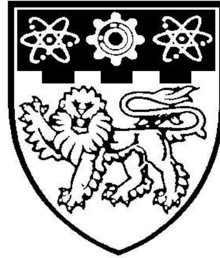
ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Nanyang Technological University



# Parallel and Distributed Algorithms for Computational Biology

by

**Weiguo Liu**

**Supervisor: Dr. Bertil Schmidt**

**Division of Information Systems**
**School of Computer Engineering**

**A thesis submitted to the Nanyang Technological University in fulfillment of the requirement for the degree of Doctor of Philosophy**

**October 2005**

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Statement of Originality

I hereby certify that the content of this thesis is the result of work done by myself and has not been submitted for a higher degree to any other University or Institution.

. . . . . . . . . . . . . . . . . . . .
Date

. . . . . . . . . . . . . . . . . . . . . . . . . . .
Signature

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

*To my wife and my dear parents,*
*who always give me love and support.*

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



# Abstract

The *computational biology* (CB) research area is now faced with an obstacle of ever-increasing genome data. The amount, complexity and increased need for the rigorous postprocessing of this data requires an increased role for *high performance computing* (HPC).

*Dynamic programming* (DP) is an important algorithm design technique in scientific computing. It has been widely applied to solve CB problems. Typical applications using this technique are compute-intensive and suffer from long runtimes on sequential architectures. In this thesis, we are proposing a *tunable coarse-grained partitioning and communication scheme* for DP algorithms. By introducing two performance-related parameters *division* and *rowwidth*, we can tradeoff between computation time and communication time by tuning these two parameters and thus obtain the maximum possible performance. We will demonstrate how this scheme leads to substantial performance gains for both *regular* and *irregular* DP applications.

*Genetic algorithms* (GAs) are efficient search methods based on Darwinian evolution. They have powerful characteristics such as robustness and flexibility to capture global solutions of complex optimization problems. The fundamental nature of GAs relies on heuristics to quickly search large numbers of candidate results in order to achieve better solutions over time. Unfortunately, the more likely a good solution can be found, the more computational resources are needed by GAs. This leads to high runtimes on sequential architectures. Because of their inherent parallelism, GAs are promising candidates for efficient and scalable parallel implementations. In this thesis, we present the design of a new *hierarchical parallel genetic algorithm* (HPGA) for the *protein folding problem* (PFP) on PC clusters and computational grids. Our hierarchical approach unites the inter-cluster and intra-cluster parallelism in an efficient way by using a combination of two communication models, i.e. the stepping stone model and the island model. This

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



unique combination achieves super-linear speedups on two different parallel architectures. Based on the concept of *hit rates* we also introduce a mathematical model to explain and predict our experimental results.

Although HPC can reduce the runtime of many compute-intensive applications significantly, the development and implementation of HPC programs are very complex. Therefore, this task is usually done by a small number of experts. In this thesis we propose a parallel pattern-based framework to facilitate the semi-automatic development of HPC programs. Parallel patterns are derived from sequential design patterns that are successfully used in object-oriented programming (OOP). By separating the communication structure of a parallel program from the sequential application, parallel patterns can be reused and therefore allow for a rapid development of HPC applications. We show how our parallel pattern-based framework can be deployed to implement parallel DP algorithms and HPGAs effectively and efficiently.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



# Acknowledgement

I would like to express my deepest gratitude and respect to the following people. Without their help, I could not have done so far.

I would like to thank the following people for working with me on the researches discussed in the thesis. I benefit a lot from their kindly helps. First and foremost, I would specially thank my supervisor, Dr. Bertil Schmidt, for his remarkable guidance, heuristic advices and encouragement during my research work. In the research, Dr. Bertil is always encouraging me to explore new problems and face new challenges. His broad knowledge, insightful vision and pure-hearted personality help and encourage me greatly during the research procedure. I would also thank Dr. Lee Bu-Sung, Dr. Cai Wentong, and Dr. Stephen John Turner for their valuable ideas in grid meetings that have inspired me so much on my research work. And I would like to say thanks to Mrs. Irene Goh, Mr. Chen Chunxi, Mr. Zeng Yi, Mr. Tang Ming, Mr. Yuan Zijing, Ms. Wang Lihua, Ms. Wang Xiaoguang, Ms. Feng Yuhong and all others who have supported my research work.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



# Author's Publications

### Journal Papers

1. Weiguo Liu and Bertil Schmidt, Mapping of Hierarchical Parallel Genetic Algorithms for Protein Folding onto Computational Grids, *IEICE Transactions on Information and Systems*, E89-D(2): 589–596, February 2006.

2. Weiguo Liu and Bertil Schmidt, A Parallel Pattern-based System for High Performance Computational Biology, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17, No. 8, pp. 750–763, August 2006.

### Conference Papers

1. Weiguo Liu and Bertil Schmidt, Parallel Design Pattern for Computational Biology and Scientific Computing, *Proceedings of 2003 IEEE International Conference on Cluster Computing (Cluster 2003)*, IEEE Computer Society Press, Hong Kong, 2003, pp. 456–459. (acceptance rate: 29.3%)

2. Weiguo Liu and Bertil Schmidt, A Generic Parallel Pattern-based System for Bioinformatics, *Proceedings of EuroPar'04*, Pisa, Italy, Springer, LNCS 3149, 2004, pp. 989–996. (acceptance rate: 35%)

3. Weiguo Liu and Bertil Schmidt, A Case Study on Pattern-based Systems for High Performance Computational Biology, *Proceedings of* $1^{st}$ *International BioEngineering Conference (IBEC 2004)*, Singapore, 2004, pp. 205–208.

4. Weiguo Liu and Bertil Schmidt, A Tunable Coarse-Grained Parallel Algorithm for Irregular Dynamic Programming Applications, *Proceedings of* $11^{th}$ *Annual International Conference on High Performance Computing (HiPC 2004)*, Bangalore, India, Springer, LNCS 3296, 2004, pp. 91–100. (acceptance rate: 22%)

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



5. Weiguo Liu and Bertil Schmidt, A Case Study on Pattern-based Systems for High Performance Computational Biology, *Proceedings of* $19^{th}$ *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005)(HICOMB workshop)*, IEEE Computer Society Press, Denver, 2005. (acceptance rate: 31%)

6. Weiguo Liu and Bertil Schmidt, Mapping of Genetic Algorithms for Protein Folding onto Computational Grids, *Proceedings of IEEE TENCON'05*, IEEE Computer Society Press, Melbourne, Australia, 2005.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Contents

**Abstract** **i**

**Acknowledgement** **iii**

**Publication** **iv**

**Contents** **vi**

**List of Figures** **xi**

**List of Tables** **xvii**

**1 Introduction** **1**

1.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.1.1 Nucleic Acids . . . . . . . . . . . . . . . . . . . . . . . 1

1.1.2 Proteins . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.1.3 Genome Databases . . . . . . . . . . . . . . . . . . . . 3

1.2 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

1.3 Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



1.4 Contributions . . . . . 12

1.5 Synopsis of the Thesis . . . . . 14

**2 Literature Review** **16**

2.1 Introduction . . . . . 16

2.2 Algorithm Design Techniques . . . . . 16

2.2.1 Exhaustive Search . . . . . 17

2.2.2 Branch-and-Bound Algorithms . . . . . 17

2.2.3 Greedy Algorithms . . . . . 18

2.2.4 Dynamic Programming . . . . . 18

2.2.5 Divide-and-Conquer Algorithms . . . . . 19

2.2.6 Machine Learning . . . . . 19

2.2.7 Randomized and Heuristic Algorithms . . . . . 20

2.3 Popular Genome Analysis Tasks . . . . . 21

2.3.1 Sequence Alignment . . . . . 21

2.3.2 Protein Structure Prediction . . . . . 25

2.4 Parallel Architectures: A Brief Introduction . . . . . 28

2.4.1 Flynn's Taxonomy . . . . . 28

2.4.2 A Further Breakdown of MIMD . . . . . 32

2.4.3 Grids . . . . . 35

2.5 Parallel Program Design Environments . . . . . 42

2.6 Parallel Program Design Methods . . . . . 46

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



2.6.1 Explicit Parallel Programming . . . . . . . . . . . . . . . . . . . . . 47

2.6.2 Parallel Pattern Programming . . . . . . . . . . . . . . . . . . . . . 49

2.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

**3 Characteristic Analyses of Sequential Computational Biology Algorithms 56**

3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

3.2 Dynamic Programming Algorithms . . . . . . . . . . . . . . . . . . . . 57

3.2.1 Characteristic Analysis of Dynamic Programming Algorithms . . 59

3.2.2 Space-Saving Algorithm . . . . . . . . . . . . . . . . . . . . . . . 78

3.3 Genetic Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . 80

3.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86

**4 Design of Partitioning and Communication Schemes 87**

4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

4.2 Parallel Dynamic Programming Algorithms . . . . . . . . . . . . . . . . 89

4.2.1 Striped Partitioning . . . . . . . . . . . . . . . . . . . . . . . . 89

4.2.2 Block-based Partitioning . . . . . . . . . . . . . . . . . . . . . . 90

4.2.3 Tunable Coarse-grained Partitioning and Communication Scheme 91

4.3 Design of a Hierarchical Parallel Genetic Algorithm for Protein Folding on Computational Grids . . . . . . . . . . . . . . . . . . . . . . . . . . . 100

4.3.1 Protein Folding Problems with HP Lattice Models . . . . . . . . . 100

4.3.2 A Hierarchical Parallel Genetic Algorithm for Protein Folding Problems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 105

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



4.3.3 Communication Scheme on Computational Grids . . . . . . . . . 110

4.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115

**5 A Generic Parallel Pattern-based Framework for Computational Biology Algorithms 116**

5.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116

5.2 Design and Development of Our Framework . . . . . . . . . . . . . . . 117

5.2.1 Multi-Paradigm Design for High Performance Computing . . . . . 118

5.2.2 Our Parallel Pattern-based Framework: Overview . . . . . . . . . 119

5.2.3 Development of An Extensible and Reusable Framework . . . . . 122

5.3 Performance Evaluations . . . . . . . . . . . . . . . . . . . . . . . 129

5.3.1 Experimental Results for Parallel Dynamic Programming Algorithms on PC Clusters . . . . . . . . . . . . . . . . . . . . . . . 129

5.3.2 Experimental Results for Parallel Genetic Algorithms . . . . . . . 135

5.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 140

**6 Conclusions and Future Work 142**

6.1 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142

6.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 143

6.2.1 Using Our Framework to Solve Other Dynamic Programming Applications . . . . . . . . . . . . . . . . . . . . . . . . . . . . 144

6.2.2 Using Our Framework to Solve Tertiary Structure Prediction for Real Proteins . . . . . . . . . . . . . . . . . . . . . . . . . . 146

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



6.2.3 Extending Our Parallel Pattern-based Framework . . . . . . . . . . 148

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# List of Figures

1.1 An abstract illustration of a segment of a DNA or RNA molecule. It shows that the molecule consists of a backbone of sugars linked together by phosphates with an amine base side chain attached to each sugar. The two ends of the backbone are conventionally called the 5' end and the 3' end . . . 2

1.2 Comparison of the rate of growth the GenBank sequence (data from Table 1.1) with the rate of growth of the number of transistors in personal computer chips (Moore's law: data from Table 1.2). . . . . . . . . . . . . . 7

2.1 The Single Instruction, Single Data (SISD) architecture . . . . . . . . . . 29

2.2 The Single Instruction, Multiple Data (SIMD) architecture . . . . . . . . 30

2.3 The Multiple Instruction, Single Data (MISD) architecture . . . . . . . . 31

2.4 The Multiple Instruction, Multiple Data (MIMD) architecture . . . . . . 31

2.5 The Symmetric Multiprocessor (SMP) architecture . . . . . . . . . . . . 32

2.6 The distributed-memory architecture . . . . . . . . . . . . . . . . . . . . 33

2.7 The layered grid architecture . . . . . . . . . . . . . . . . . . . . . . . . 37

3.1 The computation and composition of sub-problem solutions to solve problem $f(x_8)$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



3.2 Example of a wavefront computation: (a) shift direction, (b) dependency relationship . . . 62

3.3 Example of the Smith-Waterman algorithm to compute the local alignment between two sequences ATCTCGTATGATG and GTCTATCAC. . . . . 64

3.4 The syntenic alignment is an ordered list of local alignments separated by difference blocks . . . 66

3.5 (a) Dependency relationship and wavefront shift direction of SW with the linear gap penalty (b) Dependency relationship and wavefront shift direction of SW with the affine gap penalty (c) Dependency relationship and computation shift direction of syntenic alignment algorithm . . . 67

3.6 An example $8 \times 8$ skyline matrix . . . 68

3.7 The wavefront computation of skyline matrix problem: (a) shift direction, (b) dependency relationship . . . 69

3.8 The wavefront computation of Nussinov algorithm: (a) shift direction; (b) dependency relationship . . . 71

3.9 An example of four states Pre-Fix HMM . . . 73

3.10 The wavefront computation of arbitrary-order Viterbi algorithm: (a) shift direction, (b) dependency relationship . . . 74

3.11 The principle of spliced alignments . . . 76

3.12 Pseudo-code for the *LastColumn* Algorithm . . . 79

3.13 Classification of search techniques . . . 80

3.14 Genetic operators for GAs. (a) Mutation exchanges one single bit; (b) Crossover exchanges a contiguous fragment of an individual. . . . 83

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



4.1 (a) Columnwise striping (b) Columnwise cyclic striping . . . . . . . . . . 89

4.2 (a) block-based distribution of an $8 \times 8$ matrix using 4 processors, (b) DP computation for 4 processors, 8 columns and a $2 \times 2$ block size. The complete $8 \times 8$ matrix can then be computed in 7 iteration steps . . . . . 90

4.3 (a) Example of an irregular dependency pattern; (b) Distribution of load computation density . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91

4.4 The tunable coarse-grained partitioning and communication scheme for (a)The triangular matrix computation, (b)The square matrix computation 92

4.5 The general parallel algorithm for the tunable coarse-grained partitioning and communication scheme. (a) For the triangular matrix computation, (b) For the square matrix computation . . . . . . . . . . . . . . . . . . . 93

4.6 Residues are distributed evenly through the whole matrix . . . . . . . . . 95

4.7 Residues are put to the forefront of the matrix . . . . . . . . . . . . . . . 95

4.8 Performance comparison for the skyline matrix problem using different methods to treat the residue. The performance is measured on two Intel Pentium IV Xeon 2.6GHz processors in a PC cluster. . . . . . . . . . . . . 96

4.9 Communication scheme using synchronous communication . . . . . . . . 97

4.10 Communication scheme using asynchronous communication . . . . . . . 98

4.11 The general parallel algorithm after moving residues to the forefront of the matrix and using the hybrid communication mode. (a) For the triangular matrix computation, (b) For the square matrix computation . . . . . . . 99

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



4.12 Folding of a protein from a linear chain of amino acids to a three-dimensional structure. The folding pathway involves amino acid interactions. Many different amino acid patterns are found in the same types of folds. Thus making structure prediction from amino acid sequence a difficult undertaking. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101

4.13 (a) The standard HP model on the square lattice. (b) The HP model with side chains on the square lattice. (c) The HP tangent spheres model with side chains. Black denotes a hydrophobic amino acid, white denotes a hydrophilic amino acid, and gray denotes a backbone element . . . . . 103

4.14 Illustration of the crossover procedure of the GA for HP lattice models. In this example the cut point is randomly chosen to be between residues 6 and 7. The first 6 residues of (A) are then joined with the last 6 residues of (B) to form the new conformation (C). The energy value of conformation (C) is $-4$, which is lower than the energies in conformations (A) ($-3$) and (B) ($-2$). Thus the new conformation is accepted. . . . . . . . . . . . . . 103

4.15 The Unger-Moult GA for protein structure prediction . . . . . . . . . . . 104

4.16 Mapping of a sequence to the conformation space. . . . . . . . . . . . . . 105

4.17 Global single-population master-slave GAs. . . . . . . . . . . . . . . . . . 106

4.18 The multiple-population GA for PFP . . . . . . . . . . . . . . . . . . . . . 107

4.19 Migration models for multiple population GAs: (a) The island model (b) the stepping stone model . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108

4.20 An HPGA with the master-slave structure for each subpopulation . . . . 109

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



4.21 (a) An HPGA with the stepping stone model at the higher level and the island model at the lower level, (b) an HPGA with the island model at the higher level and the stepping stone model at the lower level . . . . . . . . 110

4.22 The structure of the two-layer architecture . . . . . . . . . . . . . . . . . . 112

4.23 The communication detail of the two-layer architecture. . . . . . . . . . . 114

5.1 Summarization of all the activities: from the *algorithm space* to the *implementation space*. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117

5.2 The two parts of parallel algorithms for (a) Parallel DP algorithms, (b) HPGAs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120

5.3 (a) Mapping of parallel DP algorithms onto a cluster, (b) Mapping of HPGAs onto the computational grid environment . . . . . . . . . . . . . 121

5.4 Pure OOP design for the parallel wavefront pattern. . . . . . . . . . . . . 122

5.5 Using inheritance and overriding virtual functions to develop new applications in traditional OOP. . . . . . . . . . . . . . . . . . . . . . . . . . . 123

5.6 The connection between the sequential data type and the MPI data type. 124

5.7 The structure of class template `GenericPattern`. . . . . . . . . . . . . . . 125

5.8 The UML class diagram for `GenericPattern` . . . . . . . . . . . . . . . . 126

5.9 The UML class diagram of an extension of the `GenericPattern` to implement parallel DP programs . . . . . . . . . . . . . . . . . . . . . . . . . 127

5.10 The UML class diagram of an extension of the `GenericPattern` to implement HPGA programs . . . . . . . . . . . . . . . . . . . . . . . . . . . 128

5.11 Reusability of components in the framework for different policies . . . . . 129

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



5.12 Speedups on the Alpha cluster for several regular and irregular DP CB algorithms with corresponding matrix size. . . . . . . . . . . . . . . . . . 132

5.13 Speedups of irregular DP applications using different partitioning schemes for (a) the skyline matrix problem, (b) the Smith-Waterman algorithm with general gap penalties. . . . . . . . . . . . . . . . . . . . . . . . . . 134

5.14 Speedups of irregular DP applications using different partitioning schemes for (a) the Nussinov algorithm, (b) the MCOP problem. . . . . . . . . . 134

5.15 Speedups of irregular DP applications using different partitioning schemes for the arbitrary-order Viterbi algorithm. . . . . . . . . . . . . . . . . . . 135

5.16 Hit rates of the parallel GA on the Alpha cluster . . . . . . . . . . . . . 138

5.17 Speedups of the parallel GA with (a) the island model and (b) the stepping stone model on the Alpha cluster . . . . . . . . . . . . . . . . . . . . . . 138

5.18 Hit rates of the HPGA on the computational grid environment . . . . . . 139

5.19 Speedups of our HPGA with (a) the island model on both the grid level and the cluster level, (b) the island model on the grid level and the stepping stone model on the cluster level, (c) the stepping stone model on both the grid level and the cluster level, (d) the stepping stone model on the grid level and the island model on the cluster level . . . . . . . . . . . . . . . 140

6.1 The new framework can generate HPC applications automatically. . . . . 148

6.2 Framework of the cost module. . . . . . . . . . . . . . . . . . . . . . . . . 149

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# List of Tables

1.1 Growth rate of GenBank [1] . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.2 The growth of the number of transistors in personal computer processors [7] 8

1.3 Comparison of rates of increase of different data explosion curves . . . . 8

3.1 A classification for the popular DP algorithms in CB . . . . . . . . . . . 61

3.2 Popular GA applications in CB. . . . . . . . . . . . . . . . . . . . . . . . 86

5.1 Performance comparison between two frameworks(framework1: only using OOP techniques; framework2: using GP and OOP techniques) . . . . . . 130

5.2 Speedups on the Beowulf cluster for several DP algorithms with corresponding matrix size. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131

5.3 Speedup comparison using different *division* $d$ and *rowwidth* $r$ for the Nussinov algorithm. The matrix size is 5000×5000. The number of processor is 32 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 133

5.4 Speedup comparison using different *division* $d$ and *rowwidth* $r$ for the Smith-Waterman algorithm with the general gap penalty function. The matrix size is 5000×5000. The number of processor is 32 . . . . . . . . . 133

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Chapter 1

# Introduction

## 1.1 Overview

All living organisms have a similar molecular chemistry. The most important molecules involved in bio-chemistry are *nucleic acids* and *proteins*. Nucleic acids encode the information necessary to produce proteins, and are also responsible for passing this information to subsequent generations. The research work in molecular biology is mainly devoted to the study of the structure and function of nucleic acids and proteins.

### 1.1.1 Nucleic Acids

There are two different types of nucleic acids: DNA (deoxyribonucleic acid) and RNA (ribonucleic acid). DNA was discovered in 1869 while studying the chemistry of white blood cells. The two strands of a DNA molecule are tied together in a helical structure. This is the famous double helix structure discovered by James Watson and Francis Crick in 1953 [147]. The structure holds because each base in one strand bonds to a base in the other strand. Pairs are always formed between the bases **A** and **T**, and between **G** and **C**. These pairs are known as *Watson-Crick base pairs*, and their bases are referred to as *complementary bases*. Most often, base pairs (*bp*) are used as the unit of length for

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



the DNA molecules.

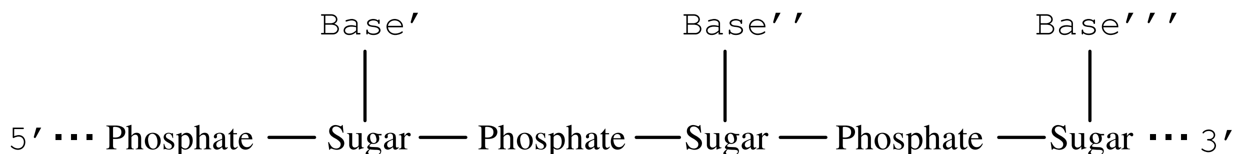



Figure 1.1: An abstract illustration of a segment of a DNA or RNA molecule. It shows that the molecule consists of a backbone of sugars linked together by phosphates with an amine base side chain attached to each sugar. The two ends of the backbone are conventionally called the 5' end and the 3' end

RNA molecules are similar to the DNA molecules, with small differences in composition and structure. DNA and RNA are chainlike molecules, called polymers, that consist of nucleotides linked together by phosphate ester bonds. A nucleotide consists of a phosphoric acid, a pentose sugar and an amine base. In DNA the pentose sugar is 2-deoxyribose and the amine base is either adenine, guanine, cytosine, or thymine. In RNA the pentose sugar is ribose instead of 2-deoxyribose and the amine base thymine is exchanged with the very similar amine base uracil. As illustrated in Figure 1.1 a DNA or RNA molecule is a uniform backbone of sugars linked together by the phosphates with side chains of amine bases attached to each sugar.

The most significant difference is that RNA does not form a double helix, although hybrid RNA-DNA helices are frequent. Also, parts of an RNA molecule may bind to other parts of the same molecule through complementarity. In terms of functionality, while DNA is used only for encoding information, there are multiple types of RNAs in a cell, performing different expression functions.

Both types of nucleic acids, DNA and RNA, are represented as strings of letters over the four-letter alphabet created by the letters corresponding to the nucleotides from which they are made.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



### 1.1.2 Proteins

Proteins are polymers that consists of amino acids linked together by peptide bonds. An amino acid consists of a central carbon atom, an amino group, a carboxyl group and a side chain. The side chain determines the type of the amino acid. A protein thus consists of a backbone of the common structure shared between all amino acids with the different side-chains attached to the central carbon atoms. Even though there is an infinite number of different types of amino acids, only twenty of these types are encountered in proteins. Similar to DNA and RNA molecules, it is thus possible to uniquely specify a protein by listing the sequence of side chains. Since there is only twenty possible side chains, the listing can be described as a string over a twenty letter alphabet.

Proteins constitute the majority of the substance types present in living organisms. For instance, *structural proteins* are the building blocks of tissues, while *enzymes* act as catalysts for biochemical reactions. Other proteins are used for the transport of oxygen, or act as antibodies for the immune system.

Only residues of the original amino acids are presented in the molecular chain. For this reason, the length of a protein is measured in residues rather than in amino acids, although often the latter term is used. Typical proteins are 300-residues long [130].

### 1.1.3 Genome Databases

In the past decade there has been an explosion in the amount of genome sequence data available, due to the very rapid progress of genome sequencing projects. There are three principal comprehensive databases of genome sequences in the world today.

- The EMBL (European Molecular Biology Laboratory) database is maintained at the European Bioinformatics Institute in Cambridge, UK [136].

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



- GenBank is maintained at the National Center for Biotechnology Information in Maryland, USA [30].
- The DDBJ (DNA Databank of Japan) is maintained at the National Institute of Genetics in Mishima, Japan [109].

These three databases share information and hence contain almost identical sets of sequences. The objective of these databases is to ensure that DNA sequence information is stored in a way that is publicly, and freely, accessible and that it can be retrieved and used by other researchers in the future. Most scientific journals require submission of newly sequenced DNA to one of the public databases before a publication can be made that relies on the sequence. This policy has proved tremendously successful for the progress of science, and has led to a rapid increase in the size and usage of sequence databases [86].

Genome databases are growing exponentially. Historically, these databases have been doubling in size about every 22 months, but that rate has rapidly accelerated due to the enormous growth in data from ESTs (expressed sequence tags). The current doubling time for EMBL is now down to under 8 months. Database growth rate will continue for the foreseeable future, since multiple concurrent genome projects have begun, with more to come. Table 1.1 shows the total length of all sequences in GenBank, and the total number of sequences in GenBank as a function of time.

The CB area is also referred to as *bioinformatics*. The two names are used interchangeably, but there seems to be a consensus forming where CB is used to refer to activities which mainly focus on constructing algorithms or programs that address problems with biological relevance, while bioinformatics is used to refer to activities which mainly focus on constructing and using computational tools to analyze available biological data. It should be emphasized that this distinction between CB and bioinformatics only serves

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Table 1.1: Growth rate of GenBank [1]

| **Year** | **Base pairs** | **Sequences** |
|---|---|---|
| 1982 | 680,338 | 606 |
| 1983 | 2,274,029 | 2,427 |
| 1984 | 3,368,765 | 4,175 |
| 1985 | 5,204,420 | 5,700 |
| 1986 | 9,615,371 | 9,978 |
| 1987 | 15,514,776 | 14,584 |
| 1988 | 23,800,000 | 20,579 |
| 1989 | 34,762,585 | 28,791 |
| 1990 | 49,179,285 | 39,533 |
| 1991 | 71,947,426 | 55,627 |
| 1992 | 101,008,486 | 78,608 |
| 1993 | 157,152,442 | 143,492 |
| 1994 | 217,102,462 | 215,273 |
| 1995 | 384,939,485 | 555,694 |
| 1996 | 651,972,984 | 1,021,211 |
| 1997 | 1,160,300,687 | 1,765,847 |
| 1998 | 2,008,761,784 | 2,837,897 |
| 1999 | 3,841,163,011 | 4,864,570 |
| 2000 | 11,101,066,288 | 10,106,023 |
| 2001 | 15,849,921,438 | 14,976,310 |
| 2002 | 28,507,990,166 | 22,318,883 |
| 2003 | 36,553,368,485 | 30,968,418 |
| 2004 | 44,575,745,176 | 40,604,319 |

to expose the main focus of the work. CB spans several classical areas such as biology, chemistry, physics, statistics and computer science, and the activities in the area are numerous. From a computational point of view the activities are ranging from algorithmic theory focusing on problems with biological relevance, via construction of computational tools for specific biological problems, to experimental work where a laboratory with test tubes and microscopes is substituted with a fast computer and a hard disk full of computational tools written to analyze huge amounts of biological data to prove or disprove a certain hypothesis.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Algorithms for solving CB problems are often associated with long runtimes. This is due to various factors:

- Biological data are obtained by experiments which are prone to errors. The need to deal with errors and uncertainties results in algorithms with high complexities.
- Many problems involve seemingly well-behaved polynomial time algorithms (such as all-to-all comparisons) but have massive computational requirements due to the large data sets that must be analyzed. For example, the assembly of the human genome in 2001 from the many short segments of sequence data produced by sequence robots required approximately 10,000 CPU hours [145].
- Many problems are compute-intensive due to their inherent algorithmic complexities (such as the protein folding and reconstructing evolutionary histories from molecular data). Some are known to be NP-hard. (An NP-hard problem is one for which an exact solution is conjectured by computer scientists to not be solvable in polynomial time, that is, an NP-hard problem requires more steps than can be grounded by a polynomial.) Thus, while NP-hard problems are thought to be intractable, HPC may provide sufficient capability for evaluating bio-molecular hypotheses or solving more limited but meaningful instances.

The work presented in this thesis is mainly concerned with constructing efficient HPC algorithms that address CB problems.

