not based on pair-wise projections that are less sensitive and trimming on outer tensor borders only and only useful for sequences of higher similarity. The presented method starts from the hyper-diagonal and increases the band around it until steady alignments scores are produced.

The initial results show that scoring less than 1% of the full search space can produce the exact optimal score as scoring the full search space. This is justified by the scoring function that calculates all possible paths from the origin up to the current scoring cell that fall within

the search space considered, and average the missing neighbours. This DP scoring method is more sensitive than the sum-of-pairs method, and this is why a very small band around the middle of the tensor was sufficient for global alignments. However, the main drawback if this scoring recurrence is the exponential growth of the visited neighbours as the number of sequences (k) increases.

Chapter 6: Experiments on Biological Sequences

This Chapter presents the results of three experiments conducted using real biological sequences. Various objectives for the experiments are highlighted and performance measures are explained. The main objective is to test the performance on larger sequences. The biological interpretation or meaningfulness is a function of the data formulation, scoring scheme used, and approximation parameters. The biological interpretation and its correctness remain open for debate, and beyond the scope of this thesis. As mentioned in Chapter 2, mainly two measures of performance are used, which are the sum-or-pairs and the Entropy. Other visualization tools are used like similarity regions plots, dendrograms and heat maps.

The experiments are done using DNA sequences for bacterial species. The first experiment focuses on testing the performance of mmDst, the tool presented in Chapters 4 and 5. The performance of mmDst is compared to other heuristic methods in aligning highly similar sequences of different species. The second experiment uses a dataset of highly divergent sequences that are known to have different reactions. The objective of the second experiment is to identify regions within the sequences sets that are responsible for the different behaviour. Section 3 concludes the Chapter.

6.1 Mycoplasma Highly Similar Sequences Experiment

This section presents an experiment conducted to identify distinct families of closely related organisms (HELAL, M, El-Gindy, H, Gaeta, G, Sinchenko, V, 2008). The rpoB gene, or the DNA-dependent RNA polymerase gene, is proposed as both a genome similarity predictor and as an alternative to 16S rDNA gene sequencing for biodiversity studies. Therefore, the nucleotide sequences of rpoB gene from Mycoplasma species are chosen as the validation set since Mycoplasma is considered the simplest free-living organism with the minimal genome. The GenBank accession numbers used in this study are AY191418, AY191419, AY191420, AY191421, AY191423, AY191424, AY191425, AY191433, and AY191437. Sequences are aligned using the proposed mmDst algorithm, and existing methods such as CLUSTAL W, MUSCLE and TCoffee. The algorithm is tested on a SunFire X2200 machine. This includes 2xAMD Opteron quad processors of 2.3 GHz, 512 KB L2-cache and 2 MB L3-caches on each processor, in addition to 8GB RAM. The sequences are aligned in the full search space,

scoring all partitions, using 8 processors and a partition size of 20. The following penalties are applied: a flat gap penalty of -2, a mismatch score of 0, and a match score of 2 in contrast to an affine gap (a minimum of 0.05 for gap extension) for CLUSTAL W, a gap opening penalty of 1 and 0 employed by MUSCLE and TCoffee, respectively.

To measure the performance of mmDst compared to the other methods, the sum-of-pairs scores with the above scoring scheme are 5003, 6233, 6467 and 6517 for CLUSTAL W, TCoffee, mmDst and MUSCLE, respectively. mmDst consistently produces alignments with higher scores than both CLUSTAL W and TCoffee, and very comparable to MUSCLE. Table 20 reports the weights of sequences from the sequence alignments using different methods as calculated by MASH (http://timpani.bmr.kyushu-u.ac.jp/~mash/weight_ex.html). It is important to measure the weight of a particular sequence in the produced alignment. A standard deviation measure is provided to emphasize the correctness of mmDst in producing alignments that are not much different from the established methods.

Table 20 : Sequence Weights from the Different Alignments (Overall Std. Dev.: 0.028351)

	Difference Method					Henikoff-Henikoff method				
	MmDst	CLUSTAL W	TCoffee	MUSCLE	Std. Dev.	MmDst	CLUSTAL W	TCoffee	MUSCLE	Std. Dev.
M.arthritidis	0.098665	0.105587	0.101317	0.100486	0.002933	0.084044	0.089908	0.091009	0.089298	0.003096
M.bovi	0.106124	0.107244	0.107623	0.105600	0.000945	0.083984	0.097180	0.094332	0.092975	0.005699
M.buccale	0.093038	0.090909	0.092209	0.092304	0.000885	0.069931	0.084373	0.080028	0.079753	0.006105
M.fermentans	0.105862	0.108665	0.110706	0.107262	0.002067	0.087820	0.102429	0.097179	0.095294	0.006048
M.gallisepticum	0.132819	0.125947	0.134529	0.134620	0.004106	0.170381	0.168571	0.175023	0.186921	0.008258
M.genitalium	0.130463	0.126894	0.127663	0.129890	0.001718	0.151906	0.135856	0.141880	0.139587	0.006864
M.hominis	0.100628	0.103456	0.098374	0.098057	0.002496	0.079683	0.090872	0.087999	0.084857	0.004791
M.pneumoniae	0.136613	0.132576	0.133268	0.138456	0.002782	0.199205	0.143638	0.149545	0.152684	0.025568
M.salivarium	0.095786	0.098722	0.094311	0.093327	0.002352	0.073045	0.087172	0.083005	0.078630	0.006051

Table 21: Mycoplasma Distances Measures Based on the Different Alignment Methods.

	M. arthritidis	M. bovis	M. buccale	M. fermentans	M. gallisepticum	M. genitalium	M. hominis	M. pneumoniae
M. bovis	{0.3503, 0.3591, 0.3591, 0.2626} :0.047	0						
M. buccale	{0.1909, 0.1890, 0.1890, 0.1572} :0.0162	{0.3122, 0.3117, 0.3117, 0.2386} :0.0366	0					
M. fermentans	{0.3555, 0.3729, 0.3729, 0.2728} :0.0479	{0.2673, 0.2673, 0.2673, 0.1387} :0.0643	{0.3047, 0.3091, 0.3091, 0.2229} :0.0424	0				
M. gallisepticum	{0.9025, 0.8973, 0.8336, 0.4032} :0.2394	{1.0203, 0.9518, 0.8968, 0.4602} :0.2531	{0.8486, 0.8159, 0.7080, 0.3184} :0.2437	{1.0345, 1.0034, 1.0109, 0.4535} :0.2817	0			
M. genitalium	{0.8574, 0.8529, 0.7412,	{1.0037, 0.9151, 0.8103,	{0.8717, 0.8119, 0.6954,	{1.0147, 0.9900, 0.9625,	{0.3593, 0.4300, 0.3980,	0		

	0.388} :0.2212	0.4822} :0.2279	0.3429} :0.2367	0.451} :0.2699	0.2628} :0.0725			
M. hominis	{0.2719, 0.2644, 0.2644, 0.2325} :0.0176	{0.3144, 0.3074, 0.3074, 0.2416} :0.0342	{0.1836, 0.2007, 0.2007, 0.1676} :0.0159	{0.3535, 0.3406, 0.3406, 0.2536} :0.0461	{0.9131, 0.8200, 0.8049, 0.3946} :0.2307	{1.0400, 0.9447, 0.8392, 0.4268} :0.2700	0	
M. pneumoniae	{0.9374, 0.9705, 0.8110, 0.4244} :0.2506	{1.1032, 1.0242, 0.8681, 0.5006} :0.2674	{0.9748, 0.9249, 0.7524, 0.4051} :0.2577	{1.0645, 1.0856, 0.9357, 0.4606} :0.2916	{0.4237, 0.5514, 0.5052, 0.3181} :0.1023	{0.2529, 0.2529, 0.2532, 0.2322} :0.0104	{1.1443, 0.9547, 0.8389, 0.4451} :0.2953	0
M. salivarium	{0.2385, 0.2321, 0.2321, 0.2159} :0.0097	{0.2938, 0.2938, 0.2938, 0.2162} :0.0388	{0.1788, 0.1826, 0.1826, 0.1513} :0.0151	{0.2963, 0.2963, 0.2963, 0.2054} :0.0454	{0.9954, 0.8153, 0.8287, 0.3882} :0.2591	{0.9106, 0.8433, 0.7435, 0.3958} :0.2289	{0.2063, 0.2200, 0.2200, 0.1736} :0.0219	{0.9724, 0.8715, 0.7802, 0.4304} :0.2356

Table 21. This is the distance measures matrix for the sequences used in the experiment. Each cell contains the distance measures between the strain in the first column and the strain in the first row as produced by the different methods in the format: ({mmDst, MUSCLE, TCoffee, CLUSTAL W} : std. Deviation). The overall standard deviation was calculated as 0.1116.

MSA algorithms identify different relationships between Mycoplasma species as shown in Figure 50 for dendrograms and Figure 51 for rpoB gene regions of similarity. The main difference between the clustering of the above MSA methods was around the clustering of M. Salivarium, M. buccale, M.arthritidis, and M. Hominis. Both TCoffee and MUSCLE cluster them in the mentioned order respectively, while Clustal W and mmDst cluster them in the order: M. Salivarium, M. buccale, M. Hominis , and M.arthritidis. Further analysis is required to investigate the correct placement of sequences in the respective species.

In the figure, the darker the blue regions, the higher the similarity between the corresponding strains. On the other hand, the darker the red region, the higher the dissimilarity between the corresponding strains. Figure 52(b) reflects the degree of discrimination that the mmDst distance matrix generates in comparison to the one generated by CLUSTAL W. This is due to the fact that the blue regions colours can easily be clustered away from the red ones.