## 1.2 Motivation

The curves with dots and circles in Figure 1.2 show the growth rate of GenBank. Note that the vertical scale is logarithmic and the curves appear approximately as straight lines. This means that the size of GenBank is increasing exponentially with time. From

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



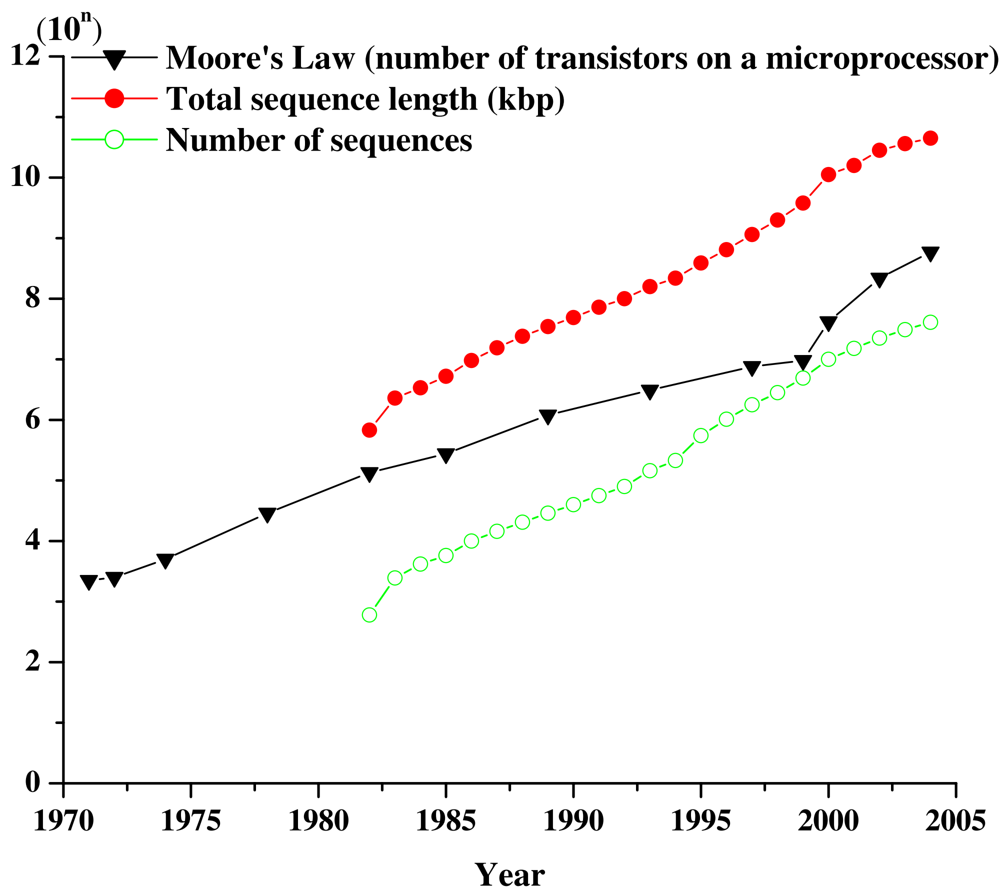



Figure 1.2: Comparison of the rate of growth the GenBank sequence (data from Table 1.1) with the rate of growth of the number of transistors in personal computer chips (Moore's law: data from Table 1.2).

these two lines we can estimate that the database doubles in size about every 1.4 years. Interestingly, the curve for the number of sequences almost exactly parallels the curve for the total length. This means that the typical length of one sequence entry in GenBank has remained at close to 1000. There are, of course, enormous variations in length between different sequence entries.

However, according to Moore's law, the number of transistors on a microprocessor would double approximately every 2.09 years; this is much slower than the growth rate of genetic sequence databases. Data on the size of Intel PC chips (Table 1.2) show that

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Table 1.2: The growth of the number of transistors in personal computer processors [7]

| Microprocessor | Year of Introduction | Transistors |
|---|---|---|
| 4004 | 1971 | 2,250 |
| 8008 | 1972 | 2,500 |
| 8080 | 1974 | 5,000 |
| 8086 | 1978 | 29,000 |
| Intel286 | 1982 | 134,000 |
| Intel386™processor | 1985 | 275,000 |
| Intel486™DX processor | 1989 | 1,200,000 |
| Intel®Pentium®processor | 1993 | 3,100,000 |
| Intel®Pentium®II processor | 1997 | 7,500,000 |
| Intel®Pentium®III processor | 1999 | 9,500,000 |
| Intel®Pentium®4 processor | 2000 | 42,000,000 |
| Intel®Itanium®2 processor | 2002 | 220,000,000 |
| Intel®Itanium®2 processor (9MB cache) | 2004 | 592,000,000 |

this exponential increase is still continuing. Looking at the data more carefully, however, we see that the estimate of doubling every year is rather overoptimistic. The chip size is actually doubling every two years. Although extremely impressive, this is slower than the rate of increase of GenBank (see Figure 1.2). As to supercomputers, Jack Dongarra and colleagues from the University of Tennessee have introduced the LINPACK benchmark, which measures the speed of computers at solving a complex set of linear equations. A list of the top 500 supercomputers according to this benchmark is published twice yearly. According to the LINPACK benchmark, the computing power of supercomputers doubles every 1.04 years [18]. So supercomputers can beat GenBank for the moment. Table 1.3 shows growth rates of GenBank, PC chips and supercomputer speed.

Table 1.3: Comparison of rates of increase of different data explosion curves

| Type of data | Doubling time |
|---|---|
| GenBank (total sequence length) | 1.44 |
| PC chips (number of transistors) | 2.09 |
| Supercomputer speed (LINPACK benchmark) | 1.04 |

With the publication of the human genome in 2001, we can now truly say that we

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



are in the "post-genome age". The availability of complete genomes is tremendously important for evolutionary studies. For the first time we can begin to compare whole sets of genes between organisms, not just single genes. For the first time we can begin to study the processes that govern the evolution of whole genomes. This is therefore an exciting time to be in the CB area. The study of large amounts of quantitative biological data poses challenges for scientists and provides a staggeringly wide field of opportunities for HPC exploration (see Table 1.3): by finding new and ever-more-efficient methods to analyze all this biological data, we may be able to unlock potential answers to many of the "secrets" of life as we have known it. This is a new area of biological sciences where computational methods are essential for the progress of the experimental science, and where algorithms and experimental techniques are being developed side by side.

Traditionally, supercomputers were rare and available for only the most critical problems. Since the mid-1990s, however, the availability of supercomputers has changed dramatically. With multi-threading support built into the latest microprocessors and the emergence of multiple processor cores on a single silicon die, supercomputers are becoming ubiquitous. Now, almost every university computer science department has at least one parallel computer. Virtually all oil companies, automobile manufacturers, drug development companies, and special effects studios use parallel computers.

The availability of very fast processors in supercomputers, together with the widespread utilization of networks, led to the notion of a "virtual parallel computer" that connected several fast microcomputers by means of a fast LAN [1.1]. This distributed-memory system was called a *multi-computer* or a *parallel computer*. Clusters of workstations and beowulf-clusters [17] are good examples of parallel computer systems. Beowulf-clusters are popular since they are composed of ordinary hardware components (like any PC) together with public domain software (like Linux, PVM or MPI).

[1.1] Local Area Network

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Grid systems are the new paradigm of parallel computers. The SETI@home project [14, 23] provides a fascinating example of the power of grids. The project seeks evidence of extraterrestrial intelligence by scanning the sky with the world's largest radio telescope, the Arecibo Telescope in Puerto Rico. The collected data is then analyzed for candidate signals that might indicate an intelligent source. The computational task is beyond even the largest supercomputer, and certainly beyond the capabilities of the facilities available to the SETI@home project. The problem is solved with public resource computing, which turns PCs around the world into a huge parallel computer connected by the Internet. Data is broken up into work units and distributed over the Internet to client computers whose owners donate spare computing time to support the project. Each client periodically connects with the SETI@home server, downloads the data to analyze, and then sends the results back to the server. The client program is typically implemented as a screen saver so that it will devote CPU cycles to the SETI problem only when the computer is otherwise idle. A work unit currently requires an average of between seven and eight hours of CPU time on a client. More than 205,000,000 work units have been processed since the start of the project. More recently, similar technology to that demonstrated by SETI@home has been used for a variety of public resource computing projects as well as internal projects within large companies utilizing their idle PCs to solve problems ranging from drug screening to chip design validation.

Given the exponential growth in the size of genomic and protein databases and the availability of complete genomes of complex organisms, the CB area has taken dramatic leaps forward with the availability of computational resources. The effective use of parallel computers will become increasingly important in CB. This continues to remain a largely unexplored territory, and is the principal motivation behind our work.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



## 1.3 Objectives

In this thesis, we have investigated how the concept of parallel patterns can be applied in CB. According to the characteristics of popular CB algorithms and the important features that should be present in the parallel pattern-based program design and development method, the objective of our research is to systematically design efficient parallel algorithms for compute-intensive CB problems. Our research work includes the following aspects:

- Characteristic Analysis: The first step to design an efficient parallel algorithm is to analyze and identify the characteristics of the sequential algorithm. Characteristics of sequential algorithms will affect the design and development of corresponding parallel programs greatly.

- Partitioning and Communication Scheme: According to the characteristics of the sequential algorithm, we will design appropriate partitioning and communication schemes on parallel computers. The partitioning and communication scheme significantly affects the performance of a parallel program. So, it is important to determine what scheme is the most appropriate one for each algorithm according to its characteristics.

- Separation of Specification: This is the central feature of parallel pattern-based programming methods. It means that it should be possible to specify the pattern (i.e., the parallelization aspects of the application) separately from the sequential application code. This key feature is crucial for rapid prototyping and performance tuning of a parallel application. It also allows for the application code and its parallelization structures to be evolved in an independent manner. It allows for rapid prototyping of programs, and permits users to quickly experiment with alternative parallel communication structures.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



- Extensibility: A design pattern does not represent a single solution to a given problem, but rather embodies a family of potential solutions. So, under limited parallel patterns, an extensible mechanism should be provided to meet novel utilities.

- Execution Performance: The maximum performance possible should be achievable. There will always be limitations to the achievable performance. The complexity and interdependence of components external to the system (communication subsystem, operating system, network, etc.) make it very difficult to abstract and still attain the highest possible performance.

## 1.4 Contributions

The contributions of our work are briefly summarized below:

1. Characteristic analyses for popular CB algorithms:

   We have analyzed two categories of popular CB problems: those that can be addressed analytically, and those that can not. These two categories are addressed in turn by two respective classes of algorithms: analytical solution approaches and heuristic approaches. Analysis and understanding of characteristics of CB algorithms will help us develop efficient parallel applications for them.

2. Design of a tunable coarse-grained partitioning and communication scheme for parallel DP algorithms:

   We have proposed a general parameterized coarse-grained parallel algorithm for regular and irregular DP applications. By introducing two performance-related parameters, we can tradeoff between computation time and communication time by tuning these two parameters and thus obtain the maximum possible performance.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



We have demonstrated how this algorithm leads to substantial performance gains for regular and irregular DP applications.

3. Design of a new hierarchical parallel genetic algorithm (HPGA) for protein folding on computational grids:

   According to the characteristics of GAs, we have designed a new HPGA for protein folding on PC clusters and computational grids. The high level part of this HPGA is mapped onto the grid layer and the low level part of it is mapped onto the cluster layer. Two kinds of parallel communication modules, the module within the cluster and the module between clusters, are used to implement the intra-cluster and the inter-cluster communication within distributed populations.

4. Design and development of a parallel pattern-based framework:

   We have developed a parallel pattern-based framework to facilitate the semi-automatic development of HPC programs. The underlying programming technique is based on object-oriented programming (OOP) and generic programming (GP) which is a program design technique that deals with finding abstract representations of algorithms, data structures, and other software concepts [144]. GP techniques provide a better environment for code reuse. An important aspect of our framework is the generic representation for a set of patterns, i.e. a generic pattern. With this generic pattern, we mainly focus on the extensibility of the framework rather than how many limited patterns it can support. The user can extend the generic pattern by specifying the application-dependent template parameters. Different specialization will lead to different implementation strategies for a concrete parallel application.

5. Evaluate experimental results:

   HPC brings new methods into the experimental results evaluation in CB. Although speedup is a well-accepted way of measuring the efficiency of a parallel algorithm, in

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



the CB community the topic of parallel speedups has raised significant controversy. In this thesis, we have introduced some new concepts into the deterministic parallel application field. By using them, we have provided appropriate ways to explain the super-linear speedups achieved by our HPGA for the PFP.

## 1.5 Synopsis of the Thesis

The rest of the thesis is organized as follows:

- Chapter 2 presents a general survey of the state-of-the-art HPC backgrounds and techniques. We review algorithm design techniques, popular genome analysis tasks, parallel architectures, parallel program design environments and methods.
- Chapter 3 discusses sequential characteristics of two popular CB algorithms: DP algorithms and GAs. We give a general classification for sequential DP algorithms as well as the theoretical foundation of GAs.
- Chapter 4 presents corresponding partitioning and communication schemes according to the characteristics of CB applications. For DP algorithms, a tunable coarse-grained algorithm is introduced. We have also designed a new HPGA for the PFP with hydrophobic-hydrophilic (HP) lattice models.
- Chapter 5 elaborates the implementation detail of our parallel pattern-based framework. This framework is designed and developed using multi-paradigm techniques with the objectives of extensibility and reusability. We also presents a performance evaluation of our framework on different parallel architectures. According to super-linear speedups measured in the HPGA for PFP, we introduce a mathematical model to explain and predict the experimental results.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



- Chapter 6 concludes the achievements of our research work and suggests possible areas for future work.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Chapter 2

# Literature Review

## 2.1 Introduction

The task of parallel programming in CB is to solve compute-intensive problems more efficiently than their sequential counterparts. However, because a parallel program is more complex than an equivalent sequential program, to realize this increase in speed some challenges must be overcome first and this daunting task usually falls on a small number of experts [26]. In this chapter, we review algorithm design techniques, parallel architectures, parallel program design environments and methods.

## 2.2 Algorithm Design Techniques

The word *algorithm* comes from the name of the $9^{th}$ century Persian mathematician Abu Abdullah Muhammad bin Musa al-Khwarizmi. The word *algorism* originally referred only to the rules of performing arithmetic using Arabic numerals but evolved into *algorithm* by the $18^{th}$ century. The word has now evolved to include all definite procedures for solving problems or performing tasks. With the advent of automated computing devices such as modern computers, an algorithm has in most contexts become synonymous with a description that can be turned into a computer program that instructs a

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



computer how to solve the problem addressed by the algorithm. The ability of modern computers to perform billions of simple calculations per second and to store billions of bits of information, makes it possible by using the proper computer programs to address a wide range of problems that would otherwise remain out of reach. Such possibilities have spawned several interdisciplinary activities where the objective is to utilize the capacities of computers to gain knowledge from huge amounts of data. An important part of such activities is to construct good algorithms that can serve as basis for the computer programs that are needed to utilize the capacities of computers.

Over the last forty years, computer scientists have discovered that many algorithms share similar ideas, even though they solve very different problems. There appear to be relatively few basic techniques that can be applied when designing an algorithm, and we cover the most common algorithm design techniques in this section.

### 2.2.1 Exhaustive Search

An *exhaustive search*, or *brute force*, algorithm examines every possible alternative to find one particular solution. In general, though, exhaustive search algorithms are too slow to be practical for anything but the smallest instances and we will try to avoid the exhaustive algorithms or how to finesse them into faster versions.

### 2.2.2 Branch-and-Bound Algorithms

In certain cases, as we explore the various alternatives in a brute force algorithm, we discover that we can omit a large number of alternatives, a technique that is often called *branch-and-bound*, or *pruning*.

Branch-and-Bound is a general search method. It starts by considering the root problem (the original problem with the complete feasible region), the lower-bounding

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



and upper-bounding procedures are applied to the root problem. If the bounds match, then an optimal solution has been found and the procedure terminates. Otherwise, the feasible region is divided into two or more regions, these subproblems partition the feasible region. The algorithm is applied recursively to the subproblems. If an optimal solution is found to a subproblem, it is a feasible solution to the full problem, but not necessarily globally optimal. If the lower bound for a node exceeds the best known feasible solution, no globally optimal solution can exist in the subspace of the feasible region represented by the node. Therefore, the node can be removed from consideration. The search proceeds until all nodes have been solved or pruned, or until some specified threshold is met between the best solution found and the lower bounds on all unsolved subproblems.

### 2.2.3 Greedy Algorithms

Many algorithms are iterative procedures that choose among a number of alternatives at each iteration. *Greedy algorithms* choose the "most attractive" alternative at each iteration, without regard for future consequences. Generally, this means that some local optimum is chosen. This 'take what you can get now' strategy is the source of the name for this class of algorithms. When the algorithm terminates, we hope that the local optimum is equal to the global optimum. If this is the case, then the algorithm is correct; otherwise, the algorithm has produced a suboptimal solution. If the best answer is not required, then simple greedy algorithms are sometimes used to generate approximate answers, rather than using the more complicated algorithms generally required to generate an exact answer.

### 2.2.4 Dynamic Programming

Some algorithms break a problem into smaller subproblems and use the solutions of the subproblems to construct the solution of the larger one. During this process, the number

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



of subproblems may become very large, and some algorithms solve the same subproblem repeatedly, needlessly increasing the runtime. *Dynamic programming* organizes computations to avoid recomputing values that you already know, which can often save a great deal of time.

### 2.2.5 Divide-and-Conquer Algorithms

One big problem may be hard to solve, but two problems that are half the size may be significantly easier. In these cases, *divide-and-conquer algorithms* fare well by doing just that: splitting the problem into smaller subproblems, solving the subproblems independently, and combining the solutions of subproblems into a solution of the original problem. The situation is usually more complicated than this and after splitting one problem into even smaller sub-subproblems, and so on, until it reaches a point at which it no longer needs to recurse. A critical step in many divide-and-conquer algorithms is the recombining of solutions to subproblems into a solution for a larger problem. Often, this merging step can consume a considerable amount of time.

The divide-and-conquer approach is similar to the DP in that the solution of a large problem depends on previously obtained solutions to easier subproblems. The significant difference, however, is that DP permits subproblems to overlap. By overlap, we mean that the subproblem can be used in the solution of two different subproblems. In contrast, the divide-and-conquer approach creates subproblems that are completely separate and can be solved independently.

### 2.2.6 Machine Learning

*Machine learning algorithms* often base their strategies on the computational analysis of previously collected data. Machine learning algorithms are presented with training data, which are used to derive important insights about the parameters (often hidden).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Once an algorithm has been suitable trained, it can apply these insights to the analysis of a test sample. As the amount of training data increases, the accuracy of the machine learning algorithm typically increases as well. The parameters that are learned during training represent knowledge; application of the algorithm with those parameters to new data (not used in the training phase) represents the algorithm's use of that knowledge.

### 2.2.7 Randomized and Heuristic Algorithms

*Randomized algorithms* make random decisions throughout their operation. At first glance, making random decisions does not seem particularly helpful. Basing an algorithm on random decisions sounds like a recipe for disaster, but an eighteenth-century French naturalist, Comte de Buffon, proved the opposite by developing an algorithm to accurately compute $\pi$ by randomly dropping needles on a sheet of paper with parallel lines. The fact that a randomized algorithm undertakes a nondeterministic sequence of operations often means that, unlike deterministic algorithms, no input can reliably produce worst-case results. Randomized algorithms are often used in hard problems where an exact, polynomial-time algorithm is not known.

*Genetic algorithms* attempt to find solutions to problems by mimicking biological evolutionary processes, with a cycle of random mutations yielding successive generations of solutions. Thus, they emulate reproduction and "survival of the fittest".

The term *heuristic* is used for algorithms which find solutions among all possible ones, but they do not guarantee that the best will be found, therefore they may be considered as approximately and not accurate algorithms. An example of this would be simulated annealing algorithms, a class of heuristic probabilistic algorithms that vary the solution of a problem by a random amount.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



## 2.3 Popular Genome Analysis Tasks

CB has been revolutionized by advances in both computer hardware and software algorithms. High-throughput techniques for DNA sequencing and analysis of gene expression have led to exponential growth in the amount of publicly available genomic data. Biologists are keen to analyze and understand this data, since genetic sequences determine biological structure, and thus the function. Understanding the function of biologically active molecules leads to understanding biochemical pathways and disease-prevention strategies and cures, along with the mechanisms of life itself. However, the amount of genomic data is now so great that traditional database approaches are no longer sufficient for rapidly performing life science queries involving the fusion of data types. Computing systems are now so powerful that it is possible for researchers to consider modelling the folding of a protein or even the simulation of an entire human body. As a result, computer scientists and biomedical researchers face the challenge of transforming data into models and simulations that will enable scientists for the first time to gain a profound understanding of the deepest biological functions. Traditional uses of HPC systems in physics, engineering, and weather forecasting involve problems that often have well-defined and regular structures. In contrast, many problems in CB are irregular in structure, are significantly more challenging for software engineers to parallelize.

In this thesis, we have focused on two popular genome analysis tasks: sequence alignment and protein folding.

### 2.3.1 Sequence Alignment

*Sequence alignment* is the most basic sequence analysis task. It is used to tell whether two or more sequences are related and give an impression how close their relationship is in terms of sequence similarity. It is centrally important to find the best possible alignment

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



of sequences for bioinformatics and data processing after routine laboratory procedures like sequencing nucleic acids. A list of problems that are related to sequence alignment are identified in [130]. These problems are summarized as follows:

- Two sequences over the same alphabet are almost equal, except for a few isolated insertions, deletions, and substitutions of characters. The average frequency of these differences is very low; however, their exact positions must be found. This problem occurs when a gene is sequenced by two different labs and the results are compared.

- Two sequences over the same alphabet, with a few hundred characters each, are given. The problem is to decide whether there is a prefix of one sequence that is similar to a suffix of the other. If the answer is yes, the matching prefix and suffix must be produced. This problem appears when small DNA fragments are assembled into a longer sequence in the process of large-scale DNA sequencing. Often, this problem must be applied pairwise to several hundred sequences, most of which are unrelated.

- Two sequences over the same alphabet, with a few hundred characters each, are given. The problem is to decide whether there are two substrings, one from each sequence, that are similar. This problem appears when searching for local similarities in large sequence databases. In this context, a sequence must be compared against thousands of others.

All these problems can be solved using the same basic algorithmic idea that is used for solving the sequence comparison problem. The formal details of the pairwise sequence alignment problem are given as follows.

Consider the following pair of DNA sequences: **TATCAG** and **TAATCCG**. At a glance, they look very much alike, and this becomes more obvious when they are aligned

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



one above the other:

**T−ATCAG**

**TAATCCG**

The only differences are an extra **A** in the second sequence and a change from **A** to **C** in the second to last position. Note that a gap, marked with a "−", is introduced in the first sequence in order to allow the bases before and after the gap to align perfectly. This is a sample alignment of the two sequences.

Formally, an alignment of two sequences is obtained from the original sequences by inserting gaps until the resulting sequences are of the same size. An alignment also must obey the restriction that gaps cannot appear in the same position in both sequences. The example above satisfies the definition of an alignment.

The goal of the sequence comparison operation is to find an optimal alignment of two sequences relative to a *cost function*. One type of cost function is obtained by assigning a *score* to an alignment in the following manner: each column of the alignment is given a value based on the two characters forming the column, and the total score of the alignment is the sum of all the values assigned to its columns. If a column has two identical characters, it is valued with $+2$ (i.e. a *match*). Different characters are valued with $-1$ (i.e., a *mismatch*). Finally, if a gap is present, the column is valued with $-2$ (i.e., a *gap penalty*). An optimal alignment is one with maximal total score among the total scores of all possible alignments between the two sequences. In general, there may be many optimal alignments between two sequences.

For the alignment in the example above, there are five columns with identical characters, one column with distinct characters, and one column with a gap, giving a total score of:

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



$$5 \times 2 + 1 \times (-1) + 1 \times (-2) = 7$$

The particular values +2, −1, and −2 were chosen because they constitute a simple implementation of the policy of rewarding matches, and penalizing mismatches and gaps. In practice, the value which is assigned to a column depends on the probability with which the character from the first row can transform itself into the character from the second row after a certain number of evolutionary steps. With regard to proteins, the amino acids have biochemical properties that influence the way they replace each other during the evolution of a protein [130]. For example, it is more likely that amino acids of similar sizes will be substituted for one another than those of widely different sizes. The tendency to bind with water molecules also influences the probability of mutual substitution. Because protein comparisons are usually performed to establish an evolutionary relation between sequences, it is important to use scoring functions that reflect these probabilities accurately.

Often, the best method to derive similarity scores for pairs of residues is to empirically observe the actual substitution rates; doing so is advisable because it is difficult to account for all the factors that influence the probability of mutual substitution of amino acids. A standard procedure for achieving this goal is based on an important family of scoring matrices, known as the *point accepted mutations* of *percent of accepted mutations* PAM matrices [52].

Before two sequences are aligned, the evolutionary distance at which to compare them must be chosen. The PAM matrices are functions of this distance. For instance, a PAM-250 matrix is suitable for comparing sequences that are 250 units of evolution apart. If no information on the true evolutionary distance between the two sequences is available, the recommended approach is to align the sequences using several PAM matrices that cover a wide range-for instance, PAM-40, PAM-120, and PAM-250. In general, low PAM numbers are good for finding short, strong, local similarities, while high PAM numbers

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



detect long, weak ones [130]. It should also be noted that PAM matrices consider the mutations at the amino acid level only, without involving the DNA level.

### 2.3.2 Protein Structure Prediction

Proteins are large molecules found in all organisms built from a chain of amino acids and are responsible for the structure, function, and regulation of cells, tissues, and organs. Protein structure prediction is the process of self-assembly of an amino acid sequence into the native 3D structure of the functioning protein. Proper functioning of a protein depends on its ability to fold into its native structure. Failure to do so causes a loss of biological function and often results in illness or fatal disease. Examples are cystic fibrosis; Parkinson's; Alzheimer's; and Prion diseases (such as Creutzfeldt-Jakob Disease and Bovine Spongiform Encephalopathy, or mad cow disease). Hence, a biomedical researcher's understanding of how a protein folds has direct medical significance. The protein-folding problem is computationally challenging, and many techniques, ranging from experimental to theoretical, are being investigated for their accuracy and speed in predicting 3D structures.

Many protein structure prediction methods are based on a basic thermodynamic hypothesis: proteins tend to fold into a global minimum free energy state. Consequently, researchers predict protein structures in two steps: first, design a scoring function to reflect the relationships among the amino acids in the native states, and second, design or employ certain algorithms to optimize the scoring function. The result is the three dimensional structure of a protein. The scoring function can be constructed from the physicochemical principles of protein folding, and reflects the real energy of proteins in a native state. Alternatively, the scoring function can be a knowledge-based potential function, measuring the probability distribution of the possible conformational arrangements of a protein sequence. Traditionally, the scoring function is also called "energy function"

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



even if the scoring function does not reflect the real energy of proteins.

During the last three decades, many protein structure prediction methods have been proposed, and in the past ten years, many structure prediction computer programs have been developed to ease or speed up manual predictions. The methods can be grouped into three categories:

- *homology modelling,*
- *protein threading*,
- *ab initio folding.*

According to their most suitable target sequences. The first two are template-based, and the third one builds protein structures without referring to any structural template.

Homology modelling is for those target sequences obviously having homologue with a known three-dimensional structure. This kind of target sequences are called homology modelling targets. Homology modelling builds the tertiary structure of a target sequence by comparing the target sequence to all of its homologous sequences, recognizing the most conserved segments through multiple alignments, copying coordinates for these conserved segments from one homolog with a known structure, and finally, refining the whole structure through the energy minimization technique. Protein threading is suitable for those target sequences which have no homologous templates but do have the same fold as some templates. The three dimensional structure of a target sequence is built by placing its amino acids one-by-one and sequentially into different positions of the template which has the same fold as the target sequence.

*Ab initio* folding is the preferred methodology to apply to those targets that could not be predicted by the two former methods, that is, the targets that do not have the same fold as some templates or do not have the homologous proteins with a known structure.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



This kind of targets are called new fold targets. Ab initio folding has also been called the "new fold" method in recent years, because it is appropriate for targets that do not have the same fold as any protein with a known structure. Ab initio folding depends on a scoring function which can accurately describe the native state of proteins and on an efficient algorithm to optimize the scoring function.

Besides the traditional three structure prediction methods, a class of computer programs have been recently developed to combine the outcomes of several different prediction programs by certain consensus methods, such as 3DSX, developed by Daniel Fischer [59], 3D-Jury by Lesek Rychlewski [76], and PMODX by Janu Bujnicki [100].

There are three representative approximation algorithms for the protein structure prediction problem:

- **Monte Carlo Sampling and Genetic Algorithm:** Bryant [39, 106] et al. used GAs to search for the optimal alignment between the target sequence and the template. They began from one initial sequence-template alignment from the current alignment. The energy function is calculated for each generated alignment, and the alignment is kept according to a certain probability calculated from its energy function value. Theoretically, the GA can converge to the optimal alignment (i.e., minimal energy status) if enough computational time is given. But usually "enough computational time" is unaffordable if a long sequence is threaded.

- **Interaction-Frozen Approximation [35, 77, 78, 150]:** This is a type of iterative algorithm. Given an initial alignment between the sequence and the template, one end of each contact (interaction) is fixed to the residue in the current alignment position. Then a dynamic programming algorithm is used to search for the next alignment based on the current "frozen" pairwise contact potential. This step is repeated for a number of times or until the next alignment is the same as the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



current alignoent (i.e., the algorithm converges). The outstanding drawback of this method is that no convergence is guaranteed. that is, no optimal alignoent is guaranteed.

- **Recursive Dynamic Programming [141]:** This algorithm repeatedly uses a DP algorithm to match the target sequence to the template. At each iteration, the local alignoent between the target sequences and the templates is searched by the DP algorithm. A segoent of the target sequence is fixed onto a segoent of the template if a significant similarity is attained. If the alignoent position of one segoent of sequence is fixed, then all pairwise contact potential involved with the residues in this segoent is easy to handle. This is possible because one end of the pairwise contacts is frozen. The iteration is repeated until no significant similarity is found. those unmatched segoents of the target sequence are interpreted as gaps.

## 2.4 Parallel Architectures: A Brief Introduction

There are dozens of different parallel architectures, such as networks of workstations, clusters of off-the-shelf PCs, massively parallel supercomputers, tightly coupled symmetric multiprocessors, and multiprocessor workstations. In this section, we give an overview of these systems, focusing on the characteristics relevant to the programmer.

### 2.4.1 Flynn's Taxonomy

By far the most common way to characterize these architectures is Flynn's taxonomy [61]. He categorizes all computers according to the number of instruction streams and data streams they have, where a stream is a sequence of instructions or data on which a computer operates. In Flynn's taxonomy, there are four possibilities: SISD, SIMD, MISD, and MIMD.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



**Single Instruction, Single Data (SISD)**

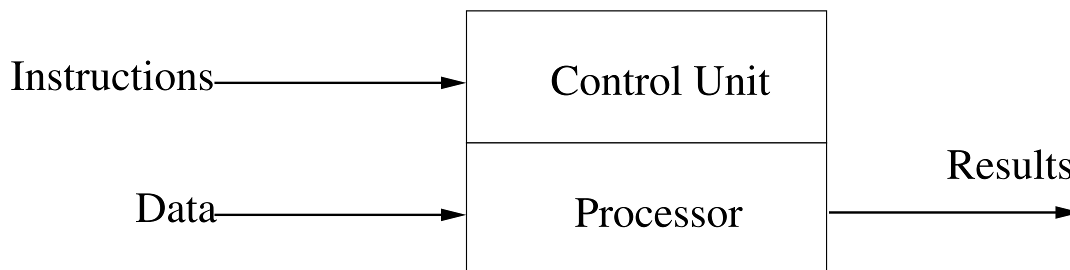



Figure 2.1: The Single Instruction, Single Data (SISD) architecture

The category SISD refers to computers with a single instruction stream and a single data stream. Figure 2.1 shows the illustration for SISD architecture. Standard serial computers fall into this category. One instruction is executed per unit time to produce one useful result because the stream of instructions and stream of data can be viewed as being tightly coupled in SISD.

**Single Instruction, Multiple Data (SIMD)**

In a SIMD system, a single instruction stream is concurrently broadcast to multiple processors, each with its own data stream (as shown in Figure 2.2). The original systems from Thinking Machines and MasPar can be classified as SIMD. The CPP DAP Gamma II and Quadrics Apemille are more recent examples; these are typically deployed in specialized applications, such as digital signal processing, that are suited to fine-grained parallelism and require little interprocess communication. Vector processors, which operate on vector data in a pipelined fashion, can also be categorized as SIMD. Exploiting this parallelism is usually done by the compiler.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



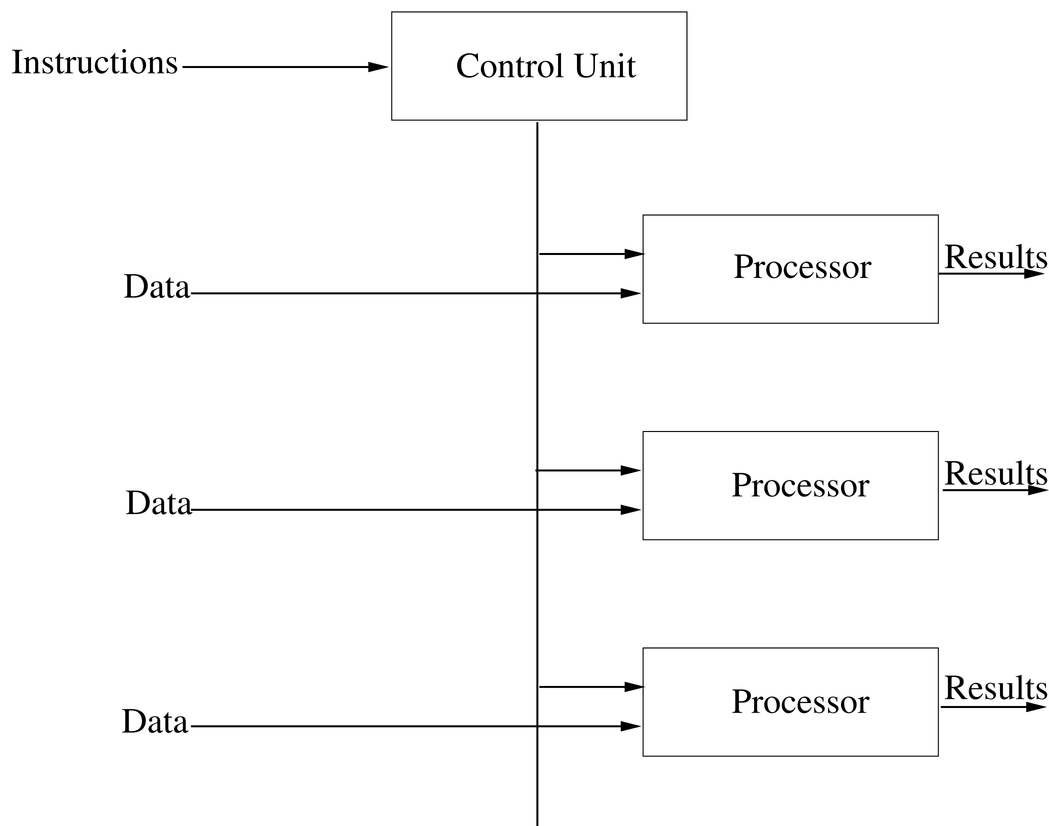



Figure 2.2: The Single Instruction, Multiple Data (SIMD) architecture

### Multiple Instruction, Single Data (MISD)

The category MISD (see Figure 2.3) refers to computers with multiple instruction steams but only a single data stream. There are few machines in this category, none that have been commercially successful or had any impact on computational science. One type of system that fits the description of an MISD computer is a systolic array [5], which is a network of small computing elements connected in a regular grid. All the elements are controlled by a global clock. On each cycle, an element will read a piece of data from one of its neighbors, perform a simple operation.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



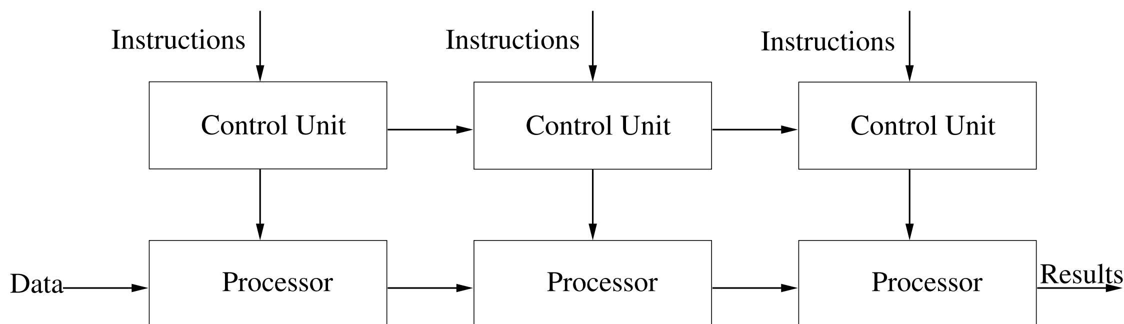



Figure 2.3: The Multiple Instruction, Single Data (MISD) architecture

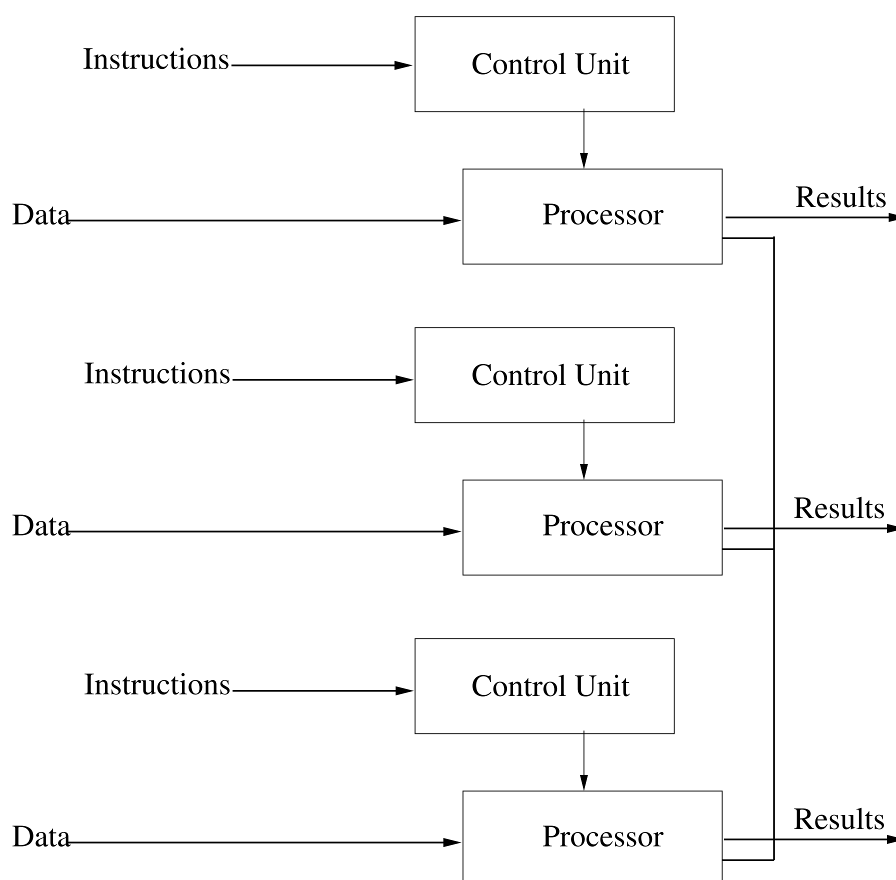



Figure 2.4: The Multiple Instruction, Multiple Data (MIMD) architecture

### Multiple Instruction, Multiple Data (MIMD)

In a MIMD system, each processing element has its own stream of instructions operating on its own data. This architecture, shown in Figure 2.4, is the most general of the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



architectures in that each of the other cases can be mapped onto the MIMD architecture. The vast majority of modern parallel computers fit into this category.

### 2.4.2 A Further Breakdown of MIMD

Most contemporary parallel computers fall into the MIMD category. Hence the MIMD designation is not particularly helpful when describing modern parallel architectures. Memory architecture has a strong influence on the global architecture of MIMD machines, becoming a key issue for parallel execution, and frequently determines the optimal programming model. Therefore, a further classification of MIMD which is based on the memory architecture is widely accepted. The classification results in three categories: *shared-memory system*, *distributed-memory system*, and *hybrid system*.

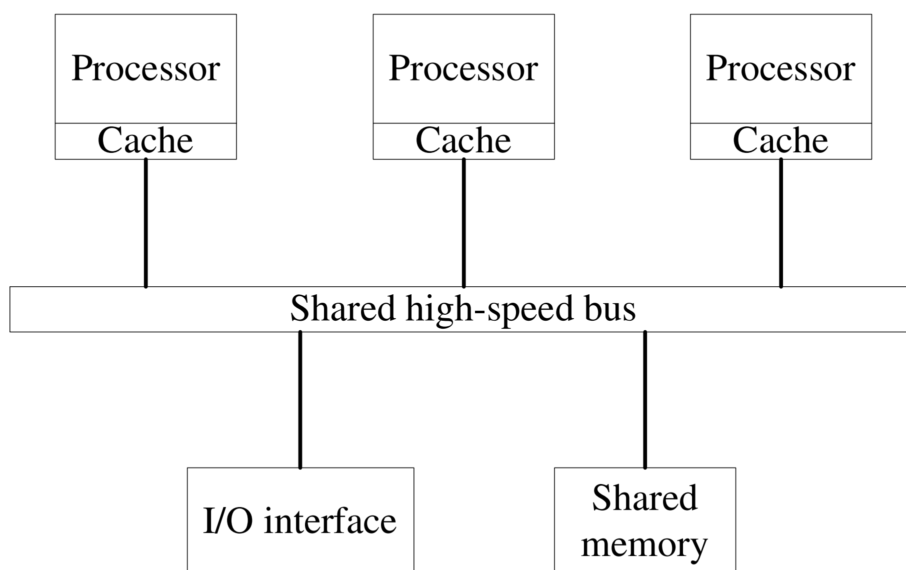



Figure 2.5: The Symmetric Multiprocessor (SMP) architecture

#### Shared-memory

In a shared-memory system, all processes share a single address space and communicate with each other by writing and reading shared variables.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



One main class of shared-memory systems is called SMPs (symmetric multiprocessors). As shown in Figure 2.5, all processors share a connection to a common memory and access all memory locations at equal speeds. SMP systems are arguably the easiest parallel systems to program because programmers do not need to distribute data structures among processors. Because increasing the number of processors increases contention for the memory, the processor/memory bandwidth is typically a limiting factor. Thus, SMP systems do not scale well and are limited to small numbers of processors.

#### Distributed-memory

In a distributed-memory system, each process has its own address space and communicates with other processes by message passing (sending and receiving messages). A schematic representation of a distributed memory computer is shown in Figure 2.6.

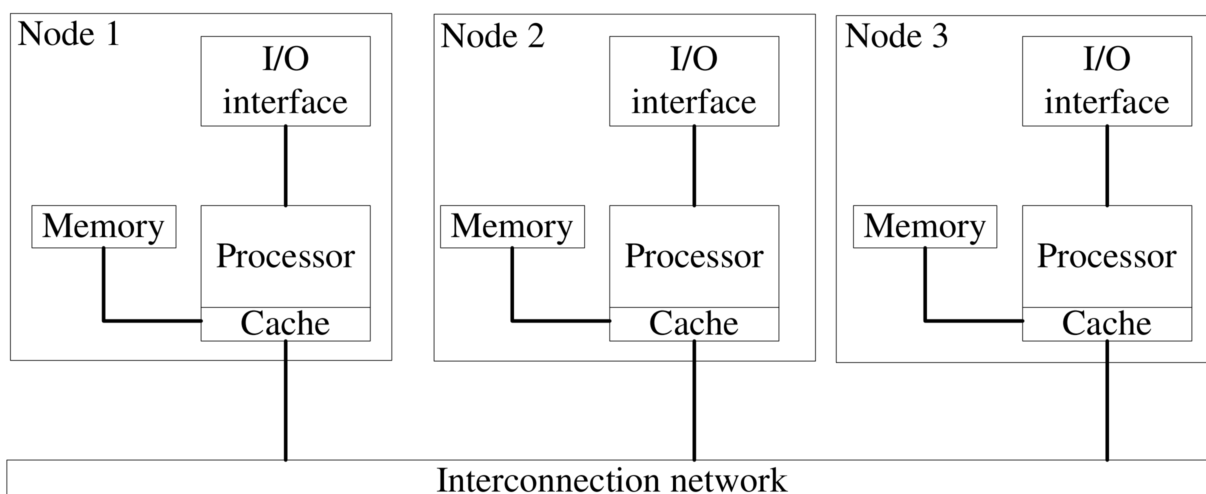



Figure 2.6: The distributed-memory architecture

Depending on the topology and technology used for the processor interconnection, communication speed can range from almost as fast as shared memory (in tightly integrated supercomputers) to orders of magnitude slower (for example, in a cluster of PCs interconnected with an Ethernet network). The programmer must explicitly program all

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



the communication between processors and be concerned with the distribution of data.

Distributed-memory computers are traditionally divided into two classes: MPP (massively parallel processors) and clusters. In an MPP, the processors and the network infrastructure are tightly coupled and specialized for use in a parallel computer. These systems are extremely scalable, in some cases supporting the use of many thousands of processors in a single system [107, 19].

Clusters are distributed-memory systems composed of off-the-shelf processors connected by an off-the-shelf network. Every processor is supported by its own memory and they form an independent computing unit together. Each of these independent computing units is called a *node* (see Figure 2.6). When the nodes are PCs running the Linux operating system, these clusters are called Beowulf clusters. As off-the-shelf networking technology improves, systems of this type are becoming more common and much more powerful. Clusters provide an inexpensive way for an organization to obtain parallel computing capabilities [17]. Pre-configured clusters are now available from many vendors. One frugal group even reported constructing a useful parallel system by using a cluster to harness the combined power of obsolete PCs that otherwise would have been discarded [83].

#### Hybrid systems

These systems are clusters of nodes with separate address spaces in which each node contains several processors that share memory.

Shared-memory architecture brings several advantages to CB applications. For instance, a single address map simplifies the design of parallel programs. In addition, there is no "time penalty" for communication between processes, because every byte of memory is accessible in the same amount of time from any CPU. However, shared-memory does not scale well as the number of processors in the computer increases. Distributed-memory

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



systems scale very well, on the other hand, but the lack of a single physical address map for memory incurs a "time penalty" for inter-process communication. Hybrid systems try to achieve the best of both shared and distributed memory architectures. A certain amount of memory physically attaches to each node (distributed architecture), but the hardware creates the image of a single memory for the whole system (shared architecture). In this way, the memory installed in any node can be accessed from any other node as if all memory were local with only a slight time penalty.

According to van der Steen and Dongarra's "Overview of Recent Supercomputers" [135], which contains a brief description of the supercomputers currently or soon to be commercially available, hybrid systems formed from clusters of SMPs connected by a fast network are currently the dominant trend in HPC. For example, in late 2003, four of the five fastest computers in the world were hybrid systems [18].

### 2.4.3 Grids

In 1998, Ian Foster and Carl Kesselman attempted a definition of the *computing grid* in [62]:

> *A computing grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.*

This definition of the grid focuses on the access to computing resource, data, and services on demand. In 2001, the concept of *Virtual Organization (VO)* is proposed in [65]:

> *A set of individuals and/or institutions defined by the sharing rules is called a virtual organization (VO).*

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Based on the concept of *VO*, the definition of the grid is refined to address social and policy issues in [65]:

> *The real and specific problem that underlies the grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.*

While Grid technology made great progress recently, there still remains technical problems to meet various requirements of QoS (Qualities of Service) when applications are executed on different types of platforms. To address the new challenges of Grid computing, the *Open Grid Services Architecture* (*OGSA*) [63] is proposed to evolve the current Grid infrastructure towards a Grid system architecture based on an integration of Grid and Web services concepts and technologies. *OGSA* defines:

- Grid service using a uniform exposed service semantics, and
- standard mechanisms for creating and naming Grid service instances.

It also supports:

- location transparency and multiple protocol bindings for service instances, and
- integration with underlying native platform facilities.

As more and more requirements are imposed on the Grid computing, new issues will arise. The Grid technologies will be continuously developed with the aim to address these new challenges as the Grid concept evolves.

### Grid Architecture

The grid architecture is fundamental for the establishment, management and exploitation of dynamic, cross-organizational *VO* resource sharing. As defined in [62, 65], the layered

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



grid architecture is shown in Figure 2.7.

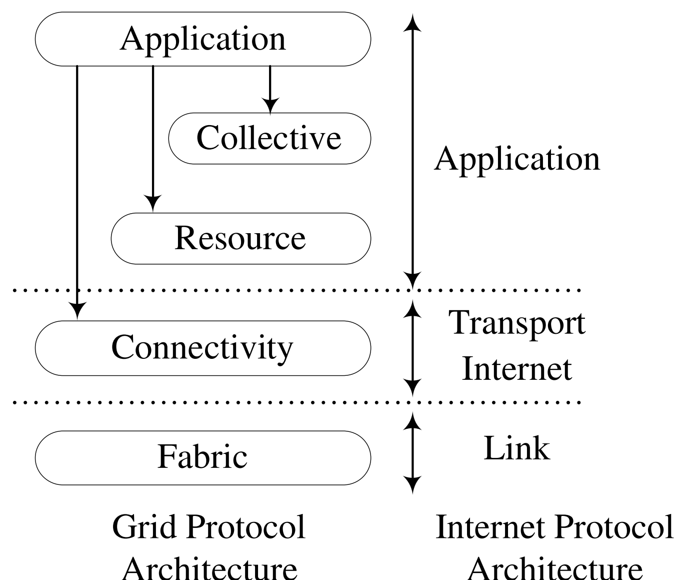



Figure 2.7: The layered grid architecture

Components within each layer in Figure 2.7 share common characteristics and can be built on the capabilities and behaviors provided by any lower layer. For example, protocols at *Resource and Connectivity* layer are designed so that they can be implemented on top of a diverse range of resource types, defined at the *Fabric* layer. On the other hand, they can be used to construct a wide range of global services and application-specific behaviors at the *Collective* layer. The detail of each layer is described as follows:

- Fabric: Interfaces to Local Control

  The grid *Fabric* layer provides basic grid protocols that enable grid applications to share resources, which can be, for example, computational resources, storage systems, network resources and sensors. *Fabric* components implement the local, resource-specific operations that occur on specific resources. These operations enable resource sharing operations at higher levels.

- Connectivity: Communicating Easily and Securely

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



The *Connectivity* layer defines core communication and authentication protocols required for grid-specific network transmission. Communication protocols enable the exchange of data between *Fabric* layer resources. Authentication protocols provide cryptographically secured mechanisms for verifying the identity of users and resources.

- Resource: Sharing Single Resources

  The *Resource* layer builds communication and authentication protocols of *Connectivity* layer to define protocols (and APIs, SDKs) for the secured negotiation, initiation, monitoring, control, accounting and payment of sharing operations on individual resources. *Resource* layer calls *Fabric* layer functions to access and control local resources.

- Collective: Coordinating Multiple Resources

  While the *Resource* layer is focused on interactions with a single resource, the *Collective* layer in the architecture contains protocols and services (and APIs, SDKs) that are not associated with any specific resource but rather interactions across collections of resources.

- Application

  The *Application* layer in the grid architecture comprises the user applications that operate within a *VO* environment. Applications are constructed in terms of services defined at any other layers.

**Globus Toolkit-the de facto standard for grid computing**

The Globus Toolkit [3] is fundamental enabling technology for the “grid”. It is a collaboration among Argonne National Laboratory, the University of Chicago, and Information Sciences Institute of the University of Southern California. Globus Toolkit is the soft-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



ware product developed by Globus project team to build computing Grid and Grid-based applications. Globus Toolkit has the following modules:

- Globus Security Infrastructure

  Grid Security Infrastructure (GSI) [64] enables secure authentication and communication over an open network. GSI provides a number of useful services for the computing grid, including mutual authentication and single sign-on. GSI is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol.

- Globus Resource Management System

  The Resource Management System (GRMS) [48, 113] is developed to support resource allocation and reservation in the computing grid. The GRMS architecture is a layered system in which a high-level global resource management services are layered on top of local resource allocation services.

  GRMS has three main components:

  - Resource Specification Language

    The Globus Resource Specification Language (RSL) provides a common interchange language to describe resources. The various components of the Globus Resource Management architecture manipulate RSL strings to perform their management functions in cooperation with the other components in the system. The RSL provides the skeletal syntax used to compose complicated resource descriptions, and the various resource management components introduce specific $< attribute, value >$ pairings into this common structure. Each attribute in a resource description serves as a parameter to control the behavior of one or more components in the resource management system.

  - Grid Resource Allocation Manager

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



The Globus Resource Allocation Manager (GRAM) processes the requests for resources for remote application execution, allocates the required resources, and manages the active jobs. It also returns updated information regarding the capabilities and availability of the computing resources to the Monitoring and Discovery Service. GRAM provides an API for submitting and cancelling a job request, as well as checking the status of a submitted job. The specifications are written by the user in the RSL, and is processed by GRAM as a part of the job request.

- Resource Broker/Co-allocator

  Resource brokers are responsible for taking high-level RSL specifications and transforming them into more concrete specifications through a process called *specialization*. Transformations effected by resource brokers generate a specification in which the locations of the required resources are completely specified. Such ground requests can be passed to a resource co-allocator. Resource co-allocator provides a co-allocation service, which coordinates ground requests that may span multiple GRAMs.

These three components of GRMS collaborate to realize the function of resource management of Globus. Applications submit resource requirements which are expressed in RSL. These resource requests are submitted to resource broker then transformed to ground specifications. Resource broker gets resource information from the grid information services. The ground specifications and resource information are processed and the GRAMs that manage required resources are specified. Resource co-allocator transfers single request to individual GRAM. With the collaboration of multiple GRAMs, resource co-allocator allocates resources to applications to fulfill their resource requirements.

- Globus Metacomputing Directory Service

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



The Globus Metacomputing Directory Service (MDS) [47] provides the necessary tools to build an LDAP[2.1] based information infrastructure for computational Grids. MDS uses the LDAP protocol as a uniform means of querying system information from a rich variety of system components, and for optionally constructing a uniform namespace for resource information across a system that may involve many organizations.

MDS has two components:

- GRIS - Grid Resource Information Service.

  GRIS provides a uniform means of querying resources on a computational Grid for their current configuration, capabilities, and status.

- GIIS - Grid Index Information Service GIIS provides a means of linking together arbitrary GRIS services to provide a coherent system image that can be explored or searched by Grid applications.

  A GIIS could pool information about all of the Grid resources in a particular research consortium, thus providing a coherent system image of that consortium's computing Grid. For example, a GIIS could list all of the computing resources available within a research lab or all of the distributed data storage systems owned by a particular agency.

- GridFTP

  GridFTP [21] is a high-performance, secured, reliable data transfer protocol optimized for high-bandwidth wide-area networks. It provides the following protocol features:

  - GSI security on control and data channels;
  - multiple data channels for parallel transfers;

[2.1] Lightweight Directory Access Protocol

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



- partial file transfers;
- third-party (direct server-to-server) transfers;
- authenticated data channels;
- reusable data channels; and
- command pipelining.

## 2.5 Parallel Program Design Environments

Parallel programming environments provide the basic tools, language features, and application programming interfaces (APIs) needed to construct a parallel program. A programming environment implies a particular abstraction of the computer system called a programming model. Traditional sequential computers use the well known von Neumann model. Because all sequential computers use this model, software designers can design software to a single abstraction and reasonably expect it to map onto most, if not all, sequential computers.

Unfortunately, there are many possible models for parallel computing, reflecting the different ways processors can be interconnected to construct a parallel system. The most common models are based on one of the parallel architectures mentioned before: shared-memory, distributed-memory with message passing, or a hybrid combination of the two.

Programming models aligned to a particular parallel system too closely will lead to programs that are not portable between parallel computers. Because the effective lifespan of software is longer than that of hardware, many organizations have more than one type of parallel computer, and most programmers insist on programming environments that allow them to write portable parallel programs. Also, explicitly managing large numbers of resources in a parallel computer is difficult, suggesting that higher-level abstractions of the parallel computer might be useful. The result is that as of the mid-1990s, there

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



was a veritable glut of parallel programming environments. This created a great deal of confusion for application developers and hindered the adoption of parallel computing for mainstream applications.

Fortunately, by the late 1990s, the parallel programming community converged predominantly on two environments for parallel programming: OpenMP [10] for shared-memory; MPI [6, 82] for message passing and PVM [72] for parallel virtual machines.

OpenMP is a set of language extensions implemented as compiler directives. Implementations are currently available for Fortran, C, and C++. OpenMP is frequently used to incrementally add parallelism to sequential code. By adding a compiler directive around a loop, for example, the compiler can be instructed to generate code to execute the iterations of the loop in parallel. The compiler takes care of most of the details of thread creation and management. OpenMP programs tend to work very well on SMPs, but because its underlying programming model does not include a notion of nonuniform memory access times, it is less ideal distributed-memory machines.

MPI is a set of library routines that provide for process management, message passing, and some collective communication operations (these are operations that involve all the processes involved in a program, such as the barrier, broadcast, and reduction). MPI programs can be difficult to write because the programmer is responsible for data distribution and explicit interprocess communication using messages. Because the programming model assumes distributed memory, MPI is a good choice for MPPs and other distributed-memory machines.

PVM is a software package that allows a heterogeneous network of computers (parallel, vector, or serial) to appear as a single concurrent computational resource–a virtual machine. Thus large computational problems can be solved effectively by using the aggregate power and memory of many computers. In this thesis, we prefer MPI over PVM because following reasons:

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



- MPI is formally specified and standard;
- MPI has full asynchronous communication;
- MPI can be used to efficiently program on clusters;
- MPI groups are solid, efficient, and deterministic;
- MPICH-G2 provides users the grid level communication support.

Neither OpenMP nor MPI is an ideal fit for hybrid architectures that combine multiprocessor nodes, each with multiple processes and a shared memory, into a larger system with separate address spaces for each node: The OpenMP model does not recognize nonuniform memory access times, so its data allocation can lead to poor performance on machines that are not SMPs, while MPI does not include constructs to manage data structures residing in a shared memory. One solution is a hybrid model in which OpenMP is used on each shared-memory node and MPI is used between the nodes. This works well, but it requires the programmer to work with two different programming models within a single program. Another option is to use MPI on both the shared-memory and distributed-memory portions of the algorithm and give up the advantages of a shared-memory programming model, even when the hardware directly supports it.

New high-level programming environments that simplify portable parallel programming and more accurately reflect the underlying parallel architectures are topics of current research. Another approach more popular in the commercial sector is to extend MPI and OpenMP. In the mid-1990s, the MPI Forum defined an extended MPI called MPI 2.0, although implementations are not widely available at the time this was written. It is a large complex extension to MPI that includes dynamic process creation, parallel I/O, and many other features. Of particular interest to programmers of modern hybrid architectures is the inclusion of one-sided communication. One-sided communication mimics

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



some of the features of a shared-memory system by letting one process write into or read from the memory regions of other processes. The term "one-sided" refers to the fact that the read or write is launched by the initiating process without the explicit involvement of the other participating process. A more sophisticated abstraction of one-sided communication is available as part of the Global Arrays [116, 115, 2] package. Global Arrays works together with MPI to help a programmer manage distributed array data. After the programmer defines the array and how it is laid out in memory, the program executes "puts" or "gets" into the array without needing to explicitly manage which MPI process "owns" the particular section of the array. In essence, the global array provides an abstraction of a globally shared array. This only works for arrays, but these are such common data structures in parallel computing that this package, although limited, can be very useful.

Just as MPI has been extended to mimic some of the benefits of a shared-memory environment, OpenMP has been extended to run in distributed-memory environments. The annual WOMPAT (Workshop on OpenMP Applications and Tools) workshops contain many papers discussing various approaches and experiences with OpenMP in clusters.