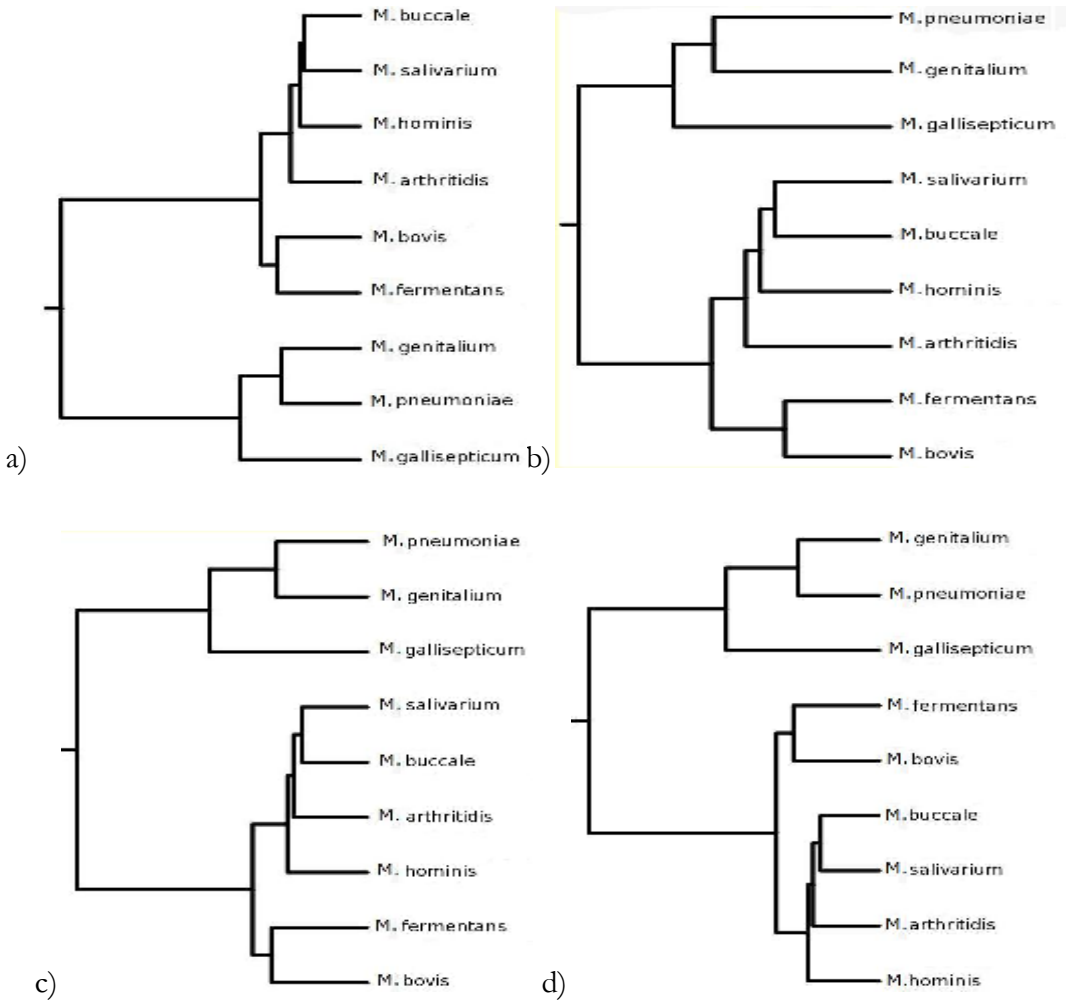


Figure 50: Dendrograms Produced by mmDst (a), CLUSTAL W (b), TCoffee (c), and MUSCLE (d).

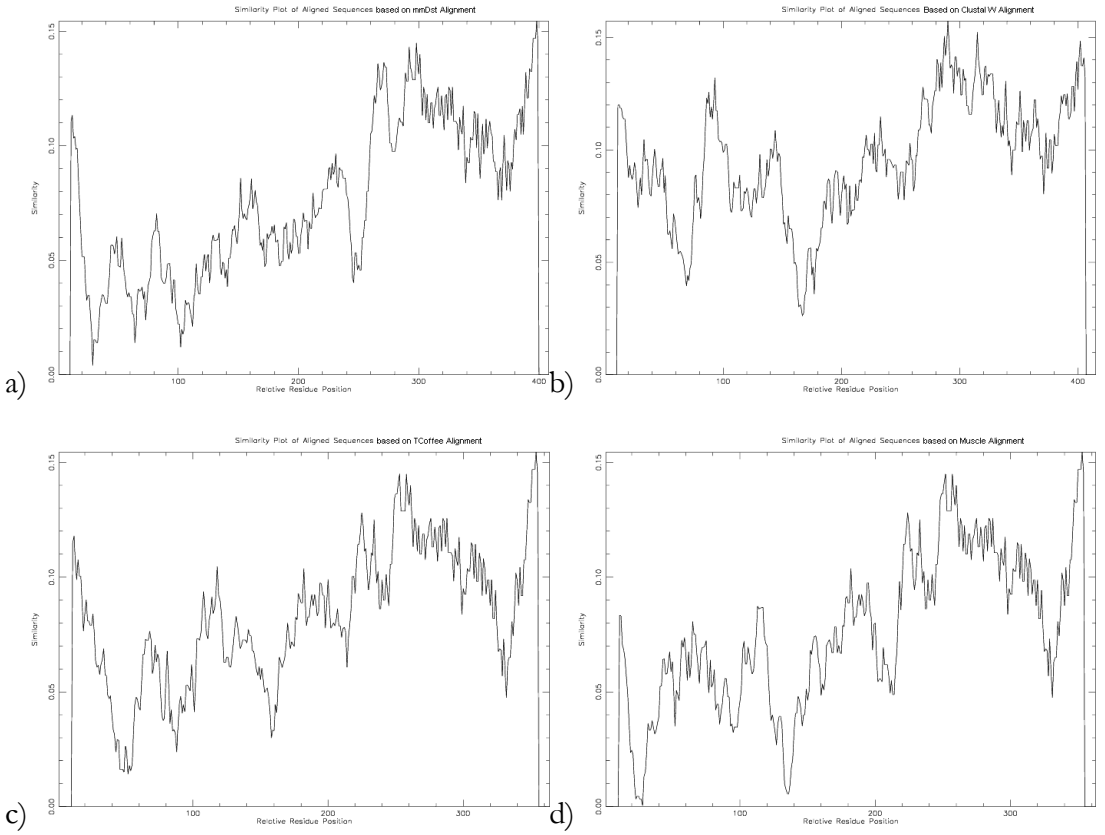
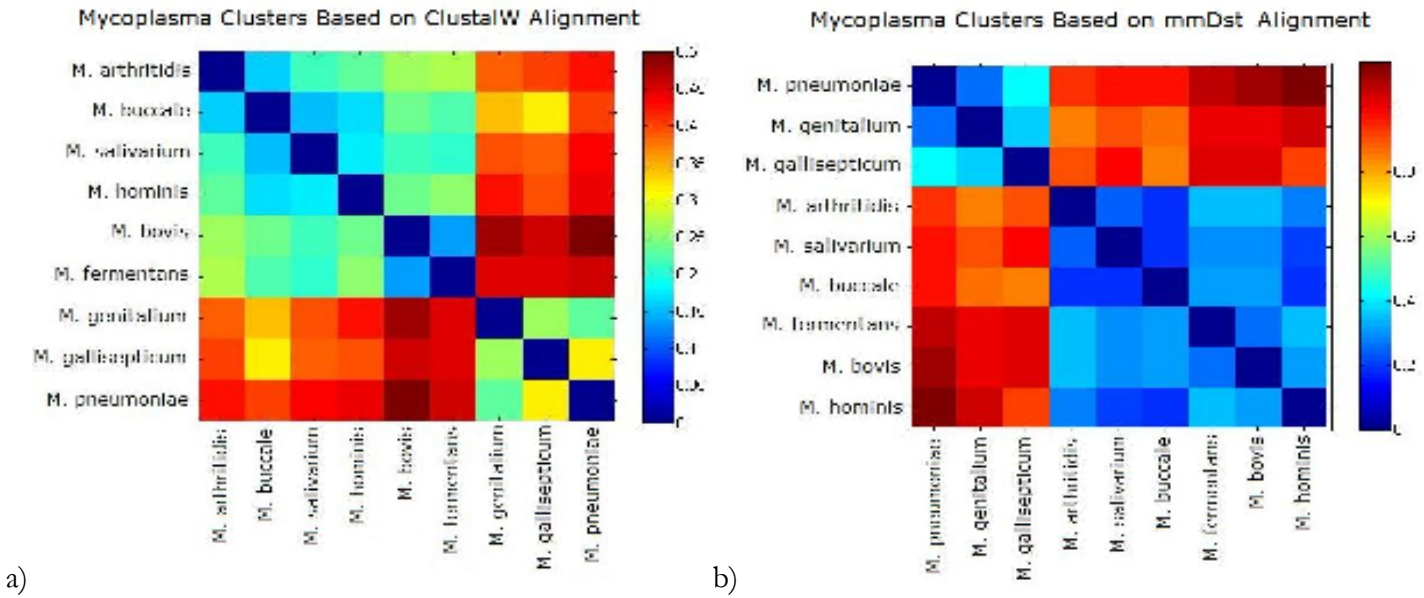


Figure 51: Similarity Regions' Plots: mmDst (a), CLUSTAL W (b), TCoffee (c), MUSCLE (d).



a) b)

Figure 52: Mycoplasma Clusters Visualized from the Distance Matrices Generated by CLUSTAL W (a) and mmDst (b).

This experiment showed that the presented method "mmDst" produced consistently better quality alignments than CLUSTAL W, and comparable to MUSCLE and TCoffee in the study of *rpoB* gene sequences of *Mycoplasma* species. The method can optimise alignments of nucleotide or amino acid sequences for comparative genomics of infectious diseases, for the identification of infectious pathogens based on gene sequencing and for sequencing-based analysis of microbial virulence and drug resistance. The potential use of the generated alignment for clustering the dataset is investigated using the heat maps sorting on the distance matrices generated from the alignment. The more sensitive the alignment, the more efficient the clustering will be.