MPI is implemented as a library of routines to be called from programs written in a sequential programming language, whereas OpenMP is a set of extensions to sequential programming languages. They represent two of the possible categories of parallel programming environments (libraries and language extensions), and these two particular environments account for the overwhelming majority of parallel computing being done today. There is, however, one more category of parallel programming environments, namely languages with built-in features to support parallel programming. Java is such a language. Rather than being designed to support HPC, Java is an object-oriented, general-purpose programming environment with features for explicitly specifying concurrent processing with shared memory. In addition, the standard I/O and network packages

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



provide classes that make it easy for Java to perform interprocess communication between machines, thus making it possible to write programs based on both the shared-memory and the distributed-memory models. However, currently the performance of parallel Java programs cannot compete with OpenMP or MPI programs for typical scientific computing applications.

## 2.6 Parallel Program Design Methods

Although computing in less time is beneficial, and may enable problems to be solved that couldn't be otherwise, it comes at a cost. Writing programs to run on parallel computers can be much more difficult than producing an equivalent sequential implementation. Parallelism requires the programmer to consider a set of factors not present in sequential code. The complexity these features create propagates throughout the program and can seriously hinder the production, maintenance and portability of efficient parallel implementations. This daunting task usually falls on a small number of experts. For example, communication is via message passing, which introduces concurrency and possibly non-determinacy: in particular the deadlock and race conditions are all possible. Non-determinacy greatly confuses reasoning about program behaviour. The characteristics of the interconnection network-its latency and bandwidth-must also be considered. Failure to do so may cause processors that are waiting for a message to block excessively or the network to become saturated.

The key to parallel computing is exploitable concurrency. Concurrency exists in a computational problem when the problem can be decomposed into subproblems that can safely execute at the same time. To be of any use, however, it must be possible to structure the code to expose and later exploit the concurrency and permit the subproblems to actually run concurrently; that is, the concurrency must be exploitable. Most large computational problems contain exploitable concurrency. The programmer's task is to

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



identify the concurrency in the problem, structure the algorithm so that this concurrency can be exploited, and then implement the solution using a suitable programming environment.

Also, the concurrent tasks making up the problem include dependencies that must be identified and correctly managed. The order in which the tasks execute may change the answers of the computations in nondeterministic ways. A good parallel programmer must take care to ensure that nondeterministic issues do not affect the quality of the final answer. Creating safe parallel programs can take considerable effort from the programmer.

Even when a parallel program is "correct", it may fail to deliver the anticipated performance improvement from exploiting concurrency. Care must be taken to ensure that the overhead incurred by managing the concurrency does not overwhelm the program runtime. Also, partitioning the work among the processors in a balanced way is often not easy. The effectiveness of a parallel algorithm depends on how well it maps onto the underlying parallel computer.

In this section we survey two main approaches currently taken for parallel programming design. We start with the mainstream method, writing parallel programs using a sequential language combined with a communication library. After identifying the strengths and weaknesses of this technique, we examine the parallel pattern programming.

### 2.6.1 Explicit Parallel Programming

The most direct method for programming parallel applications is to view them as a collection of interacting sequential processors. Each processor is programmed using a conventional sequential language. Communication is achieved by calling message-passing

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



libraries. A level of portability can be achieved by using a standard communication API such as MPI [6] or PVM [72] which are available on many different platforms.

The popularity of this method is understandable: the programmer does not need to learn a new language; the programmer has direct and absolute control over the way an algorithm is expressed; there are few abstractions to introduce execution overhead; and there is extensive experience, support and libraries of sequential code written using this method.

However, it is not easy to gain a clear picture of the global behaviour of the machine from the program source. Sections where the computations of different processors differ are typically expressed using case statements or expressions over the processor rank. This allows the behaviour of the entire system to be represented within a single executable but the code is liable to get very tangled.

Related to the difficulty in understanding the behaviour of the parallel machine is ensuring that the processors communicate and synchronize correctly. Communicating a single message typically requires two library calls  a send and a reception. As these calls will occur in different branches of the program it is hard to ensure that the pairs of library calls match up as intended. Similarly, calls to initialize communication system objects, such as the communicators of MPI, must be executed by all processors, and each must pass identical parameters to the call. The sequential language provides no support for these constraints, and so deadlock and other hard-to-find errors are a common occurrence during development.

Furthermore, this method makes it hard to develop parallel programs incrementally. Exploration and experimentation are discouraged because they require extensive restructuring of the code. Rather, the programmer makes a set of arbitrary implementation decisions and then codes the program. It is also difficult to prove properties of programs written in this style because non-determinism is exposed by the communication model.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



In summary, explicit parallel programming provides fine control of all aspects of the parallel execution but exposes the programmer to great complexity. The code produced is brittle, hard to write, understand and maintain. Although this is one of the commonest techniques used to produce applications for parallel machines, its many limitations have led researchers to investigate other ways to express parallel computation.

### 2.6.2 Parallel Pattern Programming

#### Design Patterns

A design pattern describes a recurring design problem to be solved, a solution to the problem, and the context in which that solution works [40, 71]. The description specifies objects and classes that are customized to solve a general design problem in a particular context. A design pattern is a larger-grained form of software reuse than a class because it involves more than one class and the interconnection among objects from different classes. A design pattern is sometimes referred to as a micro-architecture.

After the original success of the design pattern concept, other kinds of patterns were developed. The main kinds of reusable patterns are:

- **Architectural patterns.** This work was described by Buschmann et al. [40] at Siemens. Architectural patterns are larger-grained than design patterns, addressing the structure of major subsystems of a system.
- **Analysis patterns.** Analysis patterns were described by Fowler [66], who found similarities during analysis of different application domains. He described recurring patterns found in object-oriented analysis and described them with static models, expressed in class diagrams.
- **Product line-specific patterns.** These are patterns used in specific application areas, such as factory automation [71] or electronic commerce.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



- **Idioms.** Idioms are low-level patterns specific to a programming language. For example, C++. These patterns are closest to code, but they can be used only by applications that are coded in the same programming language.

### Algorithmic skeletons

The concept of algorithmic skeleton has been presented in [44]. It has been followed by different research groups [97, 129, 20, 45]. The basic idea is to provide users by a set of either language constructs or library calls that completely take care of exploiting a given, recurring, parallel computation pattern [51]. Users need to supply specific parameters or codes, such as the sequential portions of code, to get a working parallel program.

### Parallel Patterns

It has been observed that explicitly parallel programs are made up of two different kinds of code: task specific code that implements the steps of the algorithm; and code for structuring the program into patterns of computation and communication for parallel execution. The second kind of code deals with the problematic aspects of parallel programming and handles the low-level details of the target machine. Although the code used to structure a parallel computation is complex, it often forms familiar patterns [121]. Parallel patterns are based on sequential program design patterns. The main idea is to separate the communication structure of a parallel program from the sequential application code. Thus, both parts of a parallel program can evolve independently. This allows for rapid prototyping of programs, and permits users to experiment with alternative communication structures quickly and easily.

Parallel patterns look like to be very close to the algorithmic skeleton idea, but for the different roots: the former being originated from Object-Oriented programming (OOP) area, the latter from parallel processing area. One of the most important contributions of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



parallel pattern technology lies in the nice exemplification of how layered programming environments can be designed, that both allow plain user to fully exploit design patterns in software design and implementation and more experienced users to intervene adding new patterns in the programming environment once those patterns have been understood to be useful and effective [105].

### Parallel Pattern-Based Systems

Parallel patterns have made a substantial impact on the mainstream practice in parallel programming [45]. Over the last two decades, several pattern-based systems have been built with the intention to facilitate the rapid development of parallel applications through the use of pre-implemented, reusable components. Some of the earlier systems include *Code* [37] and *Frameworks* [132]. Some of the recent systems based on similar ideas are *Enterprise* [125], *Code2* [38], *HeNCE* [29], *Tracs* [28], and $CO_2P_3S$ [36].

- *CODE* and *CODE2*: Designers of Computationally Oriented Display Environment (*CODE*) in University of Texas at Austin were early advocates of separation of specifications. In *CODE*, programmers develop programs in two steps. Firstly, developers specify the content of each node, i.e., sequential computation subroutines, input/output ports, internal variables and other rules that determine how the node is run. Secondly, they specify how these annotated nodes interact by composing them in a graph using a graphical interface. Then, *CODE* translates the graph into a complete parallel program. *CODE* is one of the first systems to enforce separation of specifications and to use a graphical interface to visualize process graphs. It has flexible rules for handling data flow in graphs, called firing rules that allow expression of a wide variety of parallel algorithms. However, data-flow elements and complex firing rules can be too low level and fine-grained when designing large and complex data-flow-based parallel programs.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



- Heterogeneous Network Computing Environment (*HeNCE*) was developed at University of Tennessee. It is similar in purpose and philosophy to *CODE*. It differs, however, in its implementation. It also is graphics user interface oriented. It uses separation of specifications where developers first specify the computation in each node and then specify their interconnection using a process graph. However, *HeNCE* graphs are control-flow oriented rather than the data-flow oriented graphs of *CODE*. *HeNCE* generates parallel programs, using PVM, based on these graphs. The basic building block in *HeNCE* is a node that contains sequential computation and input/output variable declarations. Design patterns, represented by *HeNCE*'s graphical icons, include higher-level primitives such as loop, replication, and pipeline. They can be used, with basic nodes, to build structures such as master/slave, pipeline, and for-all construct. Furthermore, similar to *CODE*, *HeNCE* allows recursive invocation of graphs. *HeNCE* has much simpler firing rules than *CODE*, and thus is easier to learn and use. However, experience shows that it might not be flexible enough to express more complex parallel algorithms.

- *FrameWorks* was specifically designed to restructure existing sequential programs to exploit parallelism on workstation clusters. It uses separation of specifications as well. Communication and synchronization of distributed processes are captured in design patterns. Developers insert sequential computation procedures into these patterns to create parallel programs. *FrameWorks* is an early system that successfully applies the design patterns to re-structure sequential applications for parallelism. It demonstrated that this technique can be used effectively, thus inspiring the next generation system, called *Enterprise*, which followed the same approach of using design patterns for parallel programming.

- *Enterprise*: Its goal is to let developers develop distributed applications quickly, economically and reliably. It is an integrated development environment complete

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



with tools such as a compiler, a debugger, graphical visualization tools, and a performance debugger. Like *FrameWorks*, it also uses separation of specifications. Developers use a meta-programming model resembling a business organization to express parallel structures like pipeline, master/slave, and divide and conquer. A design pattern in *Enterprise*, called asset, represents structures such as fan-out, pipeline or divide and conquer. Developers provide sequential computation procedures and annotate them with different assets. All necessary communication code for each process is automatically generated by *Enterprise*. Design patterns in *Enterprise* describe the behaviour of the whole process by combining all three types of patterns (input, output and body) used in *FrameWorks* into a single structure called an asset. Therefore, an asset contains process behaviour such as RPC scheduling and dynamic process replication. Unlike *FrameWorks*, it uses refinement whereby a node in a process graph can be recursively replaced by another sub-graph. Refinement only allows creation of tree-structured process graphs. These graphs are acyclic and thus can never deadlock a program. The disadvantage is that not all process graphs can be easily expressed using refinement. *Enterprise* is one of the very few non-commercial, integrated, design-pattern-based parallel programming environments. This research provided many insights into how tools such as debuggers and performance monitors can be used effectively in a design-pattern-based system.

- *Tracs* was developed at University di Pisa, provides an elegant graphical user interface for developing message-passing parallel programs. It uses separation of specifications similar to *FrameWorks* and *HeNCE*. The significant contribution of *Tracs* is its use of high-level design patterns. It raised the abstraction of design patterns from a single process to a collection of processes in its architecture model. However, its node specification is limited compared to other systems such as *FrameWorks* which

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



also allow a user to specify message scheduling. *Tracs* forces all design patterns to be graphical, causing difficulties in representing some patterns that cannot be conveniently represented graphically such as divide and conquer. *Tracs*'s graphical interface is elegant, but it can actually limit the expressiveness of the system in cases such as recursive structures where graphical representation of the pattern is not the most appropriate.

- The $CO_2P_3S$ (Correct Object-Oriented Pattern-Based Parallel Programming System) project uses a layered approach to parallel programming in an effort to address correctness and openness, two problems that continue to plague current parallel programming systems. Correctness ensures that, once created, a parallel program is structurally correct (i.e. that it contains all necessary communication and synchronization). Further, though, correctness in $CO_2P_3S$ ensures that this structure cannot be used incorrectly or accidentally modified until the performance tuning stages. In contrast, most current systems require the user to correctly implement and debug the desired structure. Openness means that the programming system provides opportunities for performance tuning and allows the user to take full advantage of all language facilities and run-time libraries to improve the performance of an application.

However, most of these systems lack practical usability for the CB field because the following reasons:

(1) Most systems only provide a limited set of parallel patterns, such as pipeline and task farm [45, 97]. These patterns can not meet the requirements of most CB applications.

(2) These systems produce code with disappointing parallel performance because they don't consider the characteristics of specific application domain.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(3) They are not flexible enough for the user to reuse the components of the systems at the application level. Also, they lack of extensibility for new appeared patterns.

(4) No computational grid oriented pattern-based systems have been developed. With the increased availability of grid computing platforms, grid-enabling of pattern-based systems are of high importance.

## 2.7 Summary

In this chapter, our research background is introduced and a survey of concepts and techniques is carried out accordingly. Section 2.2 presents the algorithm design techniques. Two popular genome analysis tasks are surveyed in Section 2.3. The parallel architecture-related concepts are reviewed in Section 2.4. Section 2.5 and 2.6 describe the parallel program design environments and methods. These surveys have motivated us to analyze the characteristics of popular sequential CB algorithms, design corresponding parallel algorithms, implement and evaluate them on different parallel architectures.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Chapter 3

# Characteristic Analyses of Sequential Computational Biology Algorithms

## 3.1 Introduction

With the advent of automated computing devices such as modern computers, an algorithm has in most contexts become synonymous with a description that can be turned into a computer program that instructs a computer how to solve the problem addressed by the algorithm. The ability of modern computers to perform billions of simple calculations per second and to store billions of bits of information, makes it possible by using the proper computer programs to address a wide range of problems that would otherwise remain out of reach. Such possibilities have spawned several interdisciplinary activities where the objective is to utilize the capacities of computers to gain knowledge from huge amounts of data. An important part of such activities is to construct good algorithms that can serve as the basis for computer programs that are needed to utilize the capacities of computers.

In very broad terms, we identify two high-level categories of CB problems: those that can be addressed analytically, and those that cannot. These two categories are

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



addressed in turn by two respective classes of algorithms: analytical solution approaches and heuristic approaches.

## 3.2 Dynamic Programming Algorithms

DP is a very important analytical solution method in CB. DP views a problem as a set of interdependent sub-problems. It solves sub-problems and uses the results to solve larger sub-problems until the entire problem is solved [99]. The solution to a sub-problem is expressed as a function of solutions to one or more sub-problems at the preceding levels. For example, a problem of size $n$ may decompose into several problems of size $n-1$, each of which decomposes into several problems of size $n-2$, etc. This decomposition seems to lead to an exponential-time algorithm, which is indeed true in the travelling salesman problem. In most other problems, however, there are only a polynomial number of distinct sub-problems. DP gains its efficiency by avoiding solving common sub-problems many times. It keeps track of the solutions of sub-problems in a table, and looks up the table whenever needed.

In general, the solution to a DP problem is expressed as a minimum (or maximum) of possible alternative solutions. Each of these alternative solutions is constructed by composing one or more sub-problems. If $r$ represents the cost of a solution composed of sub-problems $x_1$, $x_2$,...,$x_l$, then $r$ can be written as:

$$r = g(f(x_1), f(x_2), \dots, f(x_l)) \tag{3.2.1}$$

The function $g()$ in Eq. 3.2.1 is called the composition function, and its nature depends on the problem described. If the optimal solution to each problem is determined by composing optimal solutions to the sub-problems and selecting the minimum (or maximum), Eq. 3.2.1 is then said to be a DP formulation [99]. Figure 3.1 illustrates an

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



instance of composition and minimization of solutions. The solution to problem $x_8$ is the minimum of the three possible solutions having costs $r_1$, $r_2$, and $r_3$. The cost of the first solution is determined by composing solutions to sub-problems $x_1$ and $x_3$, the second solution by composing solutions to sub-problems $x_4$ and $x_5$, and the third solution by composing solutions to sub-problems $x_2$, $x_6$, and $x_7$.

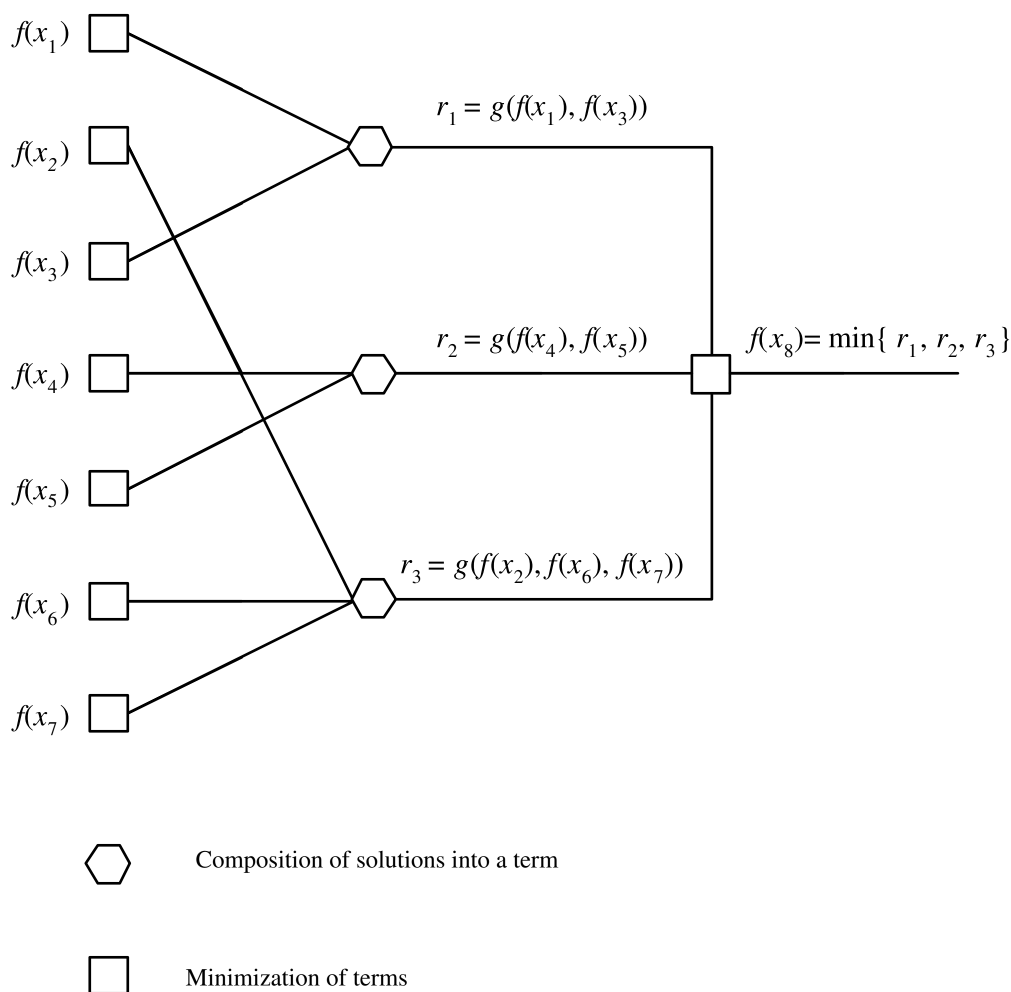



Figure 3.1: The computation and composition of sub-problem solutions to solve problem $f(x_8)$

According to [70], DP algorithms have three features in common:

(1) a table

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(2) the entry dependency of the table

(3) the order to fill in the table

Each entry of the table corresponds to a sub-problem. Thus the size of the table is the total number of sub-problems including the problem itself. The entry dependency is defined by the decomposition: if a problem $P$ decomposes into several sub-problems $P_1$, $P_2 \ldots P_k$, the table entry of the problem P depends on the table entries of $P_1$, $P_2 \ldots P_k$. The order to fill in the table may be chosen under the restriction of the table and the entry dependency. The DP formulation of a problem always provides an obvious algorithm which fills in the table according to the entry dependency.

### 3.2.1 Characteristic Analysis of Dynamic Programming Algorithms

DP algorithms can be classified according to the matrix size and the dependency relationship of each cell on the matrix [70]: a DP algorithm is called a $tD/eD$ algorithm if its matrix size is $n^t$ and each matrix cell depends on $O(n^e)$ other cells. The DP formulation of a problem always yields an obvious algorithm whose time complexity is determined by the matrix size and the dependency relationship. If a DP algorithm is a $tD/eD$ problem, it takes time $O(n^{t+e})$ provided that the computation of each term takes constant time. Four examples are given in Algorithms 1 to 4.

**Algorithm 1** ($1D/1D$): Given a real-valued function $w(i, j)$ for integers $0 \leq i < j \leq n$ and $D[0]$

$$D[j] = \min_{0 \leq i < j} \{D[i] + w(j, j)\} for\ 1 \leq j \leq n \tag{3.2.2}$$

**Algorithm 2** ($2D/0D$): Given $D[i, 0]$ and $D[0, j]$ for $1 \leq i, j \leq n$

$$D[i, j] = \min\{D[i-1, j] + x_i, D[i, j-1] + y_j, D[i-1, j-1] + z_{ij}\} \tag{3.2.3}$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



where $x_i$, $y_j$ and $z_{ij}$ are computed in constant time.

**Algorithm 3** ($2D/1D$): Given $w(i, j)$ for $1 \leq i < j \leq n$; $D[i, i] = 0$ for $1 \leq i \leq n$

$$D[i, j] = w(i, j) + \min_{i<k\leq j}\{D[i, k-1] + D[k, j]\} \tag{3.2.4}$$

where $w(i, j)$ is computed in constant time.

**Algorithm 4** ($2D/2D$): Given $w(i, j)$ for $1 \leq i < j \leq 2n$, $D[i, 0]$ and $D[0, j]$ for $0 \leq i, j \leq n$

$$D[i, j] = \min_{\substack{0\leq i'<i \\ 0\leq j'<j}}\{D[i', j'] + w(i' + j', i + j)\} for\ 1 \leq i, j \leq n \tag{3.2.5}$$

The DP formulation of a problem always yields an obvious algorithm whose efficiency is determined by the table size and entry dependency. If a DP problem is a $tD/eD$ problem, an obvious algorithm takes time $O(n^{t+e})$ provided that the computation of each term (e.g., $D[i] + w(j, j)$ in Algorithm 1) takes constant time. The space required is usually $O(n^t)$.

There are many DP algorithms in CB. DP is used for assembling DNA sequence data from the fragments that are delivered by automated sequencing machines [25], and to determine the intron/exon structure of eukaryotic genes [74]. It is used to infer function of proteins by homology to other proteins with known function [114, 133] and it is used to predict the secondary structure of functional RNA genes or regulatory elements [152]. A recent textbook [55] presents a dozen variants of DP algorithms in its introductory chapter on sequence comparison. In some areas of CB, DP problems arise in such variety that a specific code generation system for implementing such algorithms has been developed [32]. However, the development of a successful parallel DP algorithm is a matter of experience, talent, and luck. The typical matrix recurrence relations that make up a parallel DP algorithm are intricate to construct, and difficult to implement reliably. No general problem independent guidance is available. According to above classification method, we can classify the popular DP algorithms in CB as shown in Table 3.1.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Table 3.1: A classification for the popular DP algorithms in CB

| **Algorithms** | **Time Complexity** | **Applications** | **References** |
|---|---|---|---|
| Smith-Waterman algorithm with linear and affine gap penalty | $O(n^2)$ | Genome alignment | [90, 134] |
| Syntenic alignment | | Generalized genome global alignment | |
| Smith-Waterman algorithm with with general gap penalty | $O(n^3)$ | Genome alignment | [55, 134] |
| Nussinov algorithm | | RNA base pair maximization | |
| Viterbi algorithm | $O(n^2) - O(n^4)$ | Gene sequence alignment using HMMs, Multiple sequence alignment | [55] |
| Double DP algorithm | $O(n^4)$ | Protein threading | [110] |
| Spliced alignment | $O(n^3)$ | Gene finding | [73] |
| Zuker algorithm | $O(n^3) - O(n^4)$ | RNA secondary structure prediction | [153] |
| CYK algorithm | $O(n^3) - O(n^4)$ | RNA secondary structure alignment | [55] |

Due to the wide variety of problems solved using DP, it is difficult to develop generic parallel algorithm for them. However, parallel formulations of the problems in each of the four DP categories have certain similarities. We will discuss parallel formulations for sample problems in each class. These samples suggest parallel algorithms for other problems in the same class.

DP applications usually exhibit the characteristic of the *wavefront* computation, that is, each element computes a value that depends on the computation of a set of previous elements. An example is shown in Figure 3.2. Figure 3.2b displays the dependency relationship: each matrix element $(i, j)$ is computed from the matrix cells $(i-1, j)$, $(i, j-1)$, $(i-1, j-1)$. The wavefront moves in anti-diagonals as depicted in Figure 3.2a, that is, the shift direction is from northwest to southeast. Depending on the dependency

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



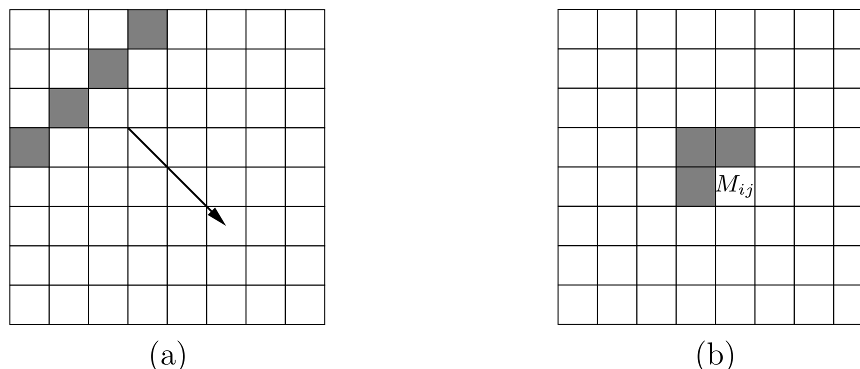



Figure 3.2: Example of a wavefront computation: (a) shift direction, (b) dependency relationship

relationship different wavefront shift directions are possible. In this section we discuss four examples of DP algorithms with different dependency relationships.

**2D/0D Dynamic Programming Algorithms**

- Smith-Waterman Algorithm

Sequence alignments are fundamental to many applications in CB, and comprise one of the best studied and well understood problem areas in this discipline. Much of the early pioneering work concentrated on two types of alignments: 1) global alignments, which are intended for comparing two sequences that are entirely similar [80, 112, 114], and 2) local alignments, which are intended for comparing sequences that have locally similar regions [89, 134].

The local alignment algorithm was developed in the early 1980's. It is frequently known as the Smith-Waterman algorithm. Given are two sequences $A$ and $B$ of length $m$ and $n$, respectively, a substitution matrix $s$ and a linear gap penalty $d$. The following recursion relation then computes the optimal local alignment of $A$ and $B$:

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(A_i, B_j), \\ M(i-1,j) - d, \\ M(i,j-1) - d, \\ 0; \end{cases} \tag{3.2.6}$$

The recursions is calculated with $i$ going from 1 to $m$ and $j$ from 1 to $n$, starting with $M(i,j) = 0$ for all $i = 0$ or $j = 0$. The order of computation of the values in the alignment matrix is strict because the value of any cell cannot be computed before the value of all cells to the left and above it has been computed. The wavefront computation procedure of Smith-Waterman algorithm is shown in Figure 3.2.

The Smith-Waterman algorithm with linear gap penalty is not ideal for biological sequences: it penalizes additional gap steps as much as the first, whereas, when gaps do occur, they are often longer than one residue. If we are given a general function for $\gamma(g)$ then we can still use the DP versions described in Eq. 3.2.6, with adjustments to the recurrence relations as typified by the following:

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(A_i, B_j), \\ M(i-1,j) + \gamma(i-k), & k = 0, \ldots, i-1 \\ M(i,j-1) + \gamma(j-k), & k = 0, \ldots, j-1 \\ 0; \end{cases} \tag{3.2.7}$$

which gives a replacement for the basic local dynamic relation. However, this procedure now requires $O(n^3)$ operations to align two sequences of length n, rather than $O(n^2)$ for the linear gap cost version, because in each cell $(i,j)$ we have to look at $i+j+1$ potential precursors, not just three as previously. This is a prohibitively costly increase in computational time in many cases. Under some conditions on the properties of $\gamma()$ the search in $k$ can be bounded, returning the expected computational time to $O(n^2)$, although the constant of proportionality is higher in these cases [112].

The standard alternative to using 3.2.7 is to assume an affine gap cost structure as: $\gamma(g) = -d - (g-1)e$. For this form of gap cost there is once again an $O(n^2)$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



implementation of DP. However, we now have to keep track of multiple values for each pair of residue coefficients $(i, j)$ in place of the single value $M(i, j)$. The recurrence relations corresponding to 3.2.7 now become:

$$M(i,j) = \max \begin{cases} M(i-1, j-1) + s(A_i, B_j), \\ I_x(i-1, j-1) + s(A_i, B_j), \\ I_y(i-1, j-1) + s(A_i, B_j), \\ 0; \end{cases} \tag{3.2.8}$$

$$I_x(i,j) = \max \begin{cases} M(i-1, j) - d, \\ I_x(i-1, j) - e; \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i, j-1) - d, \\ I_y(i, j-1) - e; \end{cases}$$

In these equations, we assume that a deletion will not be followed directly by an insertion. This will be true for the optimal path if $-d-e$ is less than the lowest mismatch score.

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

Figure 3.3: Example of the Smith-Waterman algorithm to compute the local alignment between two sequences ATCTCGTATGATG and GTCTATCAC.