6.2 Divergent Bacterial Sequences Experiment

Another experiment was conducted, as reported in (HELAL, M, Sintchenko, V, 2009), to identify "quinolone-resistance determining regions" (QRDRs), which are the highest dissimilar regions from an alignment of sequences from different bacterial species. The data set chosen are of two classes. One class is known to be sensitive to quinolone class of antimicrobials (labelled as sensitive sequences in this experiment), and the other class is known to be resistant (labelled as resistant sequences). The sequences were aligned on a reduced search space factor "epsilon" equals 1, which represents 0.21% of the search space for the sensitive sequences and 0.19% of the resistant sequences. Then for each alignment, a consensus sequence is generated to show which residues are most abundant in the alignment at each position, using the "cons" program from the EMBOSS suite of Bioinformatics programs. (RICE, P, Longden, I, Bleasby,A, 2000). The pair-wise alignments of the consensus sequences were done in full search space.

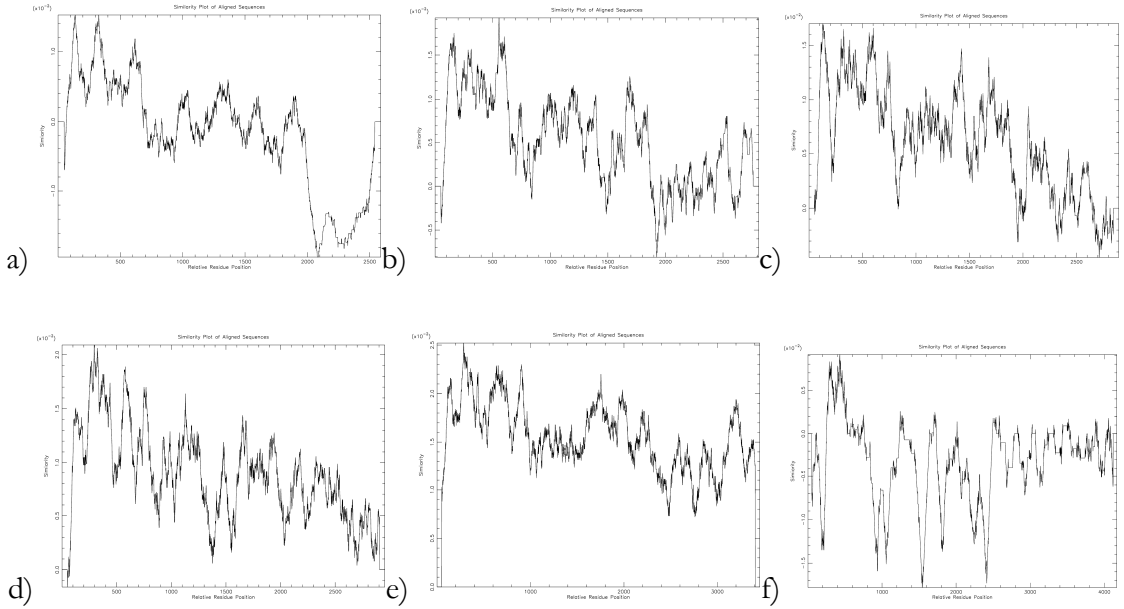


Figure 53: Similarity Regions Plot of the Alignment of the Consensus Sequence of the Sensitive Sequences with the most Resistant Sequence "Treponema pallidum"

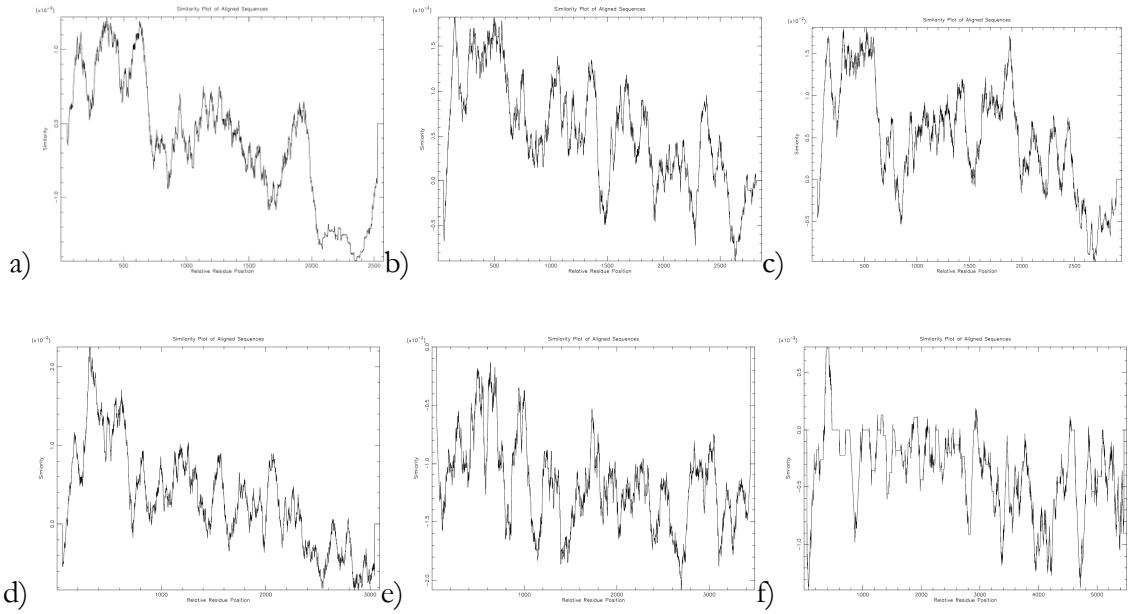


Figure 54: Similarity Regions Plot of the Alignment of the Consensus Sequence of the Sensitive Sequences with Consensus Sequence of the Resistant Sequences

The similarity regions plots in Figure 53 and Figure 54 illustrate the similarity score (on the y-axis) averaged on a window size of 100 base pairs along the length of alignment (on the x-axis). Figure 53 plots the alignment of the consensus sequence of the sensitive sequences with the most resistant sequence "Treponema pallidum", and Figure 54 plots the alignment of the consensus sequence of the sensitive sequences with consensus sequence of the resistant sequences. Both figures show the similarity regions for using the six different methods: a) mmDst, b) MUSCLE, c) TCoffee, d) CLUSTAL W, e) Kalign, f) MAFFT. The higher the curve in the plot, the more similar (has matching residues) is this region, and the lower the curve, the more dissimilar (full of gaps or many substitutions) is the region in the alignment. Since the sequences used in this experiment are of opposite behaviour (sensitive vs. Resistant to quinolone), the objective is to identify regions of highest dissimilarity so that the mutations in this region can be considered responsible for the opposite behaviour. A flickering plot like the ones produced by Kalign might not be able to identify contiguous similar or dissimilar plots. These plots show that aligning the consensus of one species as compared to another consensus sequence of a different species show less sensitive plot than the consensus of one species as compared to one sequence of a different species that has the strongest opposite function as shown in Figure 53. This figure identifies regions of highest and lowest similarities with longer regions of contiguous similarity and dissimilarity. In both figures, mmDst identifies clear contiguous similar regions of similarities and dissimilarities better than plots by other methods.

Table 22 lists the sum-of-pairs scores for the alignments produced by the different methods for the sensitive sequences alignment ("Sensitive Seq" column), resistant sequences alignment ("Resistant Seq" column), sensitive sequences consensus alignment with the most resistant sequence "Treponema pallidum" ("Sen & TP" column), and sensitive sequences consensus and resistant sequences consensus alignment ("Sen & Res Cons" column).

Table 23 lists the Entropy values for the same alignments produced in this experiment using the same column names as in Table 22. The results in Table 22 and Table 23 show that the proposed method "mmDst" score came third after TCoffee and MUSCLE in the first two cases, where similar sequences were aligned. However, in the third case where the consensus sequence of the alignment of the sensitive sequences were aligned with the most resistant sequence which is "Treponema pallidum", mmDst score came second after MUSCLE. In the fourth case, which is the alignment of the consensus sequence of the set of sensitive

sequences with the consensus sequence of the set of resistant sequences, mmDst scored the highest over all other methods. These results show that mmDst scores better when aligning sequences of large dissimilarity, and can be used to identify regions of high dissimilarity along the full length of the input sequences.

Table 24 shows the highest and lowest regions of similarities identified by the alignment of the sensitive sequences consensus sequence to the "Treponema pallidum" sequence as per alignment method. Further in-vitro analysis is necessary to determine which method best identifies the region that has the mutations responsible for the behaviour tested.

These experiments aimed at a direct comparison of six MSA algorithms highlight significant challenges in comparative genomics of pathogens. The majority of high-quality algorithms are too computationally expensive to be implemented in routine diagnostic laboratories. As discussed in Chapter 2, the existing methods are sensitive to the order in which the input sequences are arranged, such as a different ordering of sequences generates different alignments. This is due to the fact that a guide tree is calculated depending on that order. Progressive methods rely on pair-wise alignments, which is less sensitive than simultaneous alignment. This is because pairs of already aligned subfamilies (or closely related sequences) are calculated first, and there is usually more than one optimal alignment of the pairs and the choice of one of them might not be the optimal for the other pair-wise alignments nor has the highest biological relevance. The pair-wise alignments are also dependent on the parameters used in the calculations and the parameter changes are not reflected in the resulting MSA optimal alignment. The pair-wise alignments can be biased (GOLUBCHIK, T, Wise, MJ, Easteal, S, Jermiin, LS, 2007) because of the positioning of gaps and statistical uncertainty (WONG, KM, Suchard, MA, Huelsenbeck, JP, 2008). The new method "mmDst" scored better for more divergent sequences in this experiment, because it employs an innovative simultaneous alignment scoring recurrence.

Table 22: Mycoplasma Experiment Sum-of-Pairs Scores.