Figure 3.3 illustrates an example to compute the local alignment between two sequences $S1$ = ATCTCGTATGATG and $S2$ = GTCTATCAC. The matrix $H(i, j)$ is shown for the computation with $gap(k) = 0 + 1 \times k$ and a substitution cost of +2 if the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



characters are identical and −1 otherwise. From the highest score (+10 in the example), a traceback procedure delivers the corresponding alignment (shaded elements), the two subsequences TCGTATGA and TCTATCA. Since the score (+10) is the highest one in the similarity matrix, the corresponding alignment is the optimal alignment.

- Syntenic Alignment Algorithm

It is widely recognized that evolutionary processes tend to conserve genes. Along a chromosome, genes are interspersed by large regions known as "junk DNA". A gene itself is comprised of alternating regions known as *exons* and *introns*, and the introns are intervening regions that do not participate in the translation of a gene to its corresponding protein. Homologous DNA sequences from related organisms, such as the human and the mouse, are usually similar over the exon regions but different over other regions. Because the different regions are much longer than similar regions, conserved sequences cannot be identified through global alignment. This results in the problem of aligning two sequences where an ordered list of subsequences of one sequence is highly similar to a corresponding ordered list of subsequences from the other sequence. We refer to this problem as the *syntenic alignment* problem [68].

Let $A = a_1a_2 \ldots a_m$ and $B = b_1b_2 \ldots b_n$ be two sequences. A subsequence $A'$ of $A$ is said to precede another subsequence $A''$ of $A$, written $A' \prec A''$, if the last character of $A'$ occurs strictly before the first character of $A''$ in $A$. An ordered list of subsequences of $A$, $(A_1, A_2, \ldots, A_k)$ is called a chain if $A_1 \prec A_2 \prec \ldots A_k$ of subsequences in $A$ and a chain $(B_1, B_2, \ldots, B_k)$ of subsequences in $B$ such that the score:

$$\left\{\sum_{i=1}^{k} score(A_i, B_i)\right\} - (k-1)d$$

is maximized (see figure 3.4. The parameter $d$ is a large penalty aimed at preventing alignment of short subsequences which occur by chance and not because of any biological

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



significance).

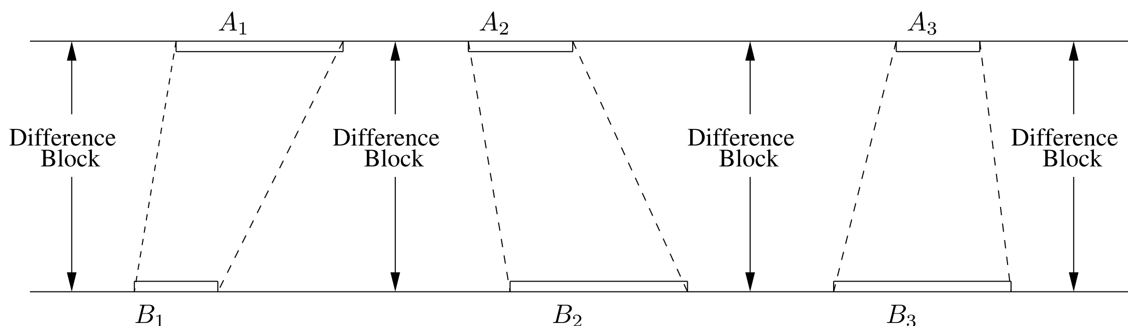



Figure 3.4: The syntenic alignment is an ordered list of local alignments separated by difference blocks

$$M(i,j) = s(A_i,B_j) + \max\begin{cases} E(i-1,j-1), \\ F(i-1,j-1), \\ H(i-1,j-1); \end{cases} \tag{3.2.9}$$

$$E(i,j) = \max\begin{cases} M(i-1,j)-g', \\ E(i-1,j)-g, \\ F(i-1,j)-g', \\ H(i-1,j)-g'; \end{cases}$$

$$F(i,j) = \max\begin{cases} M(i,j-1)-g', \\ E(i,j-1)-g', \\ F(i,j-1)-g, \\ H(i,j-1)-g'; \end{cases}$$

$$H(i,j) = \max\begin{cases} M(i-1,j)-D, \\ F(i-1,j)-D, \\ M(i,j-1)-D, \\ E(i,j-1)-D, \\ H(i-1,j), \\ H(i,j-1); \end{cases}$$

Based on the problem definition, the syntenic alignment of two sequences $A = a_1a_2\ldots a_m$ and $B = b_1b_2\ldots b_n$ can be computed by DP. Consider two strings $A$ and $B$ of length $l_1$ and $l_2$, a substitution matrix $s$ and a linear gap penalty $d$, $g' = g + h$, $g$ is the gap-open penalty and $h$ is the gap-extension penalty, $D$ is a constant penalty for each different block. To identify common subsequences, they compute the similarity matrix $M(i,j)$ of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



two sequences ending at position $i$ and $j$. It follows from these definitions that $M(i, j)$ can be computed using the recurrence equations in Eq. 3.2.9.

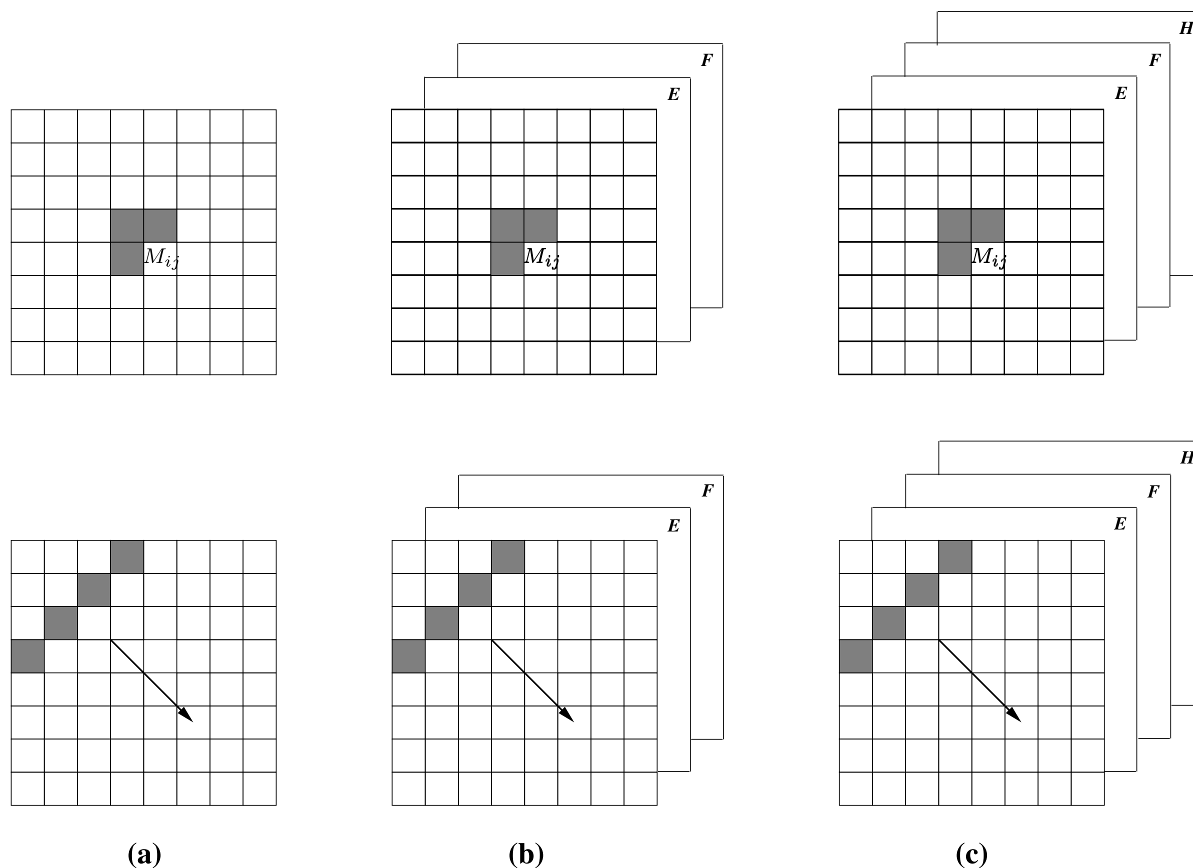



Figure 3.5: (a) Dependency relationship and wavefront shift direction of SW with the linear gap penalty (b) Dependency relationship and wavefront shift direction of SW with the affine gap penalty (c) Dependency relationship and computation shift direction of syntenic alignment algorithm

Figure 3.5 shows the dependency relationship and the computation shift direction of Smith-Waterman (with linear and affine gap penalty) and Syntenic alignment algorithms. We can find that the obvious differences between them are the number of matrices they will compute. All of these algorithms have an even workload across matrix cells, i.e. each matrix cell is computed from the same number of other matrix cells. We call such DP algorithms *regular*. In practice, the regular DP computations share these similar

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



characters.

**2D/1D Dynamic Programming Algorithms**

- Skyline Matrix Problem

The skyline matrix problem is one of the "Cowichan-Problems". The Cowichan problem set presented in [151] has been carefully selected to provide a suitable handle for assessing the usability of parallel programming systems.

An $N \times N$ matrix $A$ is known as a skyline matrix when each sub diagonal row $a_{i1} \ldots a_{ii}$ and each supra diagonal column $a_{1j} \ldots a_{jj}$ of $A$, $1 \leq i, j \leq N$, has a (possibly zero-length) prefix of zero-valued elements. More exactly, a skyline matrix is one for which there exist constants $r_i$ and $c_j$, $1 \leq i, j \leq N$, such that:

1. $1 \leq r_i \leq i$, row $i$ has non-zero values in columns $r_i$ to $i$;
2. $1 \leq c_j \leq j$, column $j$ has non-zero values in rows $c_j$ to $j$.

| $r_i$ \ $c_j$ | 1 | 2 | 3 | 4 | 3 | 2 | 6 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 17 | | | | | | | |
| 2 | | 22 | | | | 63 | | |
| 3 | | | 21 | | 3 | 58 | | |
| 3 | | | 19 | 15 | 33 | 21 | | |
| 3 | | | 11 | 98 | 49 | 27 | | |
| 2 | | 14 | 78 | 91 | 87 | 56 | 18 | |
| 5 | | | | | 25 | 33 | 53 | |
| 6 | | | | | | 34 | 76 | 37 |

Figure 3.6: An example $8 \times 8$ skyline matrix

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



A typical example of a skyline matrix is shown in Figure 3.6, together with the corresponding $r_i$ and $c_j$ values. These two vectors may be regarded as describing the skyline envelope of the non-zero values in the matrix.

The skyline matrix problem can be formulated as follows: Given an $N \times N$ skyline matrix $A$ and an $N$-vector $b$, we seek to find an $N$-vector $x$ such that $Ax = b$. An efficient and widely used technique for solving $Ax = b$ in the general case is the $LU$-Decomposition. This method decomposes $A$ into two matrices $L$ and $U$. The algorithm used for sequential $LU$-Decomposition is "Doolittle's Method". Generally, the algorithm works as follows:

**for** $i = 1$ to $N$ **do**

  **for** $j = 1$ to $i-1$ **do**

    $L_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} L_{ik}U_{kj}\right) / U_{jj}$

  **for** $j = 1$ to $i$ **do**

    $U_{ji} = a_{ji} - \sum_{k=1}^{j-1} L_{jk}U_{ki}$

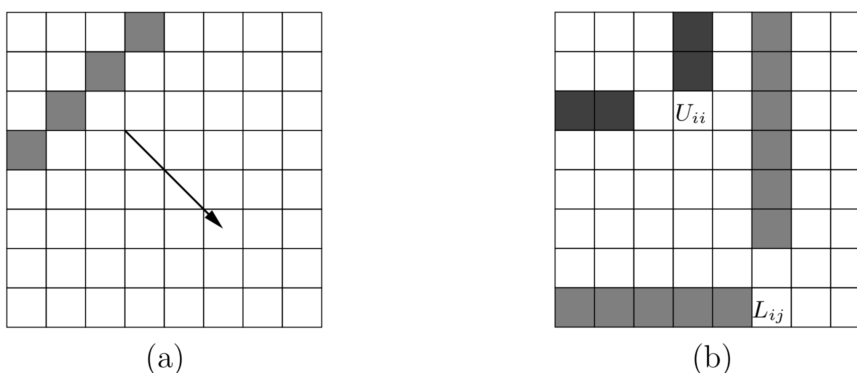



Figure 3.7: The wavefront computation of skyline matrix problem: (a) shift direction, (b) dependency relationship

The algorithm generates the dependency relationship shown in Figure 3.7. The calculation of each element $U_{ij}$ requires only the externally provided row $L_i$ and local prefix-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



column $U_j$ just up to $U_{ij}$, and the calculation of element $L_{ji}$ requires only the externally provided column $U_i$ and local prefix-row $L_j$ just up to $L_{ji}$.

- Nussinov and Matrix Chain Ordering Algorithm

Structure prediction is important for RNA. RNA molecules perform certain catalytic functions, which are the consequence of their specific three-dimensional structure. This structure is encoded in the nucleotide sequence of the RNA molecule. While RNA tertiary structure prediction appears as difficult as protein structure prediction, there are DP algorithms for the simpler problem of RNA secondary structure prediction.

Suppose we wish to predict the secondary structure of a single RNA. Many plausible secondary structures can be drawn for a sequence. The number increases exponentially with sequence length. An RNA with 200 bases long has over $10^{50}$ possible base-paired structures. We must distinguish the biologically correct structure from all the incorrect structures. We need both a function that assigns the correct structure the highest score, and an algorithm for evaluation the scores of all possible structures [55].

Initialization:

**for** $i = 2$ to $L$ **do**

$M(i, i-1) = 0$

**for** $i = 1$ to $L$ **do**

$M(i, i) = 0$

Recursion:

$$M(i,j) = max \begin{cases} M(i+1,j), \\ M(i,j-1), \\ M(i+1,j-1) + \delta(i,j), \\ \max_{1<k<j}[M(i,k) + M(k+1,j)]. \end{cases}$$

The value of $M(1, L)$ is the number of base pairs in the maximally base-paired structure.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



The Nussinov algorithm is an efficient DP algorithm to find the RNA structure with the most base pairs. We are given a sequence $A$ of length $L$ with symbols $x_1 \dots x_L$. Let $(i, j) = 1$ if $x_i$ and $x_j$ are a complementary base pair, else $(i, j) = 0$. We will recursively calculate scores $M(i, j)$ which are the maximal number of base pairs that can be formed for subsequence $x_i, \dots, x_j$. Formally, the Nussinov algorithm is as above.

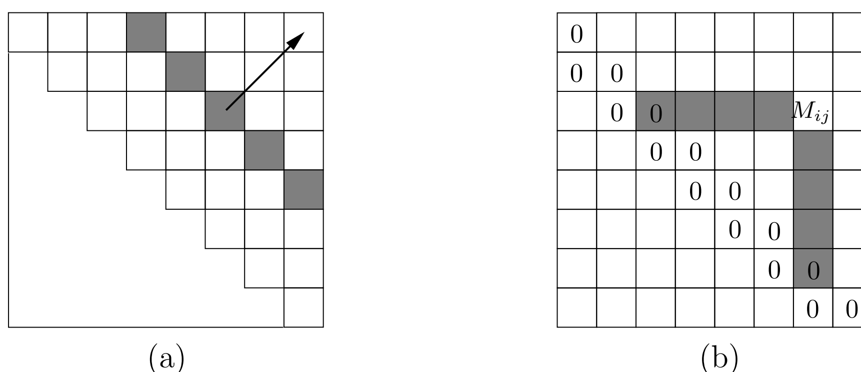



Figure 3.8: The wavefront computation of Nussinov algorithm: (a) shift direction; (b) dependency relationship

The wavefront computation procedure of Nussinov algorithm is shown in Figure 3.8. We can see that each matrix cell $(i, j)$ is computed from all matrix cells in row $i$ and column $j$ from index $min(i, j)$ up to $max(i, j)$.

The *Matrix Chain Ordering Problem* (MCOP) is very similar to the Nussinov algorithm. It can be expressed as a parenthesization of the $n$ given matrices giving an order to optimally multiply them. There are a Catalan number of ways to parenthesize any n element associative product, so a brute-force method of exhaustive search algorithm is not feasible.

Taking a chain of $n$ matrices $M_1 \times M_2 \times \cdots \times M_n$, then there are possible sub-products of the form $MC_{i,j} = M_i \times \cdots \times M_j$. Clearly, the final product $M_{1,n}$ must be made up of one of these sub-products. These sub-products, in turn, are made up of such sub-products with the single matrix base case when $i = j$.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Suppose an optimal parenthesization of $M_1 \times M_2 \times \cdots \times M_n$ splits the product between $M_k$ and $M_{k+1}$ for some $k$. Then, the prefix sub-chain $M_1 \times M_2 \times \cdots \times M_k$ within this optimal parenthesization must be optimal. The same is for $M_{k+1} \times M_{k+2} \times \cdots \times M_n$. This principle of optimality is the characteristic of DP algorithms. So, given that the matrix $MC_{i,j}$ is of dimensions $d_i \times d_{j+1}$, and taking $MC_{i,i} = 0$ as a base case, we can get the following pseudo code for solving the MCOP:

Initialization:

**for** $i = 1$ to $L$ **do**

$MC_{i,i} = 0$

Recursion:

**for** each diagonal $D$ from 2 to $n$ **do**

**for** each element $MC_{i,k}$ in diagonal $D$ **do**

$$MC_{i,k} = Min_{i \leq j \leq k}[MC_{i,j} + MC_{j+1,k} + d_i d_{j+1} d_{k+1}]$$

where $MC_{i,k}$ is the cost of computing $M_i \times M_{i+1} \times \cdots \times M_k$.

In general, using this algorithm to find the minimum for an element in diagonal $D$, it uses elements from all $D - 1$ previous diagonals. Given $n$ matrices to order, such inter-diagonal dependency relationship is same with the one shown in Figure 3.8.

**2D/2D Dynamic Programming Algorithms**

- Arbitrary-Order Viterbi Algorithm

*Hidden Markov Models* (HMMs) have been the mainstay of the statistical modeling used in modern speech recognition systems and biological sequence analysis systems [96, 122]. The *Viterbi Algorithm* is often used to find the state sequence that best explains the observation, that is, to find the single best state sequence $Q = q_1, q_2 \ldots q_r$, for the given observation sequence $O = O_1, O_2 \ldots O_r$. Here, we need to define the quantity

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



$$\delta_t(i) = \max_{q_1, q_2 \dots q_{t-1}} p[q_1, q_2 \dots q_t = i, O_1 O_2 \dots O_t | \lambda]$$

$\delta_t(i)$ is the best score (highest probability) along a single path, at time $t$, which accounts for the first $t$ observations and ends in state $S_i$, $\lambda$ is the complete parameter set of the HMM. By induction we have

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] b_j(O_{t+1})$$

where $a_{ij}$ is the state transition probability distribution, $b_j(k)$ is the observation symbol probability distribution.

First-order and second-order Viterbi algorithms are generally used in practice. In the first-order algorithm, elements in the matrix depend on all the elements in the previous row. In the second-order Viterbi algorithm elements depend on all the elements in the previous two rows.

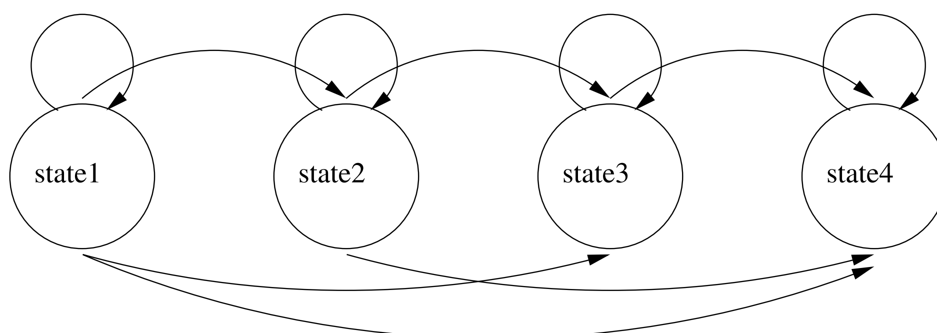



Figure 3.9: An example of four states Pre-Fix HMM

Although the first and second-order Viterbi algorithms are frequently used, they are not sufficient for some application areas such as biological sequence analysis where a position in a gene sequence not merely depends on the previous two positions. Here, we design a new kind of HMM. In this HMM, each state will depend on all the previous

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



states. We call this a *Pre-Fix* HMM (see Figure 3.9).

The corresponding modified Viterbi algorithm that we called arbitrary order Viterbi algorithm looks as follows:

Initialization:

**for** $i = 1$ to $N$ **do**

$\delta_1(i) = \delta_i b_i(O_1)$

Recursion:

**for** $t = 2$ to $T$ **do**

**for** $k = 1$ to $t$ **do**

$\delta_t(j) = Max_{1 \leq i \leq j}[\delta_{t-k}(i) a_{ij}] b_j(O_t)$

where $N$ is the number of states and $T$ is the number of observations.

The dependency relationship for the algorithm is shown in Figure 3.10.

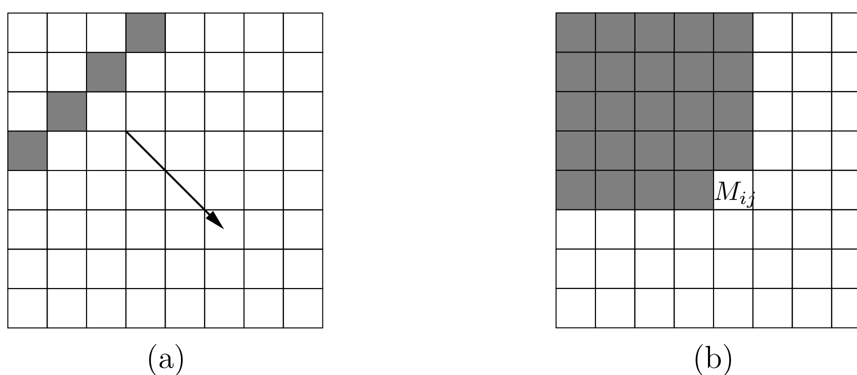



Figure 3.10: The wavefront computation of arbitrary-order Viterbi algorithm: (a) shift direction, (b) dependency relationship

From dependency relationship figures (see Figure 3.7, 3.8, and 3.10) of 2D/1D and 2D/2D DP algorithms we can see that the load to compute one cell in the matrix will increase along the shift direction of the wavefront. We call this kind of DP algorithms

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



*irregular.*

**3D/0D Dynamic Programming Algorithms**

- Spliced Alignment Algorithm

Many gene sequence alignment algorithms are multi-dimensional algorithms and the dependency relationships are more complex. The spliced aliment algorithm is one example of such kind of algorithms.

Spliced alignment belongs to the problem class of automatized gene-finding. The structure of a typical nuclear eukaryotic gene is a mosaic of exons and introns. A so-called cDNA (complementary DNA) is a DNA sequence which is reverse transcribed from RNA into DNA; it is the DNA counterpart of a mature RNA, i.e. the exons of a gene glued together, omitting the intervening intron sequences. These artificial pieces of DNA can be established in a laboratory protocol from mRNA isolated from cell tissue. Given such a cDNA, the problem is to localize it in the genomic DNA (see Figure 3.11). This is different from a simple matching problem, since the goal is to derive the correct splice pattern.

The formal statement of spliced alignment is as follows:

Let $G = g_1 \dots g_n$ be a string, and let $B = g_i \dots g_j$ and $B' = g_{i'} \dots g_{j'}$ are substrings of $G$. We write $B < B'$ if $j < i'$, i.e., if $B$ ends before $B'$ starts. A sequence $\Gamma = (B_1, \dots, B_p)$ of substrings of $G$ is a chain if $B_1 < B_2 < \dots < B_p$. $\Gamma^* = B_1 * B_2 * \dots * B_p$ is a concatenation of strings from the chain $\Gamma$. $s(G, T)$ denotes the score of the optimal alignment between strings $G$ and $T$.

Let $G = g_1 \dots g_n$ be a string called genomic sequence, $T = t_1 \dots t_m$ be a string called target sequence, and $B = B_1, \dots, B_p$ be a set of substrings of $G$ called blocks. Given $G$, $T$, and $B$, the spliced alignment problem is to find a chain $\Gamma$ of strings from $B$ such that

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



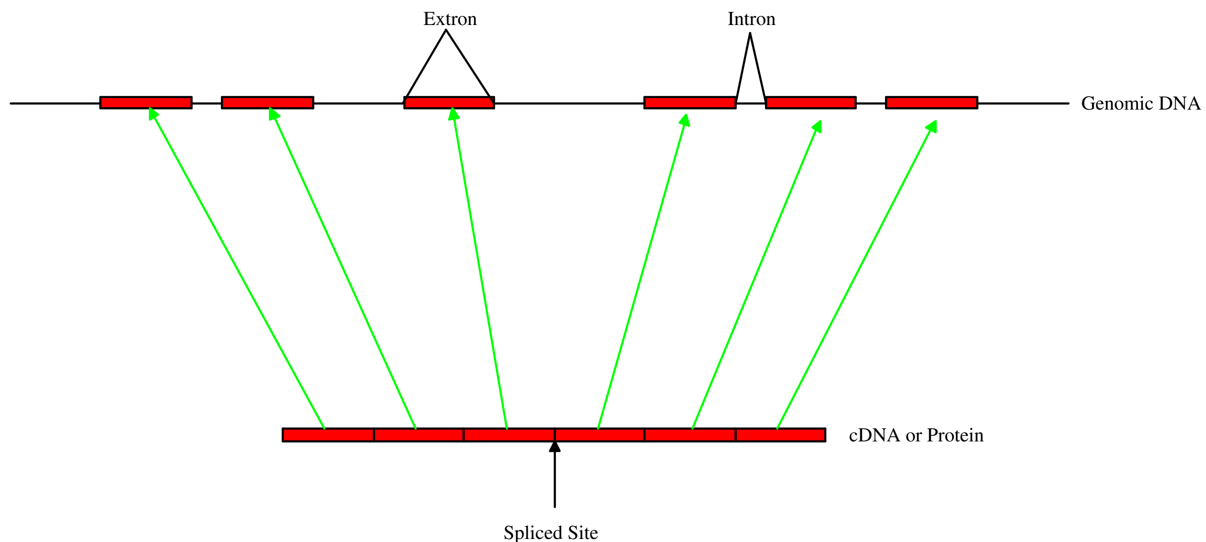



Figure 3.11: The principle of spliced alignments

the score $s(\Gamma^*, T)$ of the alignment between the concatenation of these strings and the target sequence is maximum among all chains of blocks from $B$.

Gelfand et al. [73] reduced the exon assembly problem to the search of a path in a directed graph. Let $B_k = g_m \ldots g_i \ldots g_l$ be a substring of $G$ containing a position $i$. Define $i - prefix$ of $B_k$ as $B_k(i) = g_m \ldots g_i$. For a block $B_k = g_m \ldots g_l$, let $first(k) = m$, $last(k) = l$, $size(k) = l - m + 1$. Let $B(i) = \{k : last(k) < i\}$ be the set of blocks ending before position $i$ in $G$. Let $G = (B_1, \ldots, B_k, \ldots, B_t)$ be a chain and some block $B_k$ contains position $i$. Define $\Gamma^*(i)$ as a string $\Gamma^*(i) = B_1 * B_2 * \cdots * B_k(i)$. Let

$$S(i,j,k) = \max_{all\ chains\ \Gamma\ containing\ block\ B_k} s(\Gamma^*(i), T(j)) \tag{3.2.10}$$

The following recurrence computes $S(i, j, k)$ for $1 \leq i \leq n, 1 \leq j \leq m, and\ 1 \leq k \leq b$. For the sake of simplicity we consider sequence alignment with linear gap penalties and define $\delta(x, y)$ as a similarity score for every pair of amino acids $x$ and $y$ and $\delta_{indel}$ as a penalty for insertion or deletion of amino acids.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



$$S(i,j,k) = \max \begin{cases} S(i-1,j-1,k) + \delta(g_i,t_j), & if\ i \neq first(k) \\ S(i-1,j,k) + \delta_{indel}, & if\ i \neq first(k) \\ \max_{l \in B(first(k))} S(last(l),j-1,l) + \delta(g_i,t_j), & if\ i = first(k) \\ \max_{l \in B(first(k))} S(last(l),j,l) + \delta_{indel}, & if\ i = first(k) \\ S(i,j-1,k) + \delta_{indel} \end{cases} \tag{3.2.11}$$

After computing the 3-dimensional table $S(i,j,k)$, the score of the optimal spliced alignment is

$$\max_k S(last(k),m,k) \tag{3.2.12}$$

Gelfand reduced the number of edges in the spliced alignment graph by making equivalent transformations of the described network, leading to a reduction in time and space. Define

$$P(i,j) = \max_{l \in B(i)} S(last(l),j,l) \tag{3.2.13}$$

Then 3.2.11 can be rewritten as

$$S(i,j,k) = \max \begin{cases} S(i-1,j-1,k) + \delta(g_i,t_j), & if\ i \neq first(k) \\ S(i-1,j,k) + \delta_{indel}, & if\ i \neq first(k) \\ P(first(k),j-1) + \delta(g_i,t_j), & if\ i = first(k) \\ P(first(k),j) + \delta_{indel}, & if\ i = first(k) \\ S(i,j-1,k) + \delta_{indel} \end{cases} \tag{3.2.14}$$

where

$$P(i,j) = \max \begin{cases} P(i-1,j) \\ \max_{k:last(k)=i-1} S(i-1,j,k) \end{cases} \tag{3.2.15}$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



The network corresponding to 3.2.14 and 3.2.15 has a significantly smaller number of edges, thus leading to a practical implementation of the spliced alignment algorithm.

Note that $S(i, j, k)$ is defined only if $i \in B_k$ and therefore only a portion of entries in the three-dimensional nmb matrix $S$ needs to be computed. The time complexity of the algorithm is $O(mnc + mb)$, where c = $\frac{1}{n}\sum_{k=1}^{b} size(k)$ is the coverage of the genomic sequence by blocks.

### 3.2.2 Space-Saving Algorithm

Alignment algorithms can be used to report all possible alignments between two sequences. However, their quadratic space complexities have made them unattractive for applications involving very long sequences. For instance, considering comparing two sequences with length of $l_1$ and $l_2$, the memory and time complexity for Smith-Waterman algorithm is $O(l_1 \times l_2)$. For aligning two sequences of a few million base pairs in length this would lead to a memory requirement of several Terabytes. This amount of memory is prohibitive for most commodity computers of today. To date, no algorithm is known that uses asymptotically less time than $O(n^2)$ and keeps the same generality as the DP algorithms mentioned in this section. However, with respect to space, the complexity can be improved from quadratic, $O(l_1 \times l_2)$, to linear, $O(l_1 + l_2)$, without losing any generality.

Hirschberg was the first to present a linear space algorithm capable of producing an optimal alignment of two sequences [87]. Although Hirschberg made his observations in the context of the problem of finding a longest common subsequence of two strings, the results also apply to the common sequence alignment problem. Myers and Miller are credited with developing the first linear space algorithm for optimal sequence alignment based on Hirschberg's algorithm [112]. The maximal score in the *similarity matrix* can be computed in linear space as follows. An important observation is that the derivation of the column $i$ requires only the column $i - 1$ of the matrix to be known. Figure 3.12

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



presents an algorithm that computes the best score for two sequences using linear space. Algorithm *LastColumn* takes as input the strings $a[1 \ldots n]$ and $b[1 \ldots n]$, and produces as output the vector $LL$. This vector consists of the same values as the last column, $m$, of the similarity matrix computed by the Smith-Waterman algorithm. However, the *LastColumn* algorithm requires only $n + 1$ locations of memory to compute the vector $LL$; hence, it is space linear in the size of the sequences.

```
Algorithm BestScore
    Input: sequences a and b
    Output: vector LL
    m = |a|
    n = |b|
    for j = 0 to n do
        LL[j] = j × Gap_Penalty
    for i = 1 to m do
        old = LL[0]
        LL[0] = i × Gap_Penalty
        for j = 1 to n do
            temp = LL[j]
            LL[j] = max(old + align(a[i], b[j]),
                        LL[j − 1] + Gap_Penalty,
                        LL[j] + Gap_Penalty)
            old = temp
```

Figure 3.12: Pseudo-code for the *LastColumn* Algorithm

The correctness of *LastColumn* is supported by the following invariant assertions:

- at the beginning of step $i$ of the outer loop, $LL$ holds the values of the column $i-1$ of the similarity matrix;
- in the inner loop, at the beginning of step $j$, $LL[0 \ldots j-1]$ holds the values of the column $i$, while $LL[j \ldots n]$ holds the values of the column $i-1$;
- at step $j$, $old = S[i-1, j-1]$.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



The computations done by the Smith-Waterman algorithm are mimicked using only the vector $LL$, and two temporary variables. Because of the two loops, the time complexity of the $LastColumn$ algorithm is $O(n^2)$.

## 3.3 Genetic Algorithms

If we can address a CB problem analytically, this generally means that we know enough about the structure of the search space to reliably guide a search towards the best solution. The more typical situation is that we do not have enough analytical knowledge to do this. Perhaps even more commonly, we can analyze the structure of the problem to a small extent, but not enough to be able to use the knowledge to reliably find the best solution. This type of CB problems fall into the heuristic category.

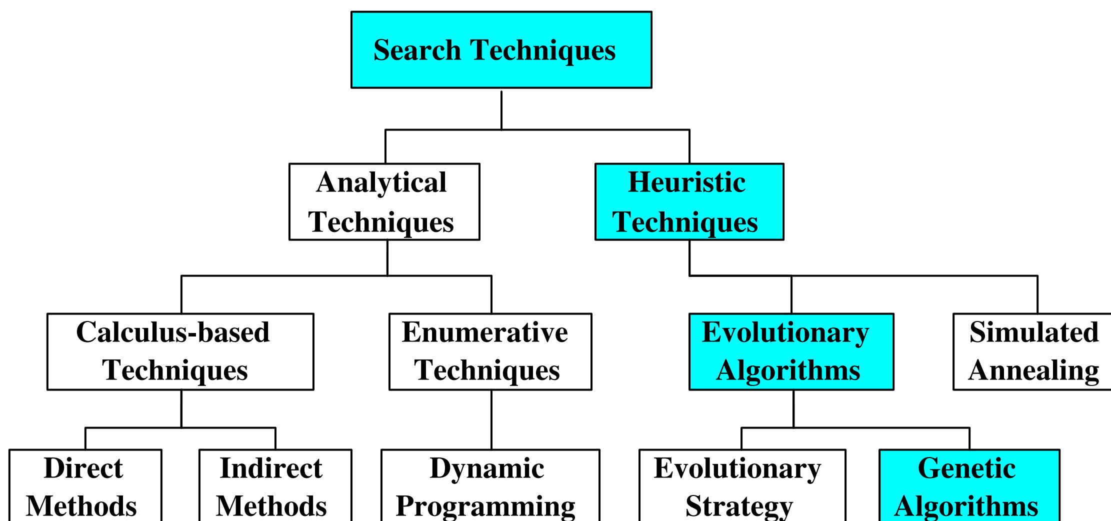



Figure 3.13: Classification of search techniques

The so-called genetic algorithm (GA) is a heuristic search method that operates on pieces of information like the nature dose on genes in the course of evolution (see Figure 3.13). GAs and evolutionary strategies emerged at about the same time. Both techniques model the natural evolution process in order to optimize either a fitness function

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(evolutionary strategies) or the effort of generation subsequent, well-adapted individuals in successive generations (GAs). In a GA, individuals are represented by a linear string of letters of an alphabet and they are evaluated by a fitness function. Depending on the generation replacement mode a subset of parents and offspring enters the next reproduction cycle. After a number of iterations the population consists of individuals that are well adapted in terms of the fitness function. Although this setting is reminiscent of a classical function optimization problem, GAs were originally designed to demonstrate the benefit of genetic crossover in an evolutionary scenario, not for function optimization. It cannot be proven that the individuals of a final generation contain an optimal solution for the objective encoded in the fitness function but it can be shown mathematically that the GA optimizes the effort of testing and producing new individuals if their representation permits development of building blocks. In that case, the GA is driven by an implicit parallelism and generates significantly more successful progeny than random search. In a number of applications where the search space was too large for other heuristic methods or too complex for analytical treatment GAs produced favorable results.

GAs differ from traditional search techniques in several ways:

- GAs optimize the trade-off between exploring new points in the search space and exploiting the information discovered thus far.
- Second, GAs have the property of implicit parallelism. Implicit parallelism means that the GA's effect is equivalent to an extensive search of hyper-planes of the given space, without directly testing all hyperplane values.
- Third, GAs are randomized algorithms, in that they use operators whose values are governed by probability. The results for such operations are based on the value of a random number.
- GAs operate on several solutions simultaneously, gathering information from cur-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



rent search points to direct subsequent search. Their ability to maintain multiple solutions concurrently makes GAs less susceptible to the problems of local maxima and noise.

As noted, GAs are randomized-but not random-search algorithms. Each organism represents a point in the search space. Randomization must balance two competing concerns, exploration and exploitation. A solution cannot be tested unless it appears as an organism. Therefore, a reasonable number of solutions must be explored. On the other hand, unlimited exploration would not be efficient search. The strength of highly fit organisms must be exploited and allowed to propagate in the population. Yet, giving too much precedence to such organisms results in premature termination at a local optimum.

The basic outline of a GA is as follows [95]:

1. *Initialise a population of individuals.* This can be done either randomly or with domain specific background knowledge to start the search with promising seed individuals.
   - *Individuals* are represented as a string of bits.
   - A *fitness function* must be defined that takes as input an individual and returns a number that can be used as a measure of the quality of that individual.
2. *Evaluate* all individuals of the initial population.
3. *Generate* new individual. Reproduction involves domain specific genetic operators (see Figure 3.14). Operations to produce new individuals are:
   - *Mutation.* Substitute one or more bits of an individual randomly by a new value (0 or 1).
   - *Crossover.* Exchange parts of one individual with the corresponding parts of another individual.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



4. *Select* individuals for the new parent generation.

5. Go back to step 2 until either a desired fitness value is reached or until a predefined number of iteration is performed.

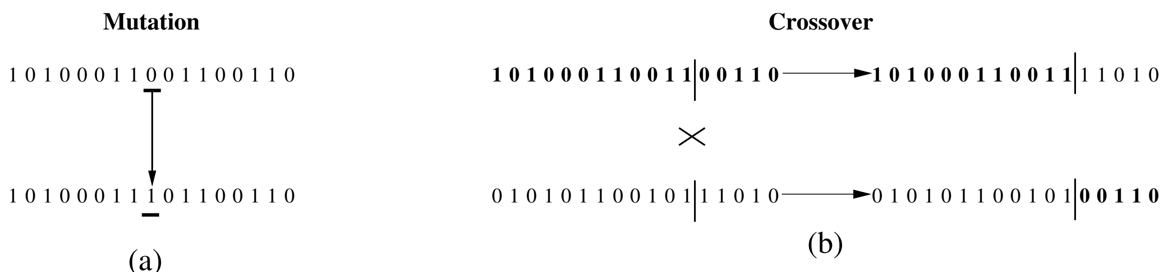



Figure 3.14: Genetic operators for GAs. (a) Mutation exchanges one single bit; (b) Crossover exchanges a contiguous fragment of an individual.

The theoretical foundation of GAs is the *schemata theorem* [88]. It makes a statement about the propagation of *schemata* (or *building blocks*) within all individuals of one generation. A schema is implicitly contained in an individual. Like individuals, schemata consist of bit strings (1,0) and can be as long as the individual itself. In addition, schemata may contain undefined positions where it is not specified whether the bit is 1 or 0. A string is said to match a schema if they agree in the defined positions. For example:

The string 1010101 matches the schemata $*0*0***$ and $***01**$ among others, but does not match $*1*0***$ since they differ in the second position. The *length* ($\delta(H)$) of $*0*0***$ is 3 which is the distance from the first to the last fixed symbol (1 or 0 but not $*$). The *order* of a schema $o(H)$ is the number of fixed positions (1 or 0 but not $*$).

According to [79], let $s(H,t)$ be the number of occurrences of a particular schema $H$ in a population of $n$ individuals at time $t$. The bit string $A_i$ of individual $i$ then gets selected for reproduction with probability $p_i$:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{3.3.1}$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



where $f_i, f_j$ is the fitness value of the $i, j-$th individual. The expected number of occurrences of schema $H$ at time $t+1$ is:

$$s(H, t+1) = s(H, t) \cdot n \cdot \frac{\overline{f}(H)}{\sum_{i=1}^{n} f(i)} \tag{3.3.2}$$

where $\overline{f}(H)$ is the average fitness of all individuals (strings $A_i$) that contain $H$.

Crossover and mutation operators can destroy schemata during reproduction. The longer a single individual, the smaller the probability that a schema $H$ will be involved in a crossover event. The longer a schema, i.e. the larger $\delta(H)$, the more likely is its destruction through recombination with another individual. Hence, for *crossover* the lower bound for the survival probability of a schema $H$ is:

$$p_s \geq 1 - \frac{\delta(H)}{L-1} \tag{3.3.3}$$

where L is the length of one whole individual. If we perform crossover stochastically at a frequency $p_c$ the survival probability $p_s$ becomes:

$$p_s \geq 1 - p_c \cdot \frac{\delta(H)}{L-1} \tag{3.3.4}$$

Combining the effects of independent crossover and reproduction we arrive at the following equation for the expected occurrence of a schema $H$ at time $t+1$:

$$s(H, t+1) = s(H, t) \cdot n \cdot \frac{\overline{f}(H)}{\sum_{i=1}^{n} f(i)} \cdot \left(1 - p_c \cdot \frac{\delta(H)}{L-1}\right) \tag{3.3.5}$$

3.3.5 presents that schemata increase over time proportional to their relative fitness and inversely proportional to their length.

*Mutation* can effect a schema $H$ at each of its $o(H)$ fixed positions with mutation

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



probability $p_m$. Survival of a single constant position in a schema is then $p_s = 1 - p_m$ and survival of the entire schema:

$$p_s = (1 - p_m)^{o(H)} \tag{3.3.6}$$

where small $p_m$ can be approximated by $p_s \approx 1 - o(H) \cdot p_m$. Combining the effects of independent mutation, crossover and variation we get the following formula for the expected count of a schema $H$:

$$s(H, t+1) = s(H, t) \cdot n \cdot \frac{\overline{f}(H)}{\sum_{i=1}^{n} f(i)} \cdot \left(1 - p_c \cdot \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right) \tag{3.3.7}$$

Assuming a schema $H$ could always outperform other schemata by a fraction $b$ of the total mean fitness then this equation can be rewritten as:

$$\begin{aligned} s(H, t+1) &= s(H, t) \cdot \frac{\frac{1}{n}\sum_{i=1}^{n} f_i + b\frac{1}{n}\sum_{i=1}^{n} f_i}{\frac{1}{n}\sum_{i=1}^{n} f_i} \cdot \left(1 - p_c \cdot \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right) \\ &= s(H, t) \cdot (1 + b) \cdot \left(1 - p_c \cdot \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right) \end{aligned} \tag{3.3.8}$$

This equation tells us that the number of schemata better than average will exponentially increase over time. Effectively, many different schemata are sampled implicitly in parallel and good schemata will persist and grow. This is the basic rationale behind GAs. It is suggested that if the representation of a problem allows the formation of schemata then the GA can efficiently produce individuals that continuously improve in terms of the fitness function.

GAs have been widely used, Table 3.2 shows some popular GA applications in CB.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Table 3.2: Popular GA applications in CB.

| **Algorithms** | **Applications** | **References** |
|---|---|---|
| SAGA | Mutiple protein sequence alignment | [117] |
| RAGA | Pairwise RNA sequence alignment | [118] |
| Hybrid GA | DNA sequence reconstruction | [33] |
| KENOBI | Protein structure alignment | [140] |
| GGA | Microarray data clustering | [58] |
| GARC | Feature selection methods for in silico drug design | [53, 56] |

## 3.4 Summary

The characteristics of sequential CB algorithms are fundamental and important to the design of efficient parallel algorithms. In this chapter, we have analyzed characteristics of two categories of popular CB algorithms: DP algorithms and GAs. According to the problem dimension and the data dependency relationship, we have presented a general classification for popular DP algorithms in CB. From the point of view of computational load density, we have identified two kinds of DP algorithms further-*regular* and *irregular*. The data dependency relationship graphs are also provided to facilitate the understanding of characteristics of variable DP algorithms in CB. As to GAs, the theoretical foundation of them has been introduced.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Chapter 4

# Design of Partitioning and Communication Schemes

## 4.1 Introduction

The exponential growth in the size of genomic and protein databases and the availability of complete genomes of complex organisms have made the CB area keen for HPC. Due to the wide variety of problems solved in CB and variable HPC architectures, we need to consider the characteristics of specific applications to perform. Also, we must manage additional implementation concerns introduced by HPC.

The most basic problem is which parallel partitioning scheme to use, that is, how the algorithm is to be broken into independent computations that may be executed in parallel. There are two primary methods for partitioning a problem:

- *Data Partition*: partition the data first, then partition the computation based on the data partition.
- *Functional Partition*: partition the computation into smaller tasks, then, partition the data based on these tasks. This is common in problems where there are no obvious data structures to partition, or where the data structures are highly

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



unstructured.

In this thesis, we have used the data partition method because it is general-purpose and suitable for CB algorithms such as DP and GAs which operate on regular data-structures. The data-structure is divided between the processors, which compute a portion of the result.

Another concern is to balance the computational load among processors. It is discussed below:

- A parallel computation usually proceeds at the speed of the slowest processor. Therefore it is important to ensure that a computational component completes at approximately the same time as its result is required by another component. For some problems it is possible to determine which are the more expensive computations by inspection. In these cases the load balancing can be achieved statically by allocating these components more processors. In other situations a dynamic approach must be used in which the implementation adapts to expensive computations by re-routing work elsewhere. As to our parallel DP applications in CB, we proposed a *tunable coarse-grained partitioning and communication scheme* to achieve better load balancing.

- In parallel GAs, the load balancing will not matter greatly. For example, imagine one process being deployed to a slow machine with slow networking capabilities, and another one to a fast machine with lots of networking resources. For some kinds of parallel algorithms this would be a problem, since the slow machine would probably keep the faster one up. As to GAs this is not a problem because both sub-populations would evolve with different speeds. The slower sub-population may see the immigration of advanced individuals earlier in its evolution than the faster sub-population. In this thesis, we have mainly focused on the design and development

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



of a hierarchical parallel GA running on computational grids.

## 4.2 Parallel Dynamic Programming Algorithms

### 4.2.1 Striped Partitioning

The parallelization of DP algorithms has been done in different ways depending on the particular parallel architecture being used. On fine-grained architectures, the computation of each cell within an anti-diagonal is parallelized [128, 127, 126]. However, this technique is only efficient on architectures such as systolic arrays, which have an extremely fast inter-processor communication.

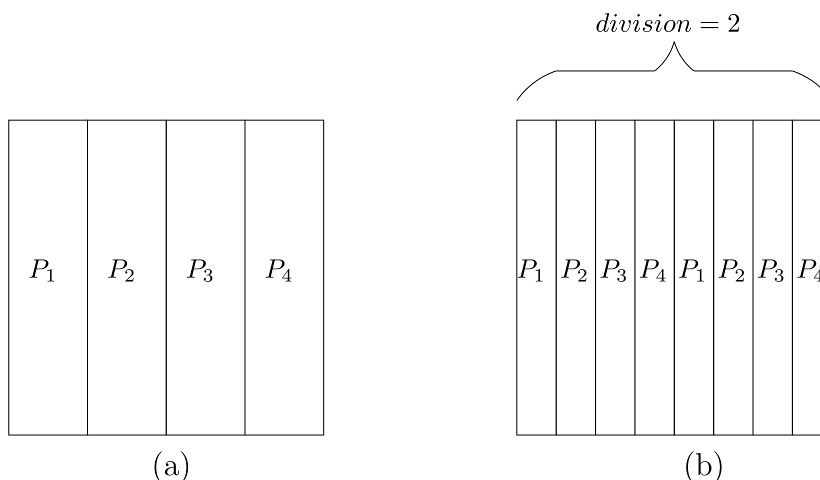



Figure 4.1: (a) Columnwise striping (b) Columnwise cyclic striping

On coarse-grained architectures (The term *coarse-grained* comes from the fact that the problem size in each processor $n/p$ is considerably larger than the number of processors) like PC clusters it is more convenient to assign an equal number of adjacent columns to each processor as shown in Figure 4.1a. This method is called the *columnwise striping*. The method illustrated in Figure 4.1b is called the *columnwise cyclic striping* [99]. Using a columnwise cyclic distribution of columns can reduce uneven workload. The parameter

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



*division* is used to implement a cyclic distribution of columns to processors. Increasing the number of cyclic divisions leads to a better load balancing.

The partitioning method in Figure 4.1 does not consider dependency relations between neighbor processors. The parallel computation can proceed normally if there are not any communications among processors. However, if the result of a computational component is required by another component, the communication between them is a necessary concern and these two partitioning methods need to be improved.

### 4.2.2 Block-based Partitioning

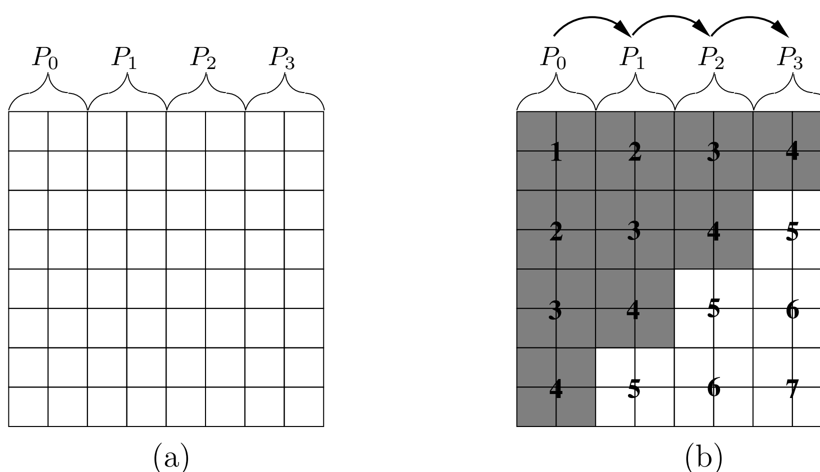



Figure 4.2: (a) block-based distribution of an $8 \times 8$ matrix using 4 processors, (b) DP computation for 4 processors, 8 columns and a $2 \times 2$ block size. The complete $8 \times 8$ matrix can then be computed in 7 iteration steps

The concept of *block-based* partitioning method is shown in Figure 4.2a. Matrix cells are grouped into blocks. Processor $i$ computes all the cells within a block after receiving the required data from processor $i - 1$. Figure 4.2b shows an example of the computation for 4 processors, 8 columns and a block size of $2{\times}2$, the numbers 1 to 7 represent consecutive phases in which the cells are computed.

The block-based partitioning method works efficiently for regular DP algorithms.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



However, there are many examples of irregular DP problems. For these problems this method does not work well, since it leads to an uneven workload.

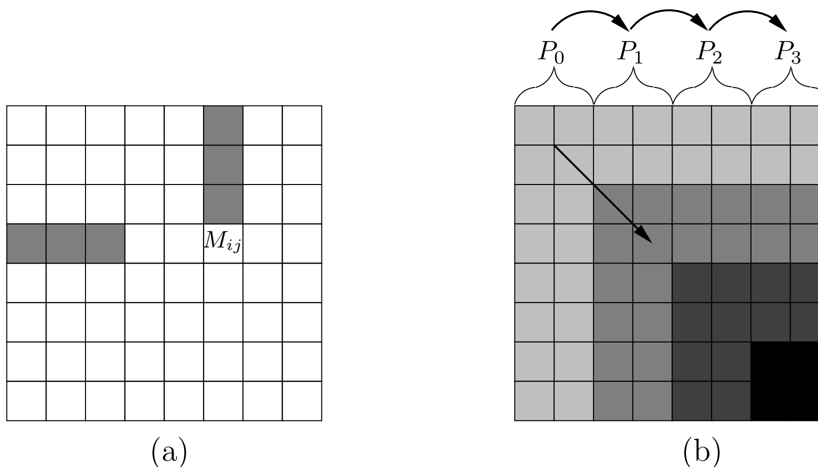



Figure 4.3: (a) Example of an irregular dependency pattern; (b) Distribution of load computation density

Figure 4.3a shows an example of an irregular DP algorithm: matrix element $(i, j)$ is computed from all matrix cells in row $i$ and column $j$ from index 1 up to $min(i-1, j-1)$. The load to compute one element in the matrix (we call this the *load computation density*) will then increase along the shift direction of computation. We can see from 4.3b that the load computation density at the bottom right-hand corner is much higher than that in the top left-hand corner. The block-based method in Figure 4.2 will therefore lead to a poor performance, since the workload on processor $P_i$ is higher than processor $P_{i-1}$.

### 4.2.3 Tunable Coarse-grained Partitioning and Communication Scheme

In this thesis, we have proposed a tunable coarse-grained (TCG) partitioning and communication scheme. It is illustrated in Figure 4.4. The parameter *division* is used to implement a cyclic distribution of columns to processors. The parameter *rowwidth* is used to control the size of messages that processor $P_i$ sends to processor $P_{i+1}$. Increasing

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



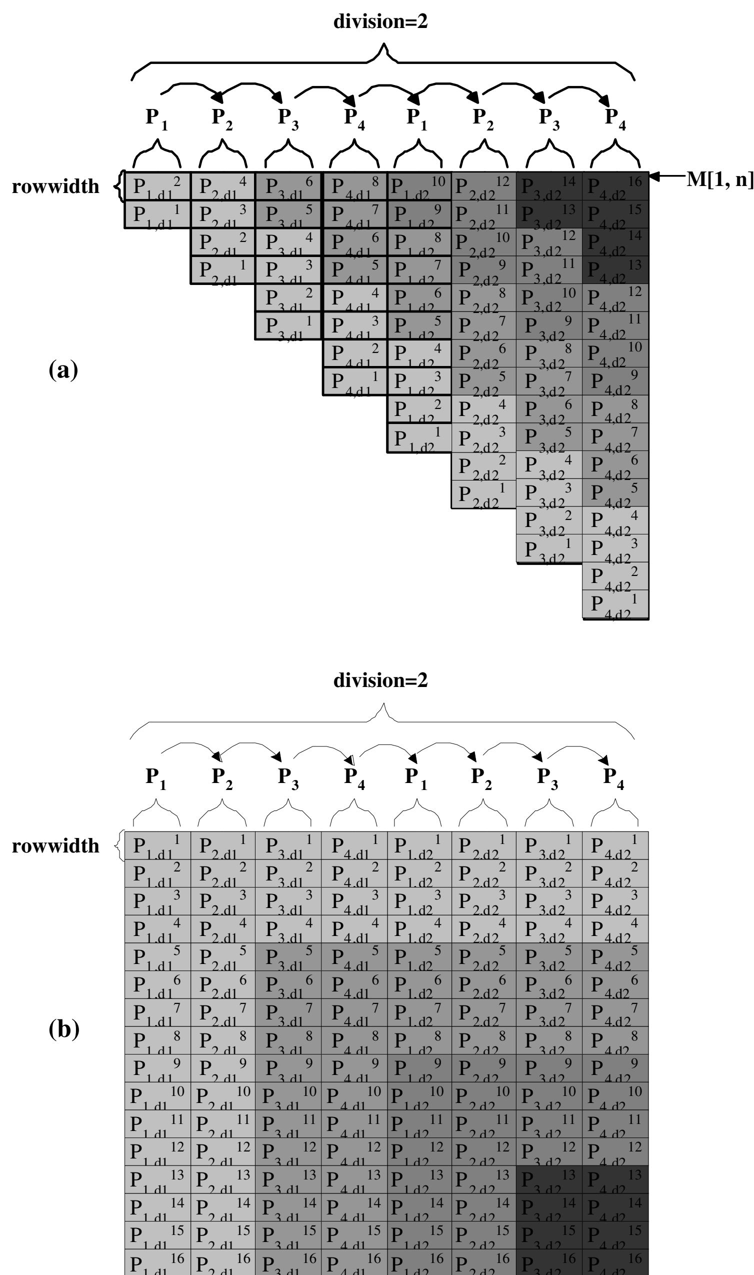



Figure 4.4: The tunable coarse-grained partitioning and communication scheme for (a)The triangular matrix computation, (b)The square matrix computation

the number of cyclic divisions and decreasing the size of messages can lead to a better load balancing. Of course, doing this also increases the communication overhead. Thus, the choice of the parameter *division* and *rowwidth* is a trade-off between the load bal-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



**Input:** The number of processors $p$, the value of *division* and *rowwidth*. ($P_k$ denotes the $k$-th processor, $n \times n$ is the size of matrix $M$, $d_t$ denotes the $t$-th division).
**Output:** Depending on the requirements of the given applications, the output will be optimal score $M[1, n]$ or the whole matrix $M$.
**Begin**
  **for** $P_k (1 \leq P_k \leq p)$ in division $d_t (1 \leq d_t \leq division)$ **do**
    **if** $P_k \neq 1$ **then**
      receive message from processor $P_{k-1}$;
    **for** $\left\{ \begin{array}{l} (a) i = P_k \times \frac{n}{p \times division} + (d_t - 1) \times \frac{n}{division} \text{ to } 1 \\ (b) i == 1 \text{ to } n \end{array} \right\}$ **do**
      after $\left\{ \begin{array}{l} (a) i \text{ is reduced by } rowwidth \\ (b) i \text{ is increased by } rowwidth \end{array} \right\}$ **do**
        **if** $P_k == 1$ **then**
          **if** $d_t == 1$ **do**
            send message to $P_2$;
          **if** $d_t > 1$ **do**
            receive message from $P_p$;
            send message to $P_2$;
        **if** $1 < P_k < p$ **then**
          receive message from $P_{k-1}$;
          send message to $P_{k+1}$;
        **if** $P_k == p$ **then**
          receive message from processor $P_{k-1}$;
          **if** $d_t \neq division$ **then**
            send message to processor $P_1$;
    **for** $\left\{ \begin{array}{l} (a) j = i \text{ to } P_k \times \frac{n}{p \times division} + (d_t - 1) \times \frac{n}{division} \\ (b) j = (P_k - 1) \times \frac{n}{p \times division} + (d_t - 1) \times \frac{n}{division} + 1 \text{ to } \\ \quad P_k \times \frac{n}{p \times division} + (d_t - 1) \times \frac{n}{division} \end{array} \right\}$ **do**
      compute $M[i, j]$;
**End**

Figure 4.5: The general parallel algorithm for the tunable coarse-grained partitioning and communication scheme. (a) For the triangular matrix computation, (b) For the square matrix computation

ancing and communication time. In Figure 4.4, $P^k_{i,d_j}$ denotes the block of processor $P_i$ at division $j$ and step $k$. Initially $P_1$ starts computing at block 1 in division 1. In the same division, after $P_i$ completes computing block 1, it sends this part to $P_{i+1}$ and then $P_{i+1}$ can work at the block 1 in this division. Between two different divisions, the last processor will send message to $P_1$. This parallel procedure will continue until all cells on

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



matrix $M$ are computed.

The general parallel algorithm for this scheduling scheme is presented in Figure 4.5. We can get the following theorem according to Figure 4.4 and Figure 4.5.