	Sensitive Seq	Resistant Seq	Sen & TP	Sen & Res Cons
mmDst	339	2231	849	1582
MUSCLE	439	3216	640	1123
TCoffee	443	2881	520	1025
CLUSTAL W	222	1966	478	1469
Kalign	-1593	-716	-285	1389
MAFFT	-3647	-4712	-1670	-2114

Table 23: Mycoplasma Experiment Entropy Scores.

	Sensitive Seq	Resistant Seq	Sen & TP	Sen & Res Cons
mmDst	23869.74	27932.28	15362.36	15430.20
MUSCLE	26855.33	25144.80	16815.42	17264.06
TCoffee	27246.99	25682.06	17355.43	17797.34
CLUSTAL W	28362.00	28240.50	17753.33	18836.40
Kalign	33336.24	34849.35	21156.91	22011.64
MAFFT	37834.49	40707.34	26597.46	37938.70

Table 24: Highest and Lowest Regions of Similarity in Quinolone Resistance Experiment..

		Sensitive consensus sequence & the "Treponema pallidum" Alignment			Sensitive consensus sequence to the resistant consensus sequence alignment		
		Score	From	To	Score	From	To
mmDst	Max	0.64	151	351	0.80	272	472
	Min	0.07	2167	2367	0.33	1567	1767
MUSCLE	Max	1.03	450	650	1.11	356	556
	Min	-0.83	2375	2575	-0.89	2668	2868
TCoffee	Max	1.06	233	433	1.15	430	630
	Min	-0.98	2495	2695	-0.36	2737	2937
CLUSTAL W	Max	0.98	233	433	1.05	292	492
	Min	-0.25	2176	2376	0.18	2589	2789
Kalign	Max	0.54	135	335	0.90	3265	3465
	Min	-0.54	3253	3453	-0.17	0	200
MAFFT	Max	0.52	101	301	0.58	3915	4115
	Min	-1.72	2654	2854	-2.00	427	627

6.3 Conclusion

The experiments conducted on real biological sequences revealed different scenarios where the "mmDst" approach can be recommended for use, in place of existing methods. In the first experiment it was shown that mmDst produced consistently better quality alignments than CLUSTAL W, and was comparable to MUSCLE and TCoffee in the study of *rpoB* gene sequences of *Mycoplasma* species as closely related sequences. In the second experiment, mmDst scored better for more divergent sequences. This arises because of the innovative simultaneous alignment scoring recurrence that mmDst implements. The implementation of mmDst allows for protein alignments that are not yet thoroughly tested on real data like Balibase benchmarks. All experiments are for global alignment (all over the sequences length). Further testing for local alignments (finding repeated motifs) is left for future work as mentioned in Chapter 7.

Chapter 7: Conclusion and Future Work

As mentioned in Chapter 1, the availability of parallel processing technology through various architectures requires research into techniques for algorithm development to address all requirements for developing applications to take full advantage of these new technologies. The main obstacles to parallelization and distributed solutions are associated with data partitioning and load balancing. The work in this thesis has successfully presented a high dimensional (tensor space) partitioning approach that was found to have inherent load balancing. The methods are generic and reconfigurable to target different platforms and architectures. The principles can be used to solve all similar problems of the same class as enumerated in the first section of Chapter 1. This Chapter summarizes the contributions of this thesis in the first section and provides ideas for future work in the second section.

7.1 Conclusion:

The detailed contributions of this thesis have been listed in Chapter 1. We now highlight specific aspects of those. The contribution of the algorithm presented in Chapter 4, together with the search space reduction techniques of Chapter 5, lies in the reconfigurable partitioning schemes of a tensor space. The importance of this comes from its ability to divide the computation complexity of the high-dimensional spaces over processors. In addition, communication requirements are clearly modelled. An automatic load balancing scheme is found to be inherent in the methods used by clustering the partitions based on a scalable high dimensional neighbourhood function. Each cluster of partitions is assigned to the same processor in order to reduce communication.

In this solution, parallelism scalability grows exponentially with the data size, which allows effective utilization of more and more processors. The upper bound of the data size is based on the configuration of the used cluster, grid, or HPC, as well as the number of processors used. Implementation is discussed using MPI, knowing that this environment is supported in both shared and distributed memory platforms. OpenMP can be used instead of MPI for shared memory architecture and performance can be compared. The partitioning allows the analysis at all levels given the communication data transfer speed as well as the computing node memory capacity and computing speed. Consequently, the proposed solution is suitable for high-performance computers with both shared and distributed memory and program

execution can be fine-tuned to the hardware configuration. In addition, it can be implemented on clusters of computing nodes with high speed networks. Moreover, it could also be applied on P2P networks with heterogeneous nodes.

Considering the MSA problem, this thesis contributes multi-processor master/slave cubical partitioning and P2P diagonal partitioning designs of the MSA scoring tensor as generalized from the dynamic programming algorithm for pair-wise alignments. The P2P results in significant performance improvements over both sequential and master/slave implementations. The former produces higher-scoring alignments as compared to the commonly used heuristic methods, specifically CLUSTAL W, and competes with the alignments produced by MUSCLE and T-Coffee. A search space reduction heuristic is developed to score only the partitions along or close to the hyper-diagonal. Scoring a very small band of partitions produced alignments of scores that are still higher then CLUSTAL W in many cases as presented in Chapter 6. The search space reduction methods reduced the exponential growth in the search space (number of cells to be scored). However, the exponential growth in the number of neighbours to be evaluated in scoring each cell is still the bottleneck in the search space reduction method presented. In the next section, a scoring heuristic is discussed based on the sum-of-pairs score instead of the dynamic programming scoring.

Further optimization techniques were used to speed up the processing while using minimal memory. Time optimization techniques were used such as merging loops of the same strides. Communication optimization techniques were used such as packing all communication generated in one wave to one processor into one MPI send and receive operation to reduce the communication TCP/IP overhead. This packing was made feasible due to the P2P partitioning where the dependency inside the same wave was eliminated. Another communication optimization employed barriers to synchronize processors to be in the same computation wave. Space optimization was used to keep only the current partition space in memory and its needed communication buffers, and releasing the extra buffers outside the current partition scope.

In summary, the impact of the thesis as identified in Chapter 1 has been supported by the implementations discussed in Chapters 3, 4 and 5, and the results discussed in Chapter 6. The novelty of the work presented is supported by the literature review in Chapter 2 that demonstrated all existing MSA methods from the literature and the problems with those

methods. The literature review explained the availability of new hardware technology that needs to be efficiently used in algorithm design and new data structures to solve problems that were previously labelled as intractable.

7.2 Future Work:

Some ideas are elaborated at the final stages of the experimentation. They are worth documenting for future work. The future research directions based on the presented methods can be summarized in the following subsections.

7.2.1 Separate Executions for Each Computation Wave

The execution of the scoring program once for every computation wave can be considered to avoid the existence of any idle processor. In each execution, the allocated processors should be equal to the number of partitions in the current run. This needs a careful change in the dependency calculation and in the trace back algorithm, as the hashing function of defining the processor that computed a specific partition based on the partition index, takes into account the cluster size (number of available processors). This new function will need to consider the wave number and the corresponding available processors for that particular wave.

7.2.2 Reduce Memory Usage

Develop further techniques to reduce memory usage. Current implementation keeps in memory overlapping cells for k waves only. Similar optimization needs to be done for the waves and partitions. Currently all wave information is stored in memory, while only wave information for the next k waves is needed while scoring the current partition, to identify where to send the overlapping cells. This optimization is required to be able to run these methods on HPC machines like the blue gene where only a 512 MB of memory is available per processing node.

7.2.3 Optimize for Memory Hierarchies on Single Processors

There are many chances to reduce cache misses, whether for completely sequential processing, or in the scoring of partitions in parallel processing. The work in (MULLIN, LR, Luo, X, Bush, L, 2003) studies the usage of MoA reshape/transpose functions to arrange the data in memory hierarchies to reduce cache misses and enhance the performance. However, the work in this thesis was more focused on parallel processor and dependency analysis, and this enhancement is left for future work.

7.2.4 Other HPC Architectures

The tests conducted in this thesis were performed on machines with tightly-coupled processors (multi-cores) or machines with homogenous processors on a high speed local area network like the SGI Altix cluster. Further tests on different machine architectures and/or open grid or cloud of computers are encouraged in order to reveal more characteristics and more optimization chances.

The processor alignment assumed in this thesis is linear (array or vector of processors) of equal communication cost from any processor to another. Further work can design the machine architecture as a MoA structure describing the physical alignment of the processors, and the corresponding cost of communication. Scheduling in this scenario would be based on Ψ-Correspondence theorem (MULLIN, LR, 1992) that relates the data alignment among processors and adjacent dependent processors based on the communication cost between each pair of processors.

7.2.5 Further Scoring Heuristics

In order to avoid the exponential growth of computation steps per cell with the data size, a heuristic scoring function can be developed that doesn't require the $O(2^K)$ computation steps for every cell in the tensor. For example the sum-of-pairs scoring can be used instead of the presented high dimensional dynamic programming scoring recurrence.

The reduced search space method presented in Chapter 5 defined a band around the hyper-diagonal where the partitions are scored. This limits the growth of the scoring space to the addition of the sequences lengths rather than their multiplication. However, the exponential growth in the dynamic programming scoring recurrence, limits the data size scalability of the

solution. This is because of the exponential growth of visited neighbours to score each cell in order of (2^k-1). Alternatively, a sum-of-pairs score can be used as calculated in Equation 3.

The trace back algorithm can be implemented as the Rubber Band method (TAHERI, J, Zomaya, AY, Zhou, BB, 2008) to connect the poles of the highest sum-of-pairs score, to maximize the overall score, and avoid local optima.

7.2.6 Other usage scenarios

Further experiments for protein sequences and local alignments have not been thoroughly conducted yet. The protein sequences differ in the alphabet used in the DNA sequences, and the required scoring schemes. There is a full implementation for protein scoring matrices, and initial tests have been conducted. A local alignment scoring recurrence has also been implemented, and some work done towards the local alignment trace back. However, the search space reduction technique needs to be refined for local alignments. The current search space reduction technique is good for global alignment only, since all the sequences are stretched from start to end to insert gaps that will maximize the matches, and minimizes the mismatches and gap insertions. That's why the alignment path is expected to be a group of points across a band around the hyper-diagonal. Since local alignments match any motif in any position in one sequence, with any similar motif in any other position in the remaining sequences, therefore, the motifs are not expected to be along the hyper-diagonal only for local alignments, it could be anywhere in the hyper-cube space.

7.2.7 Variable Partition Size

The design of a variable partition size ensures that all processors are busy as soon as computing starts. The partitioning should start with a very fine granularity, by having a small partition size. This guarantees that high parallelism is achieved in the early waves where only a few independent partitions are available for simultaneous processing in case of a large partition size. On the other hand, high communication cost is encountered by decreasing the partition size. Therefore, the granularity level should increase towards the middle waves of computation to the highest capacity of the cache memory of the processing elements. Then after the middle wave, the partition size needs to be decreased again to allow for more independent partitions. A balance needs to be achieved between the communication cost

metric and the computation capacity of the processing elements. On a machine with a high communication cost, coarse grain parallelism is to be applied, and vice versa.

The ultimate objective is to minimize the total parallel execution time as calculated in the equations presented in Section 5.4.2. Several optimization techniques can be used to find an optimal starting partition size, and the increase rate of the partition size that can be symmetrically used to decrease the partition size after the middle wave. Further algebraic and geometric analysis is required to define the equations for partitioning and wave calculations for variable partition sizes. This change will create irregular MoA shapes that can be used for sparse k-d arrays formalization that can help in the local alignment problem, among other applications.

In the definition of middle points identifying the hyper-diagonal line, it is assumed that the sequences are of equal lengths. To fix this, the division of the wave length over the different dimensions can be adjusted to consider variable sequence lengths as weights for the assignment per dimension.

Another optimization could be to employ variable partitioning size (strides over the different dimensions) to overcome the different lengths of the sequences (different elements' values in the shape vector of the scoring tensor). This will optimize the search space reduction technique presented in chapter 5, and will make the selection of the middle partitions around the middle partition in each wave, to be of sizes comparable to the sequence lengths.

7.2.8 Other High Dimensional Problems

Further problems can be addressed based on the provided solutions. The same modelling techniques can be used for data classification and clustering algorithms. A possible research direction can be towards finding a structure in data points read from a database and modelled as tensor points using the models presented in this thesis. One example of high dimensional data mining technique is building disease profiles based on records kept for diagnosed patients. The classes of features (hereditary features, lab test results, symptoms, life style patterns ... etc) of patients suffering from specific diseases are collected in a database. Then tensors are constructed from the feature spaces of the aggregated values of patients who suffer from the same disease. These tensors define the diseases' profiles. The diseases' profiles can be used against a new patient profile feature readings as it changes over time.

This allows future prediction for possible development of the feature values that can move a patient into specific disease profiles. Accordingly a disease management plan can be built to reduce specific feature values to move away from the diseases profiles.

Bibliography

AARTS, E, Korst, J. 1989. *Simulated Annealing and Boltzmann Machines – A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Wiley Interscience Series in Discrete Mathematics and Optimization.

ABDESSLEM, L, Soham, M, Mohamed, B. 2006. Multiple Sequence Alignment by Quantum Genetic Algorithm. *In*: *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*.

AC3. *The Australian Centre for Advanced Computing and Communications (ac3) Academic and eResearch web site*. [online]. [Accessed 15 Aug 2009]. Available from World Wide Web: <http://www.ac3.edu.au/HPC/Hardware/Barossa>

ACAR E, Yener B. 2009. Unsupervised Multiway Data Analysis: A Literature Survey. *IEEE Transactions on Knowledge and Data Engineering*. **21**(1).

ALMASI, G et al. 2002. Cellular Supercomputing with System-on-a-Chip. *In*: *IEEE International Solid-State Circuits Conference*. Digest of Technical Papers.

ALTHAUS E, Caprara A, Lenhof HP, Reinert K. 2002. Multiple Sequence Alignment with Arbitrary Gap Costs: Computing an Optimal Solution using Polyhedral Combinatorics. *Bioinformatics*. **18**(Suppl. S.), p.S4–S16.

ALTSCHUL, SF, Gish, W, Miller, W, Myers, EW, Lipman, DJ. 1990. Basic Local Alignment Search Tool. *Journal of Molecular Biology*. **215**(3), p.403–10.

ALTSCHUL, SF. 1991. Amino Acid Substitutions Matrices from an Information Theoretic Perspective. *Molecular Biology*. **219**(3), pp.555-565.

ALVAREZ, SA. 1997. *Note on an N-Dimensional Pythagorean Theorem*. Pittsburgh, PA, USA: Center for Nonlinear Analysis and Department of Mathematical Sciences, Carnegie Mellon University.

AMDAHL, G. 1967. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. *In*: *AFIPS '67 (Spring Joint Computer Conference) Proceedings*. ACM, p.483–485.

ATTESON, K. 1997. The Performance of Neighbor-Joining Algorithms of Phylogeny Reconstruction. *In*: T, Lee, D JIANG, (ed). *Lecture Notes in Computer Science*, Berlin: Springer-Verlag, p.101–110.

AWODEY, S. 2006. *Category Theory*. Oxford University Press.

BADER, BW, KOLDA, TG. 2007. Efficient Matlab Computations with Sparse and Factored Tensors. *SIAM Journal on Scientific Computing*. **30**(1), p.205–231.

BELL, ET. 1986. Invariant Twins: Sylvester, Cayley. *In*: *Men of Mathematics: The Lives and Achievements of the Great Mathematicians from Zeno to Poincaré*, New York: Simon and Schuster, pp.378-405.

BELLMAN, RE. 1961. *Adaptive Control Processes*. Princeton, NJ, USA: Princeton University Press.

BERG, MD. 2008. *Computational Geometry : Algorithms and Applications*. Berlin: Springer.

BLANCHETT, e M, Kent, WJ, Riemer, C, et al. 2004. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*. **14**, p.708–15.

BLANCHETTE, M, Kent, WJ, Riemer, C, Elnitski, L, Smit, AFA, Roskin, KM, Baertsch, R, Rosenbloom, K, Clawson, H, Green, ED, Haussler, D, Miller, W. 2004. Aligning Multiple Genomic Sequences With the Threaded Blockset Aligner. *Genome Research*. **14**(4), pp.708-715.

CARROLL, JD, Chang, J. 1970. Analysis of Individual Differences in Multidimensional Scaling via an N-Way Generalization of "Eckart-Young" Decomposition. *Psychometrika*. **35**(3), pp.283-319.

CHEN, C, Schmidt, B. 2005. An Adaptive Grid Implementation of DNA Sequence Alignment. *Future Generation Computer Systems Archive*. **21**(7), pp.988-1003.

CHEN, C, Schmidt, B (2). 2005. Parallel Construction of Large Suffix Trees on a PC Cluster. *In*: *Euro-Par 2005*. Lisbon, Portugal: Springer-Verlag LNCS.

CHINGCHIT, S. 1999. *Design and Performance Evaluation of a Flexible Clustering and Allocation Scheme for Parallel Processing*. Perth, Australia: A Dissertation submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at, Curtin University of Technology.

COMON, P. 1994. Independent Component Analysis: A New Concept. *Signal Processing*. **36**(3), pp.287-314.

DARLING, AC, Mau, B, Blattner, FR, et al. 2004. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome Research*. **14**, p.1394–403.

DEPIEREUX, E, Baudoux, G, Briffeuil, P, Reginster, I, De Bolle, X, Vinals, C, Feytmans, E. 1997. Match-Box_server: A Multiple Sequence Alignment Tool Placing Emphasis on Reliability. *Bioinformatics*. **13**(3), pp.249-256.

DEUPREE, JD, Scofield, MA, Bylund, DB. 2000. Analyses of Adrenergic Receptor Sequences. *In*: CA MACHIDA, (ed). *Adrenergic Receptor Protocols*, Totowa, New Jersey: Humana Press, pp.53-72.

DO, CB, Katoh, K. 2000. Protein Multiple Sequence Alignment. *In*: JD, et al. THOMPSON, (ed). *Methods in Molecular Biology*, Totowa, NJ: Humana Press, pp.379-413.

DO, CB, Mahabhashyam, MSP, Brudno, M, Batzoglou, S. 2005. PROBCONS: Probabilistic Consistency-Based Multiple Sequence Alignment. *Genome Research*. **15**(2), pp.330-340.

DRIGA, AR. 2002. *Parallel FASTLSA: A Parallel Algorithm for Parallel Sequence Alignment*. Edmonton, Alberta, Canada: A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science, Department of Computing Science, University of Alberta.

DURET, L, Abdeddaim, S. 2000. Multiple alignment for structural functional or phylogenetic analyses of homologous sequences. *In*: D, Taylor, W HIGGINS, (ed). *Bioinformatics sequence structure and databanks*, Oxford: Oxford University Press.

EDDY, SR. 1995. Multiple Alignment Using Hidden Markov Models. *In*: *3rd. ISMB*., p.114 – 120.

EDDY, SR. *HMMER: A Profile Hidden Markov Modeling Package*. [online]. [Accessed August 2009]. Available from World Wide Web: <http://hmmer.janelia.org/>

EDGAR, RC. 2004. MUSCLE: Multiple Sequence Alignment with High Accuracy and High Throughput. *Nucleic Acids Research*. **32**(5), pp.1792-1797.

EDGEWORTH, FY, Fisher, RA. 1976. On the Efficiency of Maximum Likelihood Estimation. *The Annals of Statistics*. **4**(3), p.501–514.