**Theorem.** *The Proposed algorithm uses* $\alpha \times (division \times p - 1) \times \frac{n}{rowwidth}$ *communication steps with* $O(\frac{n^3}{p})$ *sequential computing time on each processor.* ($\alpha$ is $\frac{1}{2}$ for the triangular matrix computation in Figure 4.4a; $\alpha$ is 1 for the square matrix computation in Figure 4.4b)

**Proof:**

Processor $P_i$ sends $P^k_{i,d_j}$ to $P_i + 1$ after the $k$-th step in division $d_j$. In the same division, after $\alpha \times \frac{n}{rowwidth}$ communication steps, processor $P_i$ completes its work and moves to the next division to continue this loop until finish computing the sub matrix allocated to itself. Each moving to the next division will bring another computation and communication loop, thus, after $\alpha \times (division \times p - 1) \times \frac{n}{rowwidth}$ communication steps, all the processors have completed their work.

As mentioned in Chapter 3, the time complexity of $2D/1D$ DP algorithms is $O(n^3)$. Increasing the number of *division* can balance the load among processors. For example, as to the Nussinov algorithm, the load on processor $P_i$ is about $(2p^2 \times division^3 + 6 \times p \times p \times p_i \times division^2) \times \frac{n^3}{division^3 p^3}$. When the parameter *division* is increased, the lower power item $6 \times P \times P_i \times division^2$ can be omitted. The remainder part is direct proportional to $\frac{n^3}{p}$. Thus, each processor needs almost the same $O(\frac{n^3}{p})$ sequential computing time. ♯

There are two problems that should be solved when implementing this parallel algorithm. One is the residue problem. That is because the matrix dimension can not be divided exactly by the product of the number of divisions and the number of processors, usually there is a residue. That is, there is one processor that will compute more elements than others; we call this processor the Special Processor (SP).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



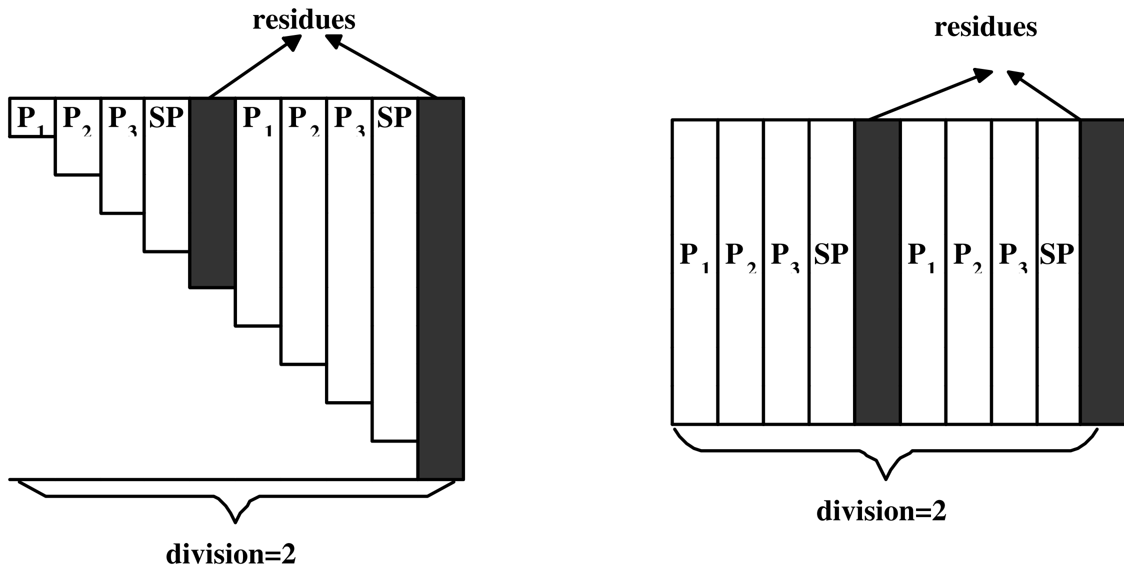



Figure 4.6: Residues are distributed evenly through the whole matrix

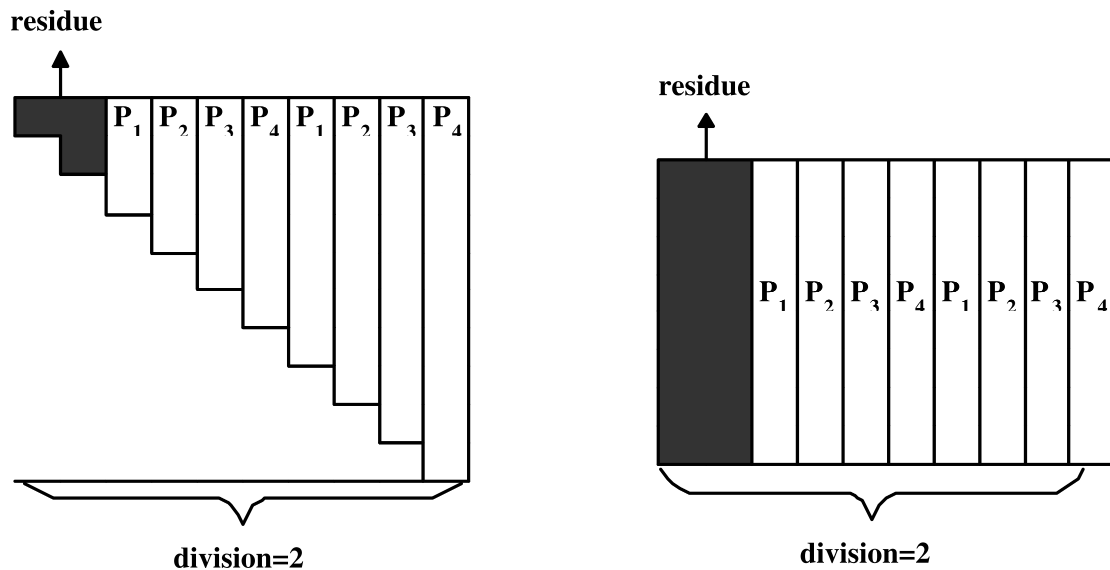



Figure 4.7: Residues are put to the forefront of the matrix

When the product of the number of divisions and number of processors is large, there will be a big residue, thus the negative influence will be serious. This is because the residue may be much larger than the average block size. If the residue is distributed evenly through the matrix (as shown in Figure 4.6), the workload on the SP (usually the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



last processor) will cause work imbalance greatly.

For example, if the matrix dimension is 4000×4000, the number of processors is 2, the division is 700, then the residue is 1200, and the average block size is only 2. So the SP will compute about 600 times more workload than the average one. Because the matrix is uneven, this big residue will cause serious imbalance. In order to solve this problem, we put the residues to the forefront of the matrix (as shown in Figure 4.7). In the forefront part the workload density is much lower, so the residue influence will be very light.

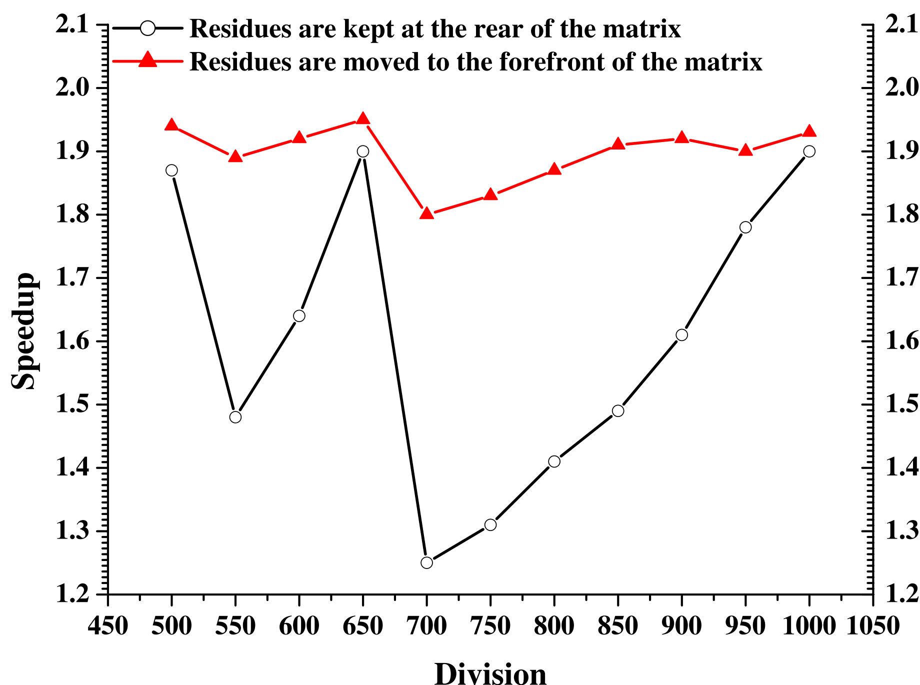



Figure 4.8: Performance comparison for the skyline matrix problem using different methods to treat the residue. The performance is measured on two Intel Pentium IV Xeon 2.6GHz processors in a PC cluster.

Figure 4.8 shows the performance measurements on a PC cluster for the skyline matrix problem using different methods to treat the residue. The matrix size is 4000×4000, the number of processors is 2, the rowwidth is 10, the division is set from 500 to 1000. From the figure we can see that if the residues are not moved to the forefront of the matrix, the performance will be affected by the values of residues greatly (see the curve with circles

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



in Figure 4.8). Otherwise, when the residues are put to the forefront of the matrix, the performance curve will be much more smooth (see the curve with triangles in Figure 4.8). So, moving residues to the forefront of the matrix is an efficient way to solve the residue problem.

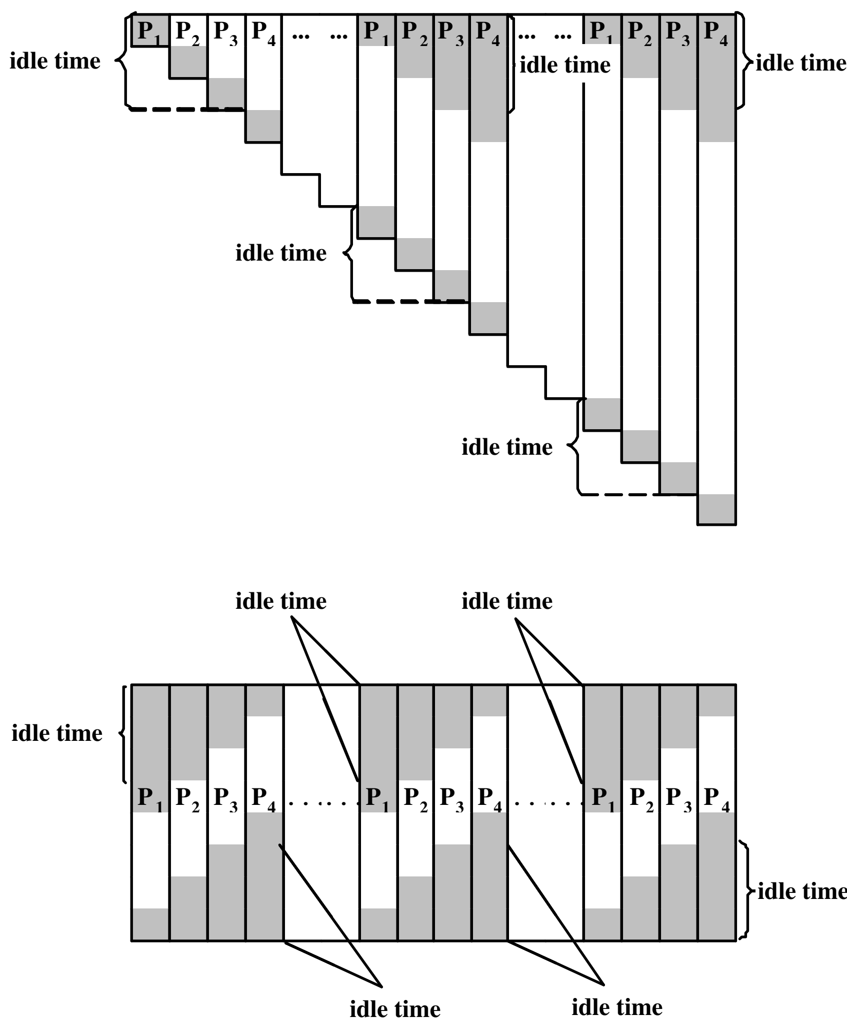



Figure 4.9: Communication scheme using synchronous communication

Another problem that should be considered is what scheduling of communication to use. If we use *synchronization barriers*, each division will not begin the computing until its previous division has completed the computing. Synchronization barriers can give reasonable predictions on the performance of algorithms when implemented on parallel

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



architectures. For instance, the *BSP* model [143] consists of a sequence of super-steps separated by Synchronization barriers. In a super-step, each processor executes a set of independent operations using local data available in each processor at the start of the super-step, as well as communication consisting of send and receive of messages. However, synchronization barriers are very expensive since it introduces the idle time between all the processors in each division. From Figure 4.9 we can see that each division will introduce twice idle time among processors because the barriers between divisions.

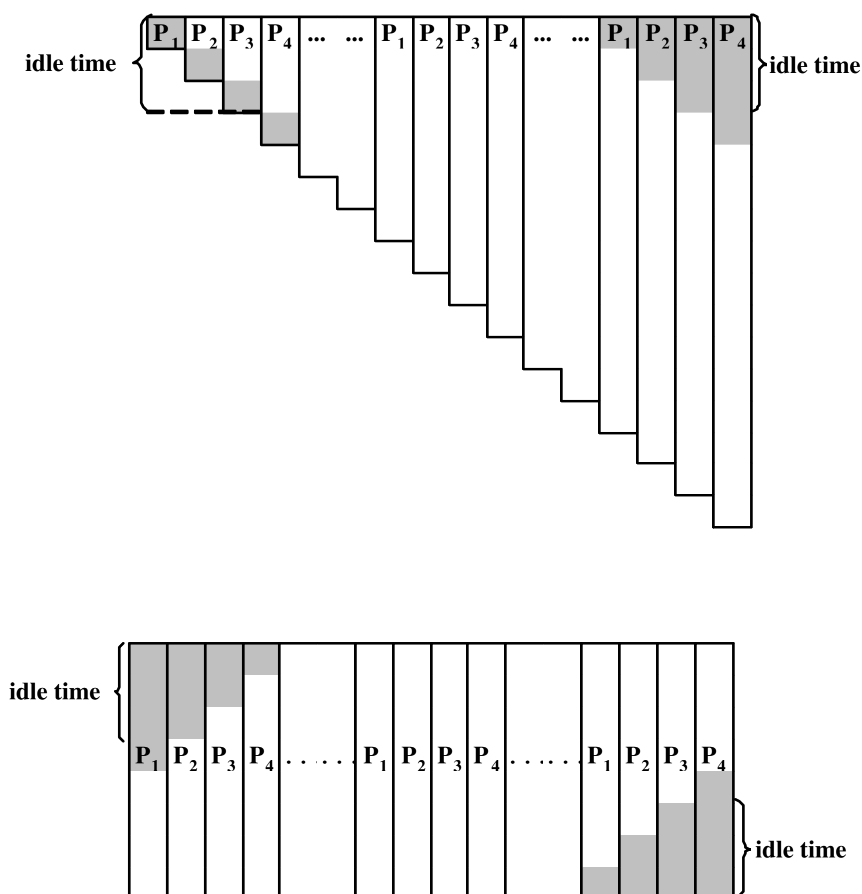



Figure 4.10: Communication scheme using asynchronous communication

In this thesis, we have used the asynchronous communication mode to reduce the idle time introduced by synchronization barriers between all the divisions. The asynchronous communication is also called non-block communication. It is used to overlap

**Input:** The number of processors $p$, the value of *division* and *rowwidth*. ($P_k$ denotes the $k$-th processor, $n \times n$ is the size of matrix $M$, $d_t$ denotes the $t$-th division).
**Output:** Depending on the requirements of the given applications, the output will be optimal score $M[1, n]$ or the whole matrix $M$.
**Begin**
compute and move residues to the forefront of the matrix;
**for** $P_k (1 \leq P_k \leq p)$ in division $d_t (1 \leq d_t \leq division)$ **do**
**if** $P_k \neq 1$ **then**
receive message (synchronous way) from processor $P_{k-1}$;
**for** $\left\{ \begin{array}{l} (a) i = P_k \times \frac{n}{p \times division} + (d_t - 1) \times \frac{n}{division} \text{ to } 1 \\ (b) i == 1 \text{ to } n \end{array} \right\}$ **do**
after $\left\{ \begin{array}{l} (a) i \text{ is reduced by } rowwidth \\ (b) i \text{ is increased by } rowwidth \end{array} \right\}$ **do**
**if** $P_k == 1$ **then**
**if** $d_t == 1$ **do**
send message (asynchronous way) to $P_2$;
**if** $d_t > 1$ **do**
receive message (synchronous way) from $P_p$;
send message (asynchronous way) to $P_2$;
**if** $1 < P_k < p$ **then**
receive message (synchronous way) from $P_{k-1}$;
send message (asynchronous way) to $P_{k+1}$;
**if** $P_k == p$ **then**
receive message (synchronous way) from processor $P_{k-1}$;
**if** $d_t \neq division$ **then**
send message (asynchronous way) to processor $P_1$;
**for** $\left\{ (a) j = i \text{ to } P_k \times \frac{n}{p \times division} + (d_t - 1) \times \frac{n}{division} \right.$dto to

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



communication mode is very suitable for our algorithm. The number of barriers is reduced, so the idle time is reduced greatly. Figure 4.10 shows that after we use the asynchronous mode, there are only twice idle time spaces throughout the matrix; it is a huge performance win.

Figure 4.11 presents the whole algorithm after moving residues to the forefront of the matrix and using the asynchronous communication mode.

## 4.3 Design of a Hierarchical Parallel Genetic Algorithm for Protein Folding on Computational Grids

### 4.3.1 Protein Folding Problems with HP Lattice Models

Knowing a protein's spatial structure is one of the foremost goals of molecular biology, because it is this structure that determines the protein's function. However, determining the three-dimensional structures of proteins using techniques such as x-ray crystallography and nuclear magnetic resonance has proved to be difficult, costly, and not always feasible. As a result, there currently exists a gap between the number of proteins with known sequences and the number of proteins with known three dimensional structures. This gap has been widening every year. For example, 192,799 protein sequence entries have been in the SwissProt protein sequence database as of August 2005, but only 32,434 protein structures have been deposited in the RCSB Protein Data Bank [13]. This corresponds to a ratio of approximately to 6 sequences to 1 structure. This situation has caused much interest in searching for protein structure prediction methods using algorithmic techniques.

Proteins are synthesized as linear chains of amino acids. They then form secondary structures along this chain, such as alpha helices and beta sheets, as a result of interactions between side chains of nearby amino acids. The region of the molecule with these

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



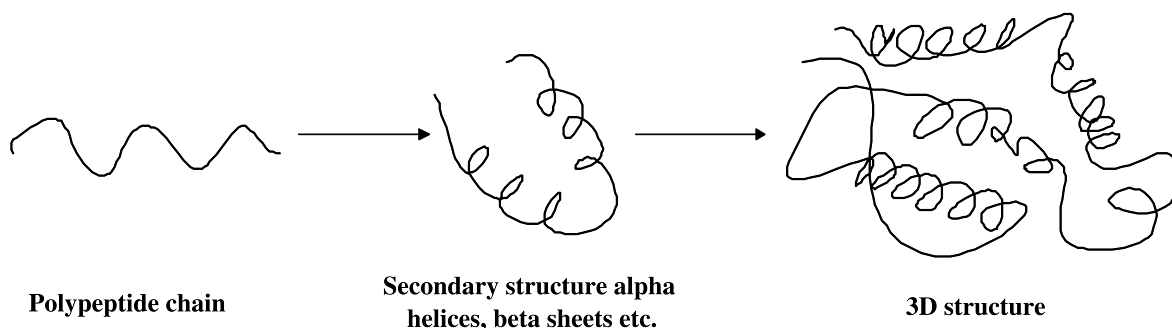



Figure 4.12: Folding of a protein from a linear chain of amino acids to a three-dimensional structure. The folding pathway involves amino acid interactions. Many different amino acid patterns are found in the same types of folds. Thus making structure prediction from amino acid sequence a difficult undertaking.

secondary structures then folds back and forth on itself to form tertiary structures. These include alpha helices, beta sheets comprising interacting beta strands, and loops (Figure 4.12) [110]. Anfinsen [24] showed that folding only requires knowledge of the amino acid sequence alone. The determination of the 3D structure from its sequence is known as the protein folding problem (PFP). Although this problem has been intensely researched since the early 1950s, no completely satisfactory solution has been found so far.

It has been shown that the PFP is NP-hard [84]. Hence, exhaustive search of a protein's conformational space is not a feasible algorithmic strategy even for small protein sequences. Consequently, heuristic optimization methods seem the most reasonable algorithmic choice to solve the PFP. In particular, a number of studies of the use of GAs for the PFP have been made in the past decade [50, 103, 119, 139, 142].

Though experiments on small proteins [24, 93] suggest that the native state of a protein corresponds to a free energy minimum, this is not yet proven. Nevertheless, this hypothesis is widely accepted, and forms the basis for computational predictions of a protein's conformation from its amino acid sequence. Lattice models are based on the minimum free energy hypothesis. They have proven to be extremely useful tools for

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



solving the PFP. By sacrificing atomic details, lattice models can be used to extract essential principles, make predictions, and unify our understanding of many different properties of proteins [54]. One of the important approximations made by lattice models is the discretization of the space of conformations. While this discretization precludes a completely accurate model of protein structures, it preserves important features for computing the minimum energy conformations [85].

The hydrophobic-hydrophilic (HP) models on the 2D square and 3D cubic lattices were proposed by Lau and Dill [54, 101]. They are the most predominant representative of lattice models. HP models abstract the hydrophobic interaction process in protein folding by reducing a protein to a heteropolymer that represents a predetermined pattern of hydrophobicity in the protein. This is one of the most studied simple exact models, and despite its simplicity, the model is powerful enough to capture a variety of properties of actual proteins [85]. Although some amino acids are not hydrophilic or hydrophobic in all contexts, this model reduces a protein instance to a string of H's and P's that represents the pattern of hydrophobicity in the protein's amino acid sequence. Some extensions of the standard linear-chain HP model have been proposed in [85]. Figure 4.13 shows some examples of these models.

Solving the PFP with HP models is a good test problem for evaluating GAs because its complexity is well understood. There has been a lot of prior work developing GAs based on HP models for the PFP. Unger and Moult have used a 2D HP model to demonstrate the usefulness of GAs in the search for minimal energy conformations [142]. In a 2D HP lattice model, each protein is a linear chain of a specific sequence of $n$ amino acids. A chain conformation is represented as a self-avoiding walk on a two-dimensional square lattice. Thus, each amino acid is represented as simply occupying one lattice site, connected to its chain neighbor(s), and unable to occupy a site filled by any other residue. Bond angles are restricted to the values $90^o$, $180^o$ and $270^o$. The force field to determine the inner

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



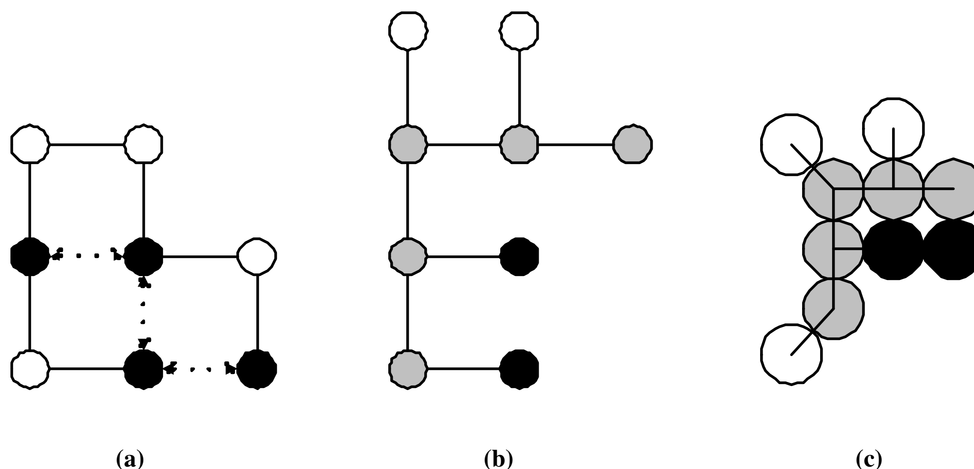



Figure 4.13: (a) The standard HP model on the square lattice. (b) The HP model with side chains on the square lattice. (c) The HP tangent spheres model with side chains. Black denotes a hydrophobic amino acid, white denotes a hydrophilic amino acid, and gray denotes a backbone element

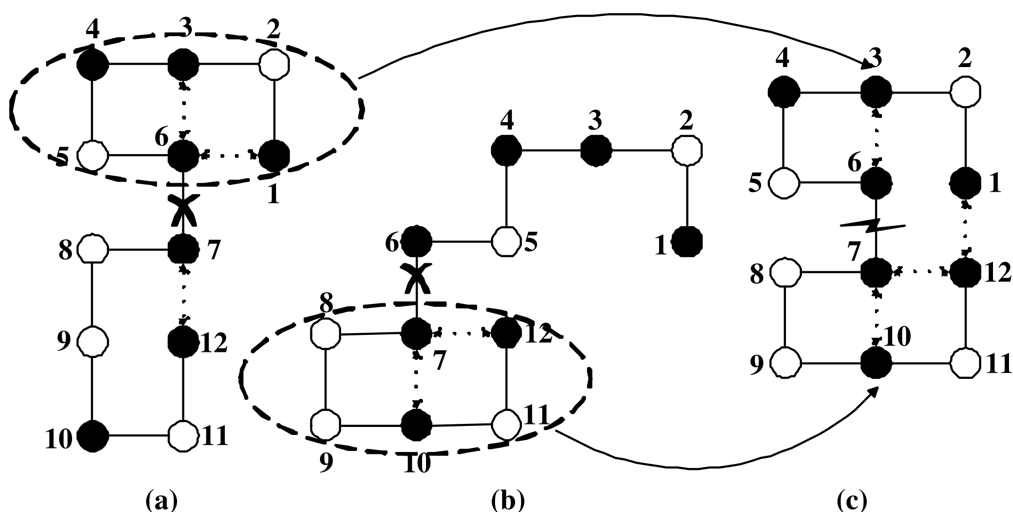



Figure 4.14: Illustration of the crossover procedure of the GA for HP lattice models. In this example the cut point is randomly chosen to be between residues 6 and 7. The first 6 residues of (A) are then joined with the last 6 residues of (B) to form the new conformation (C). The energy value of conformation (C) is $-4$, which is lower than the energies in conformations (A) ($-3$) and (B) ($-2$). Thus the new conformation is accepted.

energy of a fold is defined to be the sum of all hydrophobic interactions. A hydrophobic interaction contributes $-1$ energy units when two not directly connected hydrophobic

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



**Begin**
  $t = 0$
  initialize population $P(t)$
  $best = argmax\{F(x)|x\ in\ P(t)\}$
  repeat
    $t++$
    pointwise mutation
    $n = 0$
    while $(n < p)$
      select two conformations $m$, $f$
      produce child $c$ by crossover of $m$, $f$
      $ave$ = average$(F(m), F(f))$
      if $(F(c) \geq ave)$
        place $c$ in next generation
        $n++$
      else
        $z$ = random$(0, 1)$
        if $(z < e^{-(ave-F(c))/T})$
          place $c$ in next generation
          $n++$
    end while
    update $best$
  until convergence
**End**

Figure 4.15: The Unger-Moult GA for protein structure prediction

residues are orthogonally adjacent to each other. Every other interaction among all other possible types of neighbor pairs has energy equal to 0. For example, the "molecule" in Figure 4.13 (a) has three hydrophobic interactions and hence has the energy of $-3$ units.

The GA for HP lattice models in [142] starts with a population of $n$ extended conformations, as represented by a *chromosome*. The population at time $t$ is denoted $P(t)$. The fitness $F(c)$ of conformation $c$ equals $-E(c)$. In each generation all conformations are subject to a number of mutations. At the end of the mutation stage a crossover operation is performed as follows: For a pair of selected structures a random point is chosen along the sequence and the head part of the first structure is connected to the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



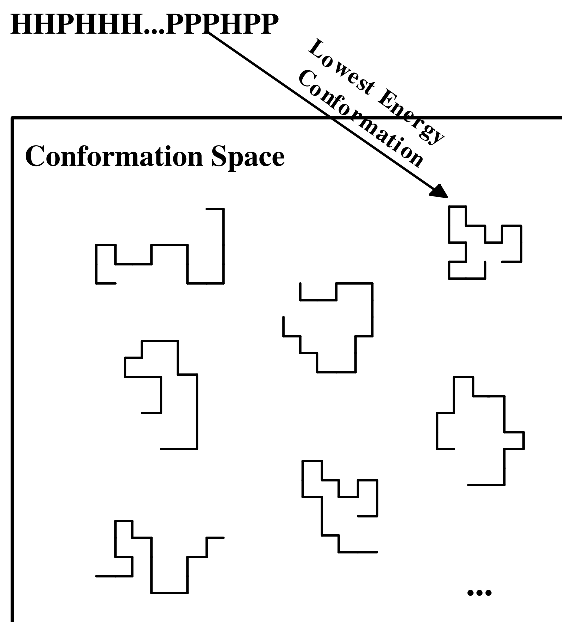



Figure 4.16: Mapping of a sequence to the conformation space.

tail part of the second structure. If the resulting path is self-avoiding and fitter than its parent conformations, the structure is then accepted. Figure 4.14 illustrates an example. The pseudocode of the algorithm is shown in Figure 4.15

For a sequence chain, the conformation space is searched, and then the energy of each conformation is evaluated to find the native conformation(s), those with the minimum energy value (see Figure 4.16).