ESSOUSSI, N, Boujenfa, K, Limam, M. 2008. A Comparison of MSA Tools. *Bioinformation*. **2**(10), p.452–455.

FALKOFF, AD, Iverson, KE, Sussenguth, EH. 1964. A Formal Description of System/360. *IBM Systems Journal*. **I**(3), pp.198-263.

FELSENSTEIN, J. 1978. Cases in which Parsimony and Compatibility Methods will be Positively Misleading. *Zoological Systematics and Evolutionary Research*. **27**(4), pp.401-410.

FELSENSTEIN, J. 1989. PHYLIP -- Phylogeny Inference Package (Version 3.2). *Cladistics*. **5**, pp.164-166.

FELSENSTEIN, J. 2004. *Inferring Phylogenies*. Sunderland, MA: Sinauer Associates.

FENG, DF, Doolittle, RF. 1987. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Molecular Evolution*. **25**(4), pp.351-360.

FIELDING, AH. 2007. *Cluster and Classification Techniques for the Biosciences*. Cambridge: Cambridge University Press.

FODOR, IK. 2002. *A Survey of Dimension Reduction Techniques*. CA, USA: Lawrence Livermore National Lab, US Department of Energy (US).

FRIEDMAN, JH, Tukey, JW. 1974. A Projection Pursuit Algorithm for Exploratory Data Analysis. *IEEE Transactions on Computers C*. **23**(9), p.881–890.

GASCUEL, O, Steel, M. 2006. Neighbor-Joining Revealed. *Molecular Biology and Evolution*. **23**(11), pp.1997-2000.

GILDER, L. 2008. Heisenberg in Helgoland. *In*: *The age of entanglement: when quantum physics was reborn*, Knopf, p.74.

GOLUBCHIK, T, Wise, MJ, Easteal, S, Jermiin, LS. 2007. Mind the gaps: evidence of bias in estimates of multiple sequence alignments. *Molecular Biology and Evolution*. **24**(11), pp.2433-2442.

GOTOH, O. 1982. An Improved Algorithm for Matching Biological Sequences. *Molecular Biology*. **162**(3), pp.705-8.

GOTOH, O. 1996. Significant Improvement in Accuracy of Multiple Protein Sequence Alignments by Iterative Refinement as Assessed by Reference to Structural Alignments. *Molecular Biology*. **264**(4), pp.823-838.

GRAHAM, SL, Snir, M, Patterson, CA. 2004. *Getting Up to Speed: The Future of Supercomputing*. Washington, DC 20055, USA: National Research Council, Committee on the Future of Supercomputing, National Academies Press.

GRAVES, S. 1981. A Review of Production Scheduling. *Operations Research*. **29**(4), pp.646-675.

GRAY, F. 1947. *Pulse Code Communication*. 2632058.

GROPP, W, Lusk, E, Skjellum, A. 1999. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press.

GROPP, W, Lusk, E, Thakur, R. 1999. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press.

GUPTA, SK, Kececioglu, JD, Schaffer, AA. 1995. Improving the Practical Space and Time Efficiency of the Shortest-Paths Approach to Sum-of-Pairs Multiple Sequence Alignments. *Computational Biology*. **2**(3), p.459–472.

GUSFIELD, D. 1993. Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds. *Bulletin of Mathematical Biology*. **55**(1), p.141–154.

GUSFIELD, D. 1997. *Algorithms on Strings, Trees, and Sequences*. New York, NY, USA: Cambridge University Press.

HAINS, G, Mullin, LR. 1993. Parallel Functional Programming with Arrays. *The Computer Journal*. **36**(3), pp.238-245.

HARARY, F. 1994. *Graph Theory*. Addison-Wesley.

HARSHMAN, RA. 1970. *Foundations of the PARAFAC Procedure: Model and Conditions for an 'Explanatory' Multi-Mode Factor Analysis*. California: UCLA Working Papers in phonetics.

HEINIKOFF, S, Heinikoff, JG. 1992. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*. **89**, pp.10915-19.

HEINZ-DIETER, E. 2007. *Ernst Zermelo: An Approach to His Life and Work*. Springer-Verlag.

HELAL, M. 2001. *Mathematics of Arrays - The Implementation and the Application*. Cairo, Egypt: A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science, Computer Science Department, School of Science and Engineering, American University in Cairo.

HELAL, M, Mullin, LM, Gaeta, B, El-Gindy, H. 2007. Multiple Sequence Alignment using Massively Parallel Mathematics of Arrays. *In*: *In: Proceedings of the International Conference on High Performance Computing, Networking and Communication Systems (HPCNCS- 07)*. Orlando, FL. USA, pp.120-127.

HELAL, M, El-Gindy, H, Gaeta, G, Sinchenko, V. 2008. High Performance Multiple Sequence Alignment Algorithms for Comparison of Microbial Genomes. *In*: *19th International Conference on Genome Informatics - GIW 2008*. Gold Coast.

HELAL, M, El-Gindy, H, Mullin, LM, Gaeta, B. 2008. Parallelizing Optimal Multiple Sequence Alignment by Dynamic Programming. *In*: *Proceedings of the International Symposium on Advances in Parallel and Distributed Computing Techniques (APDCT-08) held in conjunction with 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA-08)*. Sydney, Australia: IEEE Computer Society, pp.669-674.

HELAL, M, Sintchenko, V. 2009. Dynamic Programming Algorithms for Discovery of Antibiotic Resistance in Microbial Genomes. *In*: *Health Informatics Conference (HIC-09)*. Canberra, Australia.

HELAL, M, Mullin, L, Potter, J, Sintchenko, V. 2009. Search Space Reduction Technique for Distributed Multiple Sequence Alignment. *In*: *Sixth IFIP International Conference on Network and Parallel Computing (NPC 2009)*. Gold Coast, Queensland, Australia.

HOGEWEG, P, Hesper, B. 1984. Energy Directed Folding of RNA Sequences. *Nucleic Acids Research*. **12**(1), pp.67-74.

HOGEWEG, P, Hesper, B. 1984. The Alignment of Sets of Sequences and the Construction of Phyletic Trees: an Integrated Method. *Molecular Evolution*. **20**(2), pp.175-186.

HUGHEY, R, Krogh, A. 1995. *SAM : Sequence Alignment and Modeling Software System*. CA.

HUNT, HB, Mullin, LR, Rosenkrantz, DJ, Raynolds, JE. 2008. *A Transformation--Based Approach for the Design of Parallel/Distributed Scientific Software: the FFT*.

IYENGAR, AK. 1989. *Parallel Characteristics of Sequence Alignment Algorithms*. Cambridge, MA 02139, USA: Laboratory for computer Science, Massachusetts Institute of Technology.

JENKINS, MA, Mullin, LR. 1991. A Comparison of Array Theory and a Mathematics of Arrays, in "Arrays, Functional Languages and Parallelism". *In*: *Proceedings of the Montreal Workshop*. Kluwer Academic Publishers.

JENKINS, MA, Franksen, OI. 1992. On Axis Restructuring Operations for Nested Arrays. *In*: *2nd International Workshop on Array Structures ATABLE-92*. Montreal.

JOLLIFFE, IT. 2002. Principal Component Analysis. *Springer Series in Statistics 2nd ed*, p.28.

JONES, A. 1998. *Survey of Job Shop Scheduling Techniques*. California, USA: National Institute of Standards and Technology, Luis C. Rabelo, Industrial and Manufacturing Engineering Department, California Polytechnic State University.

JUST, W. 2001. Computational Complexity of Multiple Sequence Alignment with SP-Score. *Computational Biology*. **8**(6), pp.615-623.

KAZUTAKA, K, Hiroyuki, T. BMC Bioinformatics. Improved accuracy of multiple ncRNA alignment by incorporating structural information into a MAFFT-based framework. *BMC Bioinformatics*. **9**(212).

KECECIOGLU, J. 1993. The Maximum Weight Trace Problem in Multiple Sequence Alignment. *In*: *Proceedings of the 4th Symposium on Combinatorial Pattern Matching*. Padova, Italy: Springer-Verlag Lecture Notes in Computer Science, pp.106-119.

KIM, J, Pramanik, S, Chung, MJ. 1994. Multiple Sequence Alignment using Simulated Annealing. *Computer Applications in Bioscience*. **10**(4), pp.419-426.

KOLACZKOWSKI, B, Thornton, JW. 2004. Performance of Maximum Parsimony and Likelihood Phylogenetics when Evolution is Heterogeneous. *Nature*. **431**(7011), pp.980-984.

KOLDA, TG, Bader, BW. 2009. Tensor Decompositions and Applications. *SIAM Annual Review*. **51**(3).

KUMUR, V, Grama, Y, Nageshwara, V. 1993. *Scalable Load Balancing Techniques for Parallel Computers*. Florida, USA: Department of Computer Science, University of Minnesota, Minneapolis, Department of Computer Science, University of Central Florida.

LABORATORY, Los Alamos National web site. *Shannon Entropy Readme File*. [online]. [Accessed 13 August 2009]. Available from World Wide Web: <://www.hiv.lanl.gov/content/sequence/ENTROPY/entropy_readme.html>

LAMPING, J, Rao, R. 1994. Laying out and Visualizing Large Trees Using a Hyperbolic Space. *In*: *Proceedings of the ACM Symposium on User Interface Software and Technology, ACM Press*. CA, USA: Xerox Palo Alto Research Center, pp.13-14.

LANDRY, W. 2003. Implementing a High Performance Tensor Library. *Scientific Programming*. **11**(4), p.273–290.

LASSMANN, T, Sonnhammer, EL. 2006. Kalign, Kalignvu and Mumsa: Web Servers for Multiple Sequence Alignment. *Nucleic Acids Research*. **34**(Web Server issue), pp.W596-W599.

LEGENDRE, P, Legendre, L. 1998. *Numerical Ecology*. Elsevier.

LERMEN, M, Reinert, K. 2000. The Practical Use of the A* Algorithm for Exact Multiple Sequence Alignment. *Computational Biology*. **7**(5), p.655–671.