### 4.3.2 A Hierarchical Parallel Genetic Algorithm for Protein Folding Problems

In the previous section, we have pointed out that the PFP is NP-hard and heuristic optimization methods such as GAs are suitable for solving the PFP. Unfortunately, the more likely a good solution can be found, the more computational resources are needed by GAs [124]. This leads to high runtimes on sequential architectures. Parallel processing is one approach to reduce this runtime significantly. Because of their inherent parallelism, GAs

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



are suitable candidates for running on parallel and distributed architectures. Generally, from the point of view of basic communication structures, parallel GAs can be categorized into three main types [42]:

(1) Global single-population master-slave GAs;

(2) Single-population fine-grained GAs;

(3) Multiple-population coarse-grained GAs.

In (1) there is a single panmictic population, but the evaluation of fitness is distributed among several processors. A single master processor does the supervision of the whole population and also does the selection. Slave processors receive the individuals that are recombined to create off-springs (see Figure 4.17).

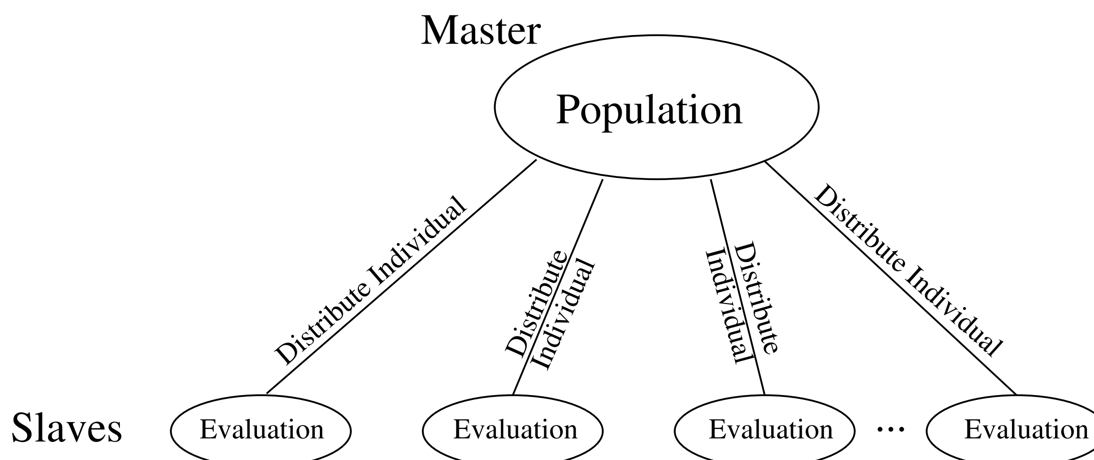



Figure 4.17: Global single-population master-slave GAs.

Fine-grained GAs are suited for massively parallel computers and consist of one spatially-structured population. Fine-grained parallel GAs have only one population,

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



but it has a spatial structure that limits the interactions between individuals. An individual can only compete and mate with its neighbors, but since the neighborhoods overlap good solutions may disseminate across the entire population. It is usually to place the individuals of a fine-grained PGA in a 2-Dimensional matrix, because in many massively parallel computers the processing elements are connected with this topology.

```
 Randomly initialize a population of
 self-avoiding HP lattice conformations;
 While not finding the lowest energy
 conformation,do
   Generate new conformations through:
     Mutation;(Substitute one or more bits
               of a conformation randomly
               by a new value)
     Crossover;(Exchange parts of one
                conformation with the
                corresponding parts of
                another conformation)
   Select conformations for the new parent
   generation;
   Evaluate all conformations;
   Migration; (Communicate with other
               populations)
end while;
```

Figure 4.18: The multiple-population GA for PFP

Multiple-population GAs consist of several subpopulations which exchange individuals occasionally. This exchange of individuals is called migration. In this paper, we mainly focus on the multiple-population coarse-grained GAs since it is the most popular GA used in CB. The outline of the multiple-population GA for the PFP with HP lattice models is shown in Figure 4.18.

Multiple-population GAs are very promising in terms of the gains in performance. Also, they are more complex than their serial counterparts. In particular, the migration

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



of individuals from one subpopulation to another is controlled by several parameters like follows:

- the topology that defines the connections between the subpopulations,
- a migration rate that controls how many individuals migrate,
- a migration interval that affects the frequency of migrations.

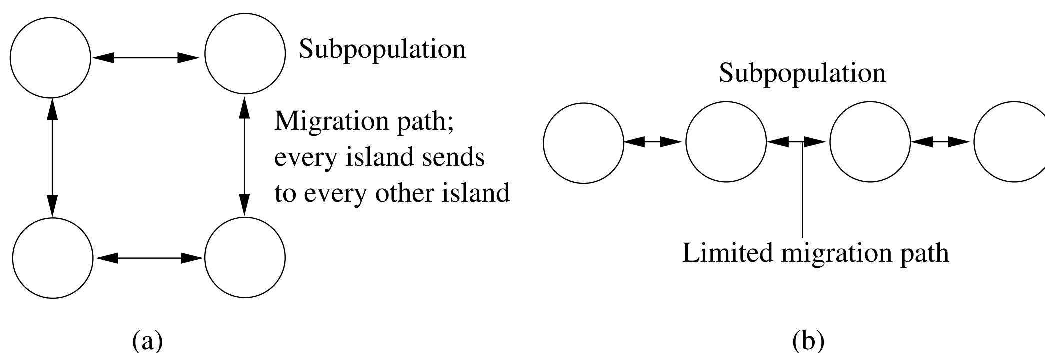



Figure 4.19: Migration models for multiple population GAs: (a) The island model (b) the stepping stone model

In the majority of multiple-population parallel GAs, migration is synchronous which means that it occurs at predetermined constant intervals. Migration may also be asynchronous so that the demes communicate only after some events occur. There are two popular approaches for modelling migration in multiple-population GAs: the island model and the stepping stone model. In the island model, individuals are allowed to be sent to any other subpopulations (see Figure 4.19a). It places no restrictions on where an individual may migrate. In the stepping stone model, migration is limited by allowing emigrants to move only to neighboring subpopulations (see Figure 4.19b). The stepping stone model reduces communication overhead by limiting the number of destinations to which emigrants may travel, and thereby limiting the number of messages. The island

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



model allows more freedom, and in some ways represents a better model of nature. However, there is significantly more communication overhead and delay when implementing such a model [149].

Combining two methods of parallel GAs at two levels producing hierarchical parallel GAs (HPGAs). HPGAs combine the benefits of its components, and it promises better performance than any of them alone [42]. Most HPGAs have multiple-population GAs at the upper level. Different types of parallel GAs can be used at the lower level. Figure 4.20 shows an HPGA with a master-slave structure on every subpopulation. Migration occurs among sub-populations and the evaluation is handled in parallel.

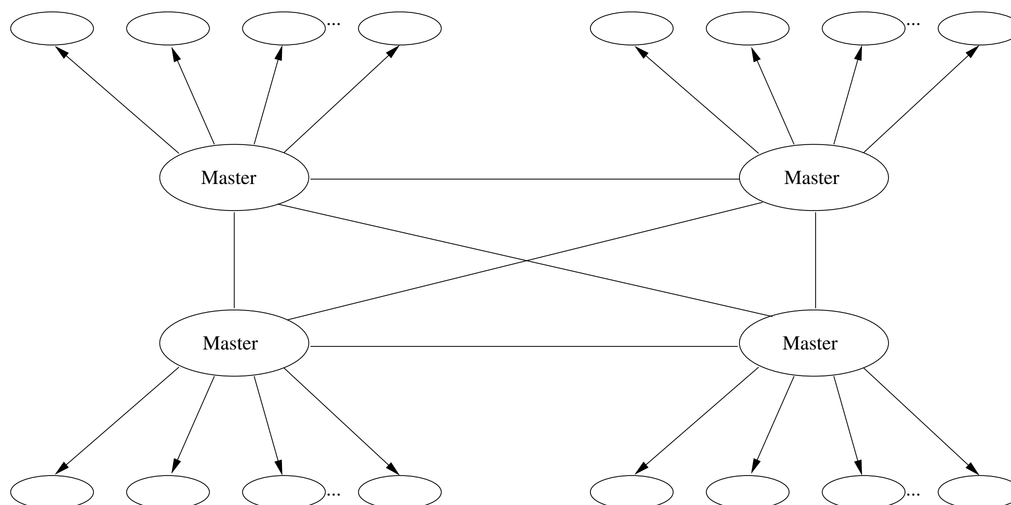



Figure 4.20: An HPGA with the master-slave structure for each subpopulation

Figure. 4.21 shows two communication architectures for HPGAs with multiple-population GAs at both the upper and the lower levels. In most cases, a high migration rate is used at the lower level and a low migration rate is used at the high level. The complexity of this kind of HPGAs would be equivalent to a multiple-population GA [42].

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



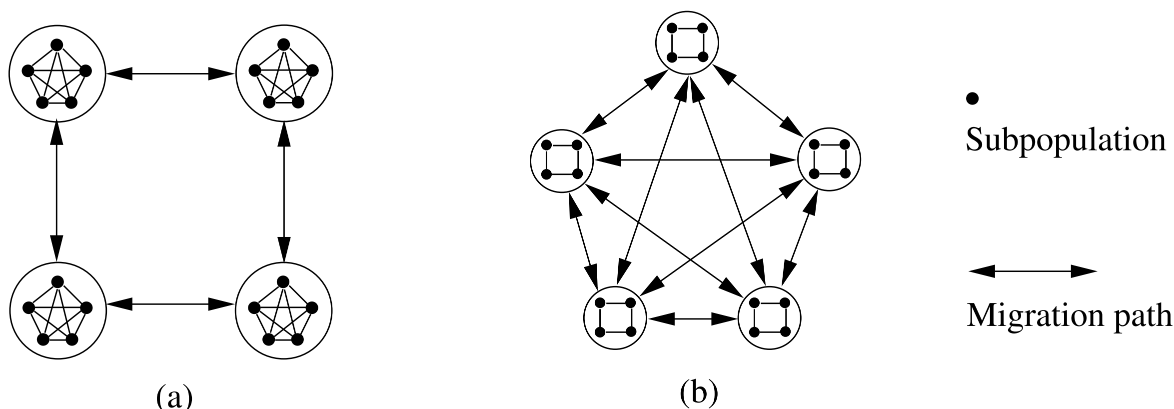



Figure 4.21: (a) An HPGA with the stepping stone model at the higher level and the island model at the lower level, (b) an HPGA with the island model at the higher level and the stepping stone model at the lower level

### 4.3.3 Communication Scheme on Computational Grids

Computational grids enable resource sharing among geographically distributed sites all over the world. These sharing resources may reside in different administrative domains, run different software, be subject to different access control policies, and be connected by networks with widely varying performance characteristics. Therefore, computational grids are hierarchical and heterogeneous environments [43]. Details are presented as follows.

- Each resource in a grid typically has grid infrastructure installation only in its control nodes. Grid users can submit tasks to these control nodes by using GRMS[4.1]. There may be a LRMS[4.2] [57, 137], e.g., PBS [12], LoadLeveler [4], LSF [11], NQE and Sun Grid Engine [16], in each geographical site shared in a grid. The LRMS will schedule the tasks in the control nodes to other nodes inside the local resource system. GRMS and LRMS constitute the hierarchical nature of the grid.

- Resources shared in the computational grid are commonly owned and controlled by

[4.1]Grid Resource Management System
[4.2]Local Resource Management System

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



different individuals or organizations in different sites. Administrators of these sites decide which resource to share and how to share. Therefore, grid users may meet different scheduling policies and security mechanisms when using resources shared in the Grid.

- Different sites may have different types of resources. Even the resources of the same type, located at different sites, may have different configurations, capacities and performance.
- Grid resources distributed all over the world linked in the Internet. In a grid system, the network links among different resources usually have widely varying performance characteristics. Furthermore, the inter-resource connection is by one or two orders of magnitude slower than the intra-resource connection.

In this section, we design an HPGA for the PFP on computational grids. Here, we are especially interested in hierarchical grid architectures. The hierarchical grid computing describes the combination of several PC clusters within one architecture. Computing resources located in geographically distributed clusters are shared in this environment. Grid infrastructures, such as the Globus Toolkit [3], are installed on the head nodes of these resources. Inside each cluster, the resource is managed by the local resource management system. Using PC clusters as in the Beowulf approach is currently one of the most efficient and simple ways to gain high compute power at a reasonable price. One such application example is shown in [102]. The development or adaptation of parallel applications for the hierarchical grid architecture is made challenging by the often heterogeneous nature of the resources involved. A typical characteristic of this architecture is the large gap between the fast connection inside a cluster and the slow connection between clusters. Thus applications designed for uniform speed interconnections will lead to performance degradation on the computational grid environment.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



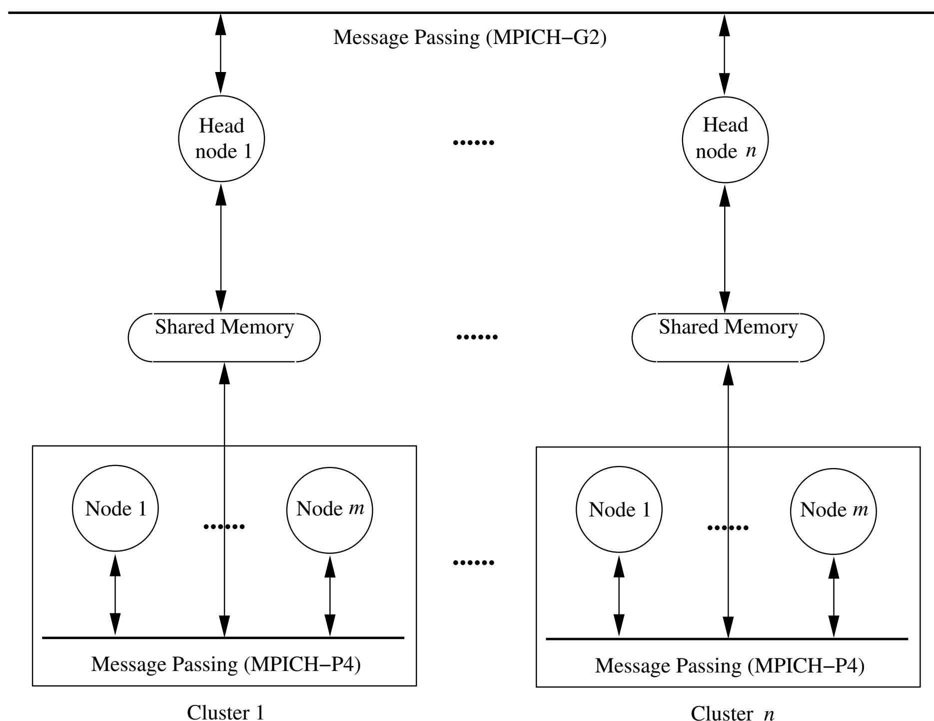



Figure 4.22: The structure of the two-layer architecture

In order to efficiently map our HPGA onto the hierarchical grid, different MPI libraries are used in the two layers. In the grid layer, MPICH-G2 [3] based processes run on the head node, it migrates the subpopulations in the local cluster to other clusters within the grid environment. Inside the cluster, MPICH-P4 [9] is used to transfer data between different nodes. Subpopulations can be exchanged between the grid layer and the cluster layer by reading and writing to shared memory areas on the head node of each cluster. Figure 4.22 shows the general structure of our two-layer architecture.

MPICH [9, 92] is a popular implementation of MPI-1 [82] standard with extensions to support the parallel I/O functionality defined in the MPI-2 [81] standard. Its free distribution and wide portability have contributed materially to the adoption of the MPI standard by the parallel computing community. MPICH achieves its portability from its interfaces and layered architecture. At the top of the layered architecture is the MPI interface defined by the MPI standards. Directly beneath the interface is the MPICH

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



layer, which implements the MPI interface. Much of the code in MPICH is independent of the networking devices or process management systems.

MPICH-G2 [92, 8] is a implementation of the complete MPI-1 [82] standard and MPI-2 [81] I/O standard. It hides heterogeneity of grid by using Globus Toolkit services for such purpose as authentication, authorization, executable staging, process creation, process monitoring, process control, communication, redirection of standard input and output, and remote file access, while also allowing for application management of heterogeneity. As a result a user can run MPI programs across multiple computers at different sites using the same commands that would be used on a parallel computer.

Before the startup of an MPICH-G2 application, the user need to employ the *Grid Security Infrastructure* (GSI) [62] to obtain a (pubic key) proxy credential that is used to authenticate the user to each site. The user may also use the *Monitoring and Discovery Service* (MDS) [60] to select computers on the basis of, for example, configuration, availability, and network connectivity.

Once authenticated, the user can use the standard `mpirun` command to request the creation of an MPI computation. The MPICH-G2 implementation of this command uses the *Resource Specification Language* (RSL) [48] to describe the job. A RSL script may contain the description of resources (e.g., computers) and requirements (e.g., number of CPUs, memory, execution time) and parameters (e.g., location of executables, command line arguments, environment variables). Based on the information found in an RSL script, MPICH-G2 calls the Dynamically-updated Request Online Co-allocator (DUROC) [49] to schedule and start the application across the various computers specified by user. The DUROC library itself uses the *Grid Resource Allocation and Management* (GRAM) [48] API and protocol to start and subsequently manage a set of subcomputations. Once an application has started, *Globus Access to Secondary Storage* (GASS) [31] will be used to direct standard output and error steams to the user's terminal and provide access to files

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



regardless of location, Thus masking essentially all aspects of geographical distribution except those associated with performance.

On running of the application, MPICH-G2 selects the most efficient communication method possible between any two processes, using vendor-supplied MPI if available, or *Globus communication* (Globus IO) with *Globus Data Conversion* (Globus DC) for TCP/IP, otherwise.

MPICH-G2 is not the only implementation of grid-enabled MPI. Others includes PACX-MPI [69], Stampi [91] and MagPIe [94], etc. However, MPICH-G2 is the most widely used one due to its performance and the degree to which it hides and manages heterogeneity of grid environments. Many applications have achieved good performance by using MPICH-G2 [41].

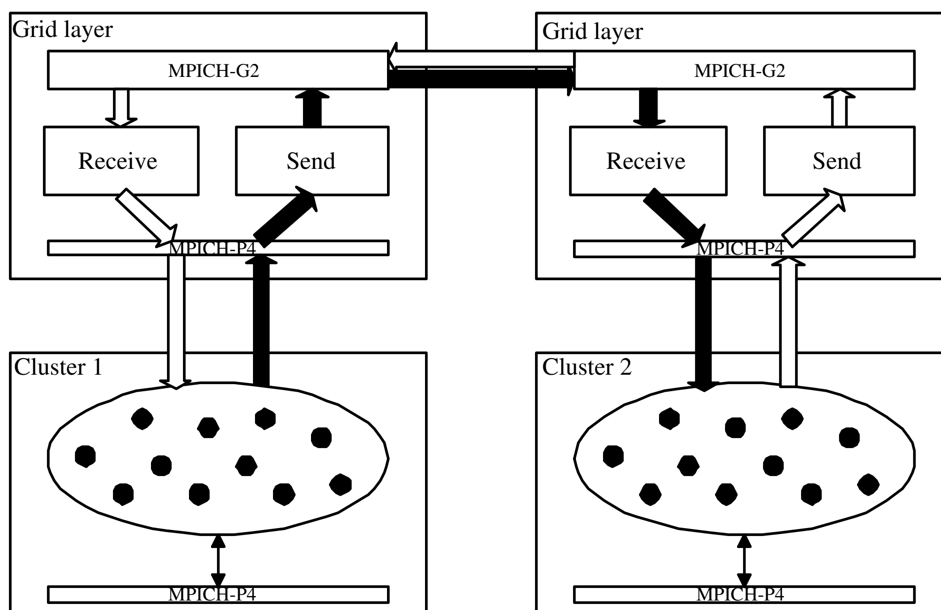



Figure 4.23: The communication detail of the two-layer architecture.

Figure 4.23 shows the communication detail of our two layer architecture. Two shared memory blocks are used on the head node of each cluster. Within each cluster, all nodes can exchange individuals through MPICH-P4 and send a portion of subpopulations to

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



the *send* memory block. When there are individuals in the *receive* block, MPICH-P4 will transfer them to each node inside the cluster. The head node of each cluster will migrate individuals in the *send* memory block to other clusters. Also, these head nodes will move individuals come from outside to the *receive* memory block.

## 4.4 Summary

In this chapter, we have proposed corresponding partitioning and communication schemes according to the characteristic of two CB problems. For DP algorithms, a tunable coarse-grained algorithm is introduced. By putting the residues to the forefront of the matrix and using asynchronous communication mode, our algorithm can work efficiently for regular and irregular DP algorithms. We also have designed an HPGA for the PFP on computational grids. By combining the inter-cluster parallelism with the intra-cluster parallelism, our HPGA can achieve super-linear speedups.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Chapter 5

# A Generic Parallel Pattern-based Framework for Computational Biology Algorithms

## 5.1 Introduction

In Chapter 3 and 4, we have analyzed and studied characteristics of two popular CB algorithms in the *algorithm space*. We have proposed corresponding parallel partitioning and communication schemes. If we view the whole development procedure for an HPC application as activities that derive a solution structure from an algorithm space, it isn't enough just to understand the algorithm itself. In this chapter, we present the implementation detail of a framework in the *implementation space*. We have used the standard C++ programming language [15] to implement this framework.

Figure 5.1 summarizes the whole procedure of our design and implementation. We have started by analyzing characteristics of two popular CB algorithms. Next, we have proposed corresponding parallel algorithms and communication schemes. At last, the algorithm space and the implementation space are combined to yield the framework at the bottom part in Figure 5.1.

According to [46], a *framework* is a software package that captures the software archi-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



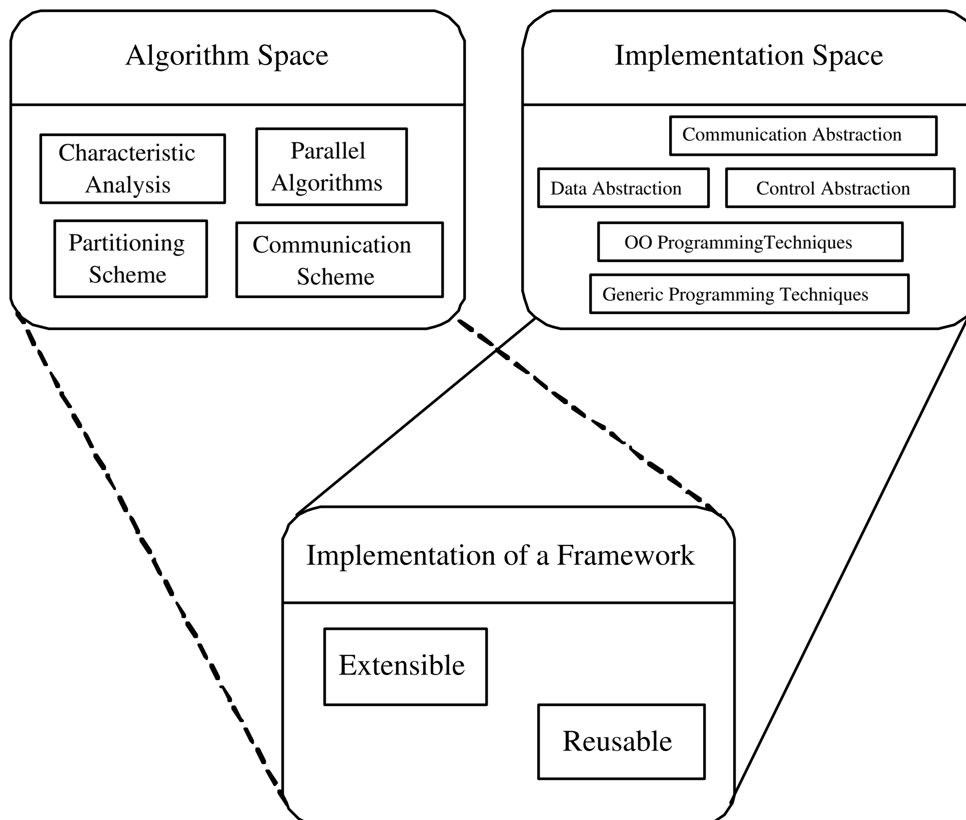



Figure 5.1: Summarization of all the activities: from the *algorithm space* to the *implementation space*.

tecture in source and object code. It is a partly filled-out implementation, with "holes" to be filled in on a per-application basis. Therefore frameworks are an abstraction and characterize a family of related implementations. Object-oriented frameworks have become increasingly popular in the literature and industry dialogue [67]. There has been notable success in some frameworks such as the STL (the Standard Template Library, which later was adapted and incorporated into the C++ standard library) [111].

## 5.2 Design and Development of Our Framework

In Chapter 2, some parallel pattern-based systems are introduced. These systems were built with the intention to facilitate the development of general parallel applications.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



However, they lack practical usability for the CB area. This is mainly because most of these systems support only a limited set of patterns in a stiff way. There is no generic model describing the structure and behavior of a pattern. Lack of genericness is often at the root of the lack of extensibility.

In this section, we present the design and development of a new parallel pattern-based framework. An important aspect of the framework is the generic representation for a set of patterns, i.e. a generic pattern. With this generic pattern, the framework provides users good extensible and reusable mechanisms.

### 5.2.1 Multi-Paradigm Design for High Performance Computing

The term *paradigm* was originally popularized by Kuhn [98], and has much broader meaning than we find in computer science. Kuhn defines paradigms as "universally recognized scientific achievements that for a time provide model problems and solutions to a community of practitioners". It is a deeper and broader concept than any notion of paradigm in contemporary computer science. Wegner extends the notion to programming language paradigms [148]. It is defined to shape the way we formulate abstractions. In practice, a paradigm encodes rules, tools, and conventions to partition the world into pieces that can be understood individually.

C++ is a programming language that supports Object Oriented Programming (OOP) paradigms and Generic Programming (GP) paradigms such as: classes, overloaded functions, inheritances, virtual functions, templates, and others [138]. OOP groups classes into hierarchies that reflect commonality in the structure and behavior, while at the same time allowing for regular variations in the structure and in the algorithm that implements a given behavior. GP comes from traditional OOP, the special features of template techniques make it different from traditional OOP. GP paradigm deals with finding abstract representations of algorithms, data structures, and other software concepts [144]. Because

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



of its good flexibility, extensibility and security, GP techniques are very suitable for the development of pattern based systems. The STL [111] and Janus [75] are two examples of GP applications.

There's no denying the vast benefits of OOP but the OOP is just one subset of the implementation space and not always appropriate for the problem at hand. Furthermore, HPC programming exhibits a rich multiplicity: You can do the same thing in many correct ways, and there are infinite nuances between right and wrong. The design of an HPC framework is a choice of solutions out of a combinational implementation space. Neither OOP nor GP in the form of templates will create a flexible design method to handle the possible combinational explosion; using multi-paradigm techniques, that is, combining the two paradigms together, will be an efficient way.

For example, if we find that framework members share common data structures, we look for a way to express that in C++ and find that inheritance fits the bill. If we find that framework members share the same interface, with each one implementing it differently, we might use inheritance with virtual functions. If we find common code structure and behavior but variability in the interface, we might use templates. This process may repeat recursively.

### 5.2.2 Our Parallel Pattern-based Framework: Overview

In this section, we demonstrate the implementation of a parallel pattern-based framework for parallel CB algorithms presented in Chapter 4. A parallel pattern does not represent a single solution to a given problem, but rather embodies a family of potential solutions. To incorporate the idea of patterns as families of solutions, we use multi-paradigms in C++ to construct an intermediate abstract framework of parallel patterns. The framework code defines virtual functions for the sequential application code as well as the corresponding parallel structure.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



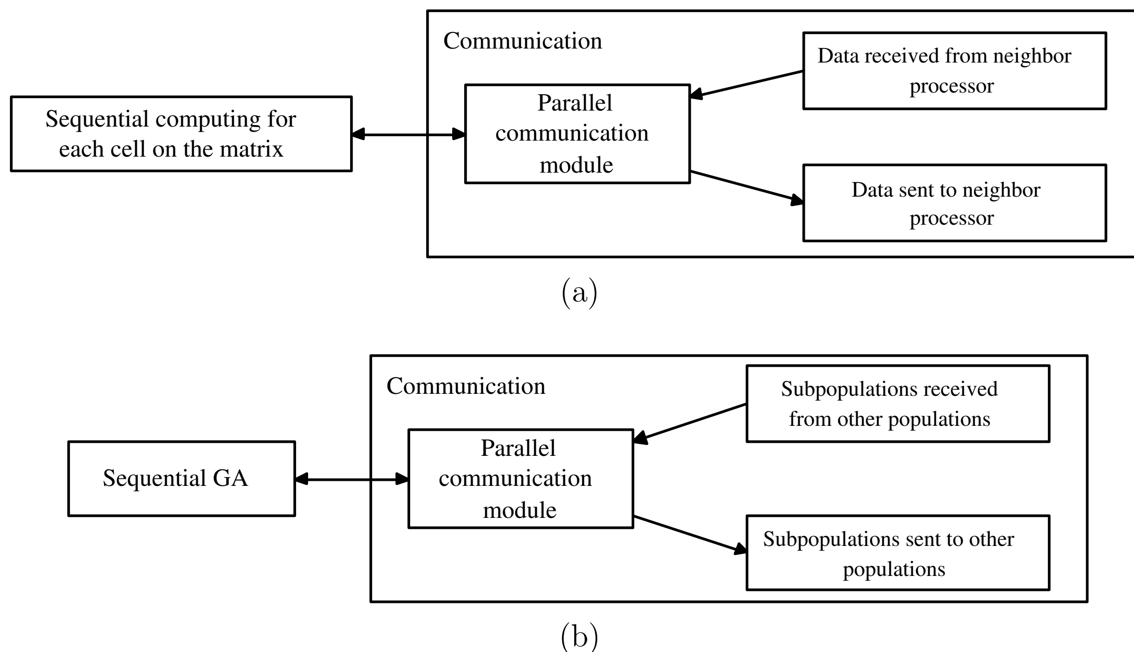