LIPMAN, DJ, Pearson, WR. 1985. Rapid and sensitive protein similarity searches. *Science*. **227**(4693), p.1435–41.

LIPMAN, D, Carrillo, H. 1988. The Multiple Sequence Alignment in Biology. *SIAM Journal of Applied Mathematics*. **48**(5), pp.1073-1082.

LIPMAN, DJ, Altschul, SF, Kececioglu, JD. 1989. A Tool for Multiple Sequence Alignment. *Proceedings of the National Academy of Sciences*. **86**(12), pp.4412-4415.

MCCLURE, MA, Vasi, TK, Fitch, WM. 1994. Molecular Biology and Evolution. *Comparative Analysis of Multiple Protein-Sequence Alignment Methods*. **11**(4), pp.571-592.

MENDELSON, E. 1997. Classic Textbook Treatment of NBG. *In*: *An Introduction to Mathematical Logic*, London: Chapman & Hall, p.225–86.

MEUER, H, Strohmaier, E, Dongarra, J, Simon, H. *Top 500*. [online]. [Accessed 31 Mar 2009]. Available from World Wide Web: <http://www.top500.org/>

MILLIGAN, GW. 1979. Ultrametric Hierarchical Clustering Algorithms. *Psychometrika*. **44**(3), pp.343-346.

MOORE, GE. 1965. *Cramming more Components onto Integrated Circuits*. Electronics Magazine.

MORE, T. 1973. *Notes on the Axioms for a Theory of Arrays*. Philadelphia, Pa.: IBM Scientific Ctr.

MORGENSTERN, B, Dress, A, Wener, T. 1996. Multiple DNA and Protein Sequence based on Segment-to-Segment Comparison. *Proceedings of the National Academy of Sciences*. **93**(22), pp.12098-12103.

MOUNT, DW. 2004. *Bioinformatics: Sequence and Genome Analysis*. NY, USA: CSHL Press, Cold Spring Harbor.

MULLIN, LR. 1988. *A Mathematics of Arrays*. Syracuse, NJ, USA: Doctor of Philosophy Dissertation in Computer and Information Science Completed at Syracuse University.

MULLIN, LR. 1992. *The Psi Correspondence Theorem: Array Mapping Using the Psi Calculus*.

MULLIN, LR, Small, S. 2002. Four Easy Ways to a Faster FFT. *Mathematical Modeling and Algorithms*. **1**(3), pp.193-214.

MULLIN, LR, Luo, X, Bush, L. 2003. Building the Support for Radar Processing across Memory Hierarchies: On the Development of an Array Class with Shapes using Expression Templates in C++. *In*: *High Performance Embedded Computing (HPEC) Workshop 2003*. Lexington, MA: MIT Lincoln Laboratory.

MULLIN, LR. 2005. A Uniform Way of Reasoning about Array-Based Computation in Radar. *Digital Signal Processing*. **15**(5), pp.466-520.

MULLIN, LR, Raynolds, J. 2005. Applications of Conformal Computing Techniques to Problems in Computational Physics: the FFT. *Computer Physics Communications*. **170**, pp.1-10.

MULLIN, LR, Raynolds, JE. 2008. *Conformal Computing: Algebraically Connecting the Hardware/Software Boundary using a Uniform Approach to High-Performance Computation for Software and Hardware Applications*. Springer.

NEEDLEMAN, SB, Wunsch, CD. 1970. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Molecular Biology*. **48**(3), pp.443-53.

NOTREDAME, C, Higgins, DG. 1996. SAGA: Sequence Alignment by Genetic Algorithm. *Nucleic Acids Research*. **24**(8), p.1515–1524.

NOTREDAME, C, O'Brien, EA, Higgins, DG. 1997. RAGA: RNA Sequence Alignment by Genetic Algorithm. *Nucleic Acids Research*. **25**(22), pp.4570-4580.

NOTREDAME, C, Higgins, DG, Heringa, J. 2000. T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment. *Molecular Biology*. **302**(1), pp.205-217.

NOTREDAME, C. 2002. Recent Progresses in MSA: a Survey. *Pharmacogenomic*. **3**(1), p.1–14.

PEI, J, Grishin, NV. 2006. MUMMALS: Multiple Sequence Alignment Improved by using Hidden Markov Models with Local Structural Information. *Nucleic Acids Research*. **34**(16), p.4364–4374.

PERREY, SW, Stoye, J, Moulton, V, Dress, AWM. 1997. *On Simultaneous Versus Iterative Multiple Sequence Alignment*. Bielefeld, Germany: Forschungsschwerpunkt Mathematisierung, University of Bielefeld.

PEVSNER, J. 2003. *Bioinformatics and Functional Genomics*. New York, USA: John Wiley.

PHUONG, TM, Do, CB, Edgar, RC, et al. 2006. Multiple alignment of protein sequences with repeats and rearrangements. *Nucleic Acids Research*. **34**, p.5932–42.

PRAKASH, SR. 1998. *Hyperplane Partitioning: An Approach to Global Data Partitioning for Distributed Memory Machines*. Bangalore.

RAO, JKM. 1987. New Scoring Matrix for Amino Acid Residue Exchanges Based on Residue Characteristic Physical Parameters. *International Journal of Peptide and Protein Research*. **29**(2), pp.276-281.

RAPHAEL, B, Zhi, D, Tang, H, et al. 2004. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*. **14**, p.2336–46.

REINERT, K, Lenhof, HP, Mutzel, P, Mehlhorn, K, Kececioglu, JD. 1997. A Branch-and-Cut Algorithm for Multiple Sequence Alignment. *In*: *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB-97)*. Santa Fe, New Mexico: ACM Press, p.241–249.

REINERT, K, Stoye, J, Will, T. 2000. An Iterative Method for Faster Sum-of-Pairs Multiple Sequence Alignment. *Bioinformatics*. **16**(9), p.808–814.

REINGOLD, EM, Nievergelt, J, Deo, N. 1997. *Combinatorial Algorithms, Theory and Practice*. New Jersey: Prentace Hall.

RICE, P, Longden, I, Bleasby,A. 2000. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics*. **16**(6), pp.276-277.

RICE, P, Longden, I, Bleasby, A. 2000. *EMBOSS: The European Molecular Biology Open Software Suite*. [online].

RUDIN, W. 1976. Chapter 10: Simple Review of Topological Properties. *In*: *Principles of Mathematical Analysis*, New York: McGraw-Hill.

SCHWARTZ, RM, Dayhoff, MO. 1978. Matrices for Detecting Distant Relationships. *Atlas of Protein Sequence and Structure*. **5**(3), pp.353-358.

SLOUGHTER, D. 2001. *The Calculus of Functions of Several Variables*. Furman University.

SMITH, TF, Waterman, MS. 1981. Identification of Common Molecular Subsequences. *Molecular Biology*. **147**(1), p.195–197.

SMITH, RF, Smith, TF. 1992. Pattern-Induced Multi-Sequence Alignment (PIMA) Algorithm Employing Secondary Structure-Dependent Gap Penalties for Comparitive Protein Modelling. *Protein Engineering*. **5**(1), pp.35-41.

SNEATH, PHA, Sokal, RR. 1973. Numerical Taxonomy. *Freeman*.

STONE, HS. 1977. Multiprocessor Scheduling with the Aid of Network Flow Algorithms. *IEEE Transactions on Software Engineering*. **3**(1), pp.85-93.

STOYE, J, Moulton, V, Dress, AWM. 1997. DCA: an Efficient Implementation of the Divide-and-Conquer Approach to Simultaneous Multiple Sequence Alignment. *Bioinformatics*. **13**(6), p.625–626.

SUBRAM, AR, Kaufmann, M, Morgenstern, B. 2008. DIALIGN-TX: greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms for Molecular Biology*. **3**(6).

TAHERI, J, Zomaya, AY, Zhou, BB. 2008. *RBT-I: A Novel Approach for Solving the Multiple Sequence Alignment*. Sydney, Australia.

TAYLOR, WR. 1987. Multiple Sequence Alignment by a Pairwise Algorithm. *Computer Applications in the Biosciences*. **3**(2), p.81–87.

TAYLOR, WR. 1988. A Flexible Method to Align Large Numbers of Biological Sequences. *Molecular Evolution*. **28**(1-2), pp.161-169.

THOMPSON, JD, Higgins, DG, Gibson, TJ. 1994. CLUSTAL W, Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position Specific Gap Penalties and Weight Matrix Choice. *Nucleic Acids Research*. **22**(22), pp.4673 - 4680.

THOMPSON, JD, Gibson, TJ, Plewniak, F, Jeanmougin, F, Higgins DG. 1997. The CLUSTAL_X Windows Interface: Flexible Strategies for Multiple Sequence Alignment Aided by Quality Analysis Tools. *Nucleic Acids Research*. **25**(24), pp.4876 - 4882.

THOMPSON, JD, Plewniak, F, Poch, O. 1999. A Comprehensive Comparison of Multiple Sequence Alignment Programs. *Nucleic Acids Research*. **27**(13), pp.2682-90.

THOMPSON, JD, Plewniak, F, Poch, O. 1999. BAliBASE: A Benchmark Alignment Database for the Evaluation of Multiple Alignment Programs. *Bioinformatics*. **15**(1), pp.87-88.

TILES, M. 2004. *The Philosophy of Set Theory: An Historical Introduction to Cantor's Paradise*. Dover Publications.

TOMPA, M. 2000. *Lecture Notes on Biological Sequence Analysis*. Washington, USA: NSF and DARPA grant DBI-9601046 and NSF grant DBI-9974498.

TUCKER, LR. 1966. Some Mathematical Notes on Three-Mode Factor Analysis. *Psychometrika*. **31**(3), p.279–311.

TUCKER, L, MacCallum, R. 1997. *Exploratory Factor Analysis*. Retrieved August 15, 2009, from: http://www.unc.edu/~rcm/book/factornew.htm.

VAN LOAN, CF. 2009. *Future Directions in Tensor-Based Computation and Modeling*. Arlington,Virginia, USA.

WANG, L, Jiang, T. 1994. On the Complexity of Multiple Sequence Alignment. *Computational Biology*. **1**(4), p.337–348.

WANG, X. 2006. On the Effects of Dimension Reduction Techniques on Some High-Dimensional Problems in Finance. *OPERATIONS RESEARCH*. **54**(6), p.1063–78.

WATSON, JD, Crick, FHC. 1953. A Structure for Deoxyribose. *Nucleic Acid. Nature*. **171**(4356), pp.737 - 738.

WONG, KM, Suchard, MA, Huelsenbeck, JP. 2008. Alignment Uncertainty and Genomic Analysis. *Science*. **319**(5862), pp.473-6.

XIA, X. 2002. Pair-wise and Multiple Sequence Alignment. *In*: *Data Analysis in Molecular Biology and Evolution*, Kluwer Academic Publishers, pp.33-39.

YAP, TK. 1995. *Parallel Computation in Biological Sequence Analysis*. Virginia, USA: A Dissertation submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at,Department of Computer Science, School of Information Technology and Engineering, George Mason University.

YAP, TK, Munson, PJ, Frieder, O, Martino, RL. 1995. *Parallel Multiple Sequence Alignment Using Speculative Computation*. Virginia, USA: Division of Computer Research and Technology, National Institutes of Health, Department of Computer Science, George Mason University.

ZOMAYA, Albert. 2006. *Parallel Computing for Bioinformatics and Computational Biology*. Wiley.

Appendix A: Glossary

Abbreviation	**Definition**
ALS	Alternating Least Squares
ANSI	American National Standards Institute
API	Application Program Interface
ASIC	Application-Specific Integrated Circuit
BLAST	Basic Local Alignment Search Tool
Bp	Base pair or Nucleotide or residue
candecomp	canonical decomposition
Cholesky	Cholesky decomposition
CPUs	Central Processing Unit
DCA	Divide and Conquer MSA
DMM	Distributed Memory Multicomputer
DNA	Deoxyribonucleic Acid
DP	Dynamic Programming
DRAM	Dynamic Random Access Memory
EBI	European Bioinformatics Institute is a centre for research and services in bioinformatics, and is part of EMBL
EMBL	European Molecular Biology Laboratory
EMBOSS	European Molecular Biology Open Software Suite
EOF	Empirical Orthogonal Function
FASTA	"FAST-All" is a DNA and protein sequence alignment software package and sequence file format.
FFT	Fast Fourier Transform
FIFO	First In First Out
FPGAs	Field-Programmable Gate Array is a semiconductor device that can be configured by the customer or designer after manufacturing.
GB/GByte	Giga Byte = 1024 x 1 MB. It is a measuring unit for memory capacity in computers.
GenBank	It is a sequence database allowing open access for annotated collection

	of all publicly available nucleotide sequences and their protein translations, produced at NCBI as part of the INSDC.
GFlops	Giga FLoating point Operations Per Second. It is a unit of measure used to measure the speed of a processor.
GHz	Giga Hertz is a unit of frequency defines the number of completed cycles per second, used to measure the clock speed for computer's Central Processing Unit (CPU).
GPUs	A Graphics Processing Unit (also occasionally called visual processing unit or VPU) is a specialized processor that offloads 3D graphics rendering from the microprocessor.
Gray Code	The reflected binary code, named after Frank Gray, is a binary numeral system where two successive values differ in only one bit.
Heuristic	It is an adjective for experience-based techniques that help in problem solving, learning and discovery. A heuristic method is particularly used to rapidly come to a solution that is hoped to be close to the best possible answer, or 'optimal solution'. Heuristics are "rules of thumb", educated guesses, intuitive judgments or simply common sense.
HMM	Hidden Markov Models.
HPC	High Performance Computers.
ICA	Independent Component Analysis.
INSDC	International Nucleotide Sequence Database Collaboration.
KB/Kbyte	Kilo Byte = 1024 Byte.
L1 Cache	Level 1 Cache Memory.
L2 Cache	Level 2 Cache Memory.
LAM	Local Area Multicomputer. LAM/MPI is one of the predecessors of the Open MPI project.
LU Decomp	Lowest/Upper decomposition.
MB/MByte	Mega Byte = 1024 x 1 KB. It is a measuring unit for memory capacity in computers.
MBps	Mega Bit Per second
MDS	Multi-Dimensional Scaling
MFC	Microsoft Foundation Class Library

MIMD	Multi-Instruction Multi-processor
ML	Maximum Likelihood
mmDst	MSA using MoA (Distributed)
MoA	Mathematics of Arrays
MPI	Message Passing Interfaces
MPP	Massively Parallel Processing
MSA	Multiple Sequence Alignment
MUSCLE	MUltiple Sequence Comparison by Log-Expectation
MWT	Maximum Weight Trace
NCBI	National Centre for Biotechnology Information
NN	Neural Networks
NP	Class of computational problems that are solved in Nondeterministic Polynomial time.
NP-complete	Class of NP problems that if solved in polynomial time, then all other problems in its class (NP) can.
NP-Hard	Class of NP problems that is at least as hard as the hardest problems in NP.
Nuc	Nucleotide or base pair or residue
OpenMP	Open Multi-Processing is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming.
OTU	Operational Taxonomic Units
P2P	Peer-to-Peer is a distributed computing architecture. In this thesis it is used to refer to dynamic scheduling and communication among processor where no master process is required.
PAM	Percent Accepted Mutation. One PAM of mutation describes an amount of evolution which changes, on the average, 1% of the amino acids
PARAFAC	Parallel Factor Analysis
PCA	Principal Component Analysis
PDEs	Partial Differential Equations
PIR	Protein Information Resource located at Georgetown University Medical Centre (GUMC), is an integrated public bioinformatics

	resource to support genomic and proteomic research.
PP	Projection Pursuit. It is a type of statistical technique which involves finding the most "interesting" possible projections in multidimensional data
PVM	Parallel Virtual Machine. It is a software tool for parallel networking of computers. It is designed to allow a network of heterogeneous Unix and/or Windows machines to be used as a single distributed parallel processor.
QC	Quantum Computing. It is a device for computation that makes direct use of quantum mechanical phenomena, such as superposition and entanglement, to perform operations on data.
RAM	Random Access Memory. It takes the form of integrated circuits that allow stored data to be accessed in any order.
RASCs	Reconfigurable Application-Specific Computers. It is a specialized reconfigurable computer for high-performance computing
SAGA	Sequence Alignment Using Genetic Algorithm is one of the MSA methods available in the literature.
SAM	Simulated Annealing MSA is one of the MSA methods available in the literature.
SIMD	Single-Instruction Multi Processor, known as vector instructions, for data level parallelism.
SMP	shared-memory multiple- processor
SPT	Shortest Path Tree. In graph theory, it is a sub-graph of a given (possibly weighted) graph constructed so that the distance between a selected root node and all other nodes is minimal.
STime	System Time in seconds - Also known as or very close to CPU Time
SVDs	Singular Value Decompositions
SVM	Support Vector Machines
SwissProt	It is a manually curated biological database of protein sequences maintained by the Swiss Institute of Bioinformatics.
TB/ TBytes	Tera Bytes = 1024 x 1 GB. It is a measuring unit for memory capacity in computers.
T-Coffee	Tree-based Consistency Objective Function For alignment

	Evaluation. It is a MSA software using a progressive approach.
TCP/IP	Transmission Control Protocol/Internet Protocol. It is the set of communications protocols used for the Internet and other similar networks.
Tflops	Tera FLoating point Operations Per Second. It is a unit of measure used to measure the speed of a processor.
UPGMA	Unweighted Pair Group Method with Arithmetic Mean.
UTime	User Time in seconds - Also known or very close to Elapsed time, or wall clock time.

Appendix B: HPC Development History

Name	Year	PEs	Memory per PE	Manufacturer
Cray-1 250 MFLOPS Energy Research and Development Administration (ERDA)	1976	4:8 I/O processors - 250 MFLOPS, 80 MHz	8 to 32M-words	Los Alamos National Laboratory, New Mexico,
Distributed Array processor (DAP)	1979	64x64 2D	4096 bits	ICL
Goodyear MPP	1983	16,384 as plane, cylinder, daisy-chain, or	35 shift registers, 1024 RAM	Goodyear Aerospace
Ultracomputer	1980	*N* processors, *N* memories and an *N log N:* 2 prototypes, 8 PEs and 16PEs were built.	512 KB : 1 MB	Courant Institute of Mathematical Sciences Computer Science Department
The Connection Machine: CM-1	1983	65,536 as hypercube	4 Kbits RAM	Danny's Hillis - MIT
The Connection Machine: CM-2	1987	4096 or 8192	: 512 MB RAM and 25 GB RAID Disk	Danny's Hillis - MIT
The Connection Machine: CM-5	1995	Fat Tree of sparc RISC, and later as cube of cubes		Danny's Hillis - MIT
Earth Simulator (was the fastest from 2002: 2004)	1997	640 nodes with 8 vector processors and; total 5120 processors; 35.86 TFLOPS	16 GByte memory / node; total 10 terabytes; 700 TBytes disk storage	NASDA, JAERI, and JAMSTIC
IBM Blue Gene/L	2004 : 2007	70.72 : 478.2 TFLOPS	512 MB to 2GB	IBM
IBM Blue Gene/P	2007	294,912: 884,736 - 1:3-PFLOPS	Up to 8 GB	IBM

IBM Roadrunner	2008	122,400(6,120 AMD64 Opteron (2 cores) @ 1.8 GHz + 12,240 PowerXCell 8i (9 cores) @ 3.2 GHz) - 1.7 : 1.105 PFLOPS	16 GB : 2.88 TiB	IBM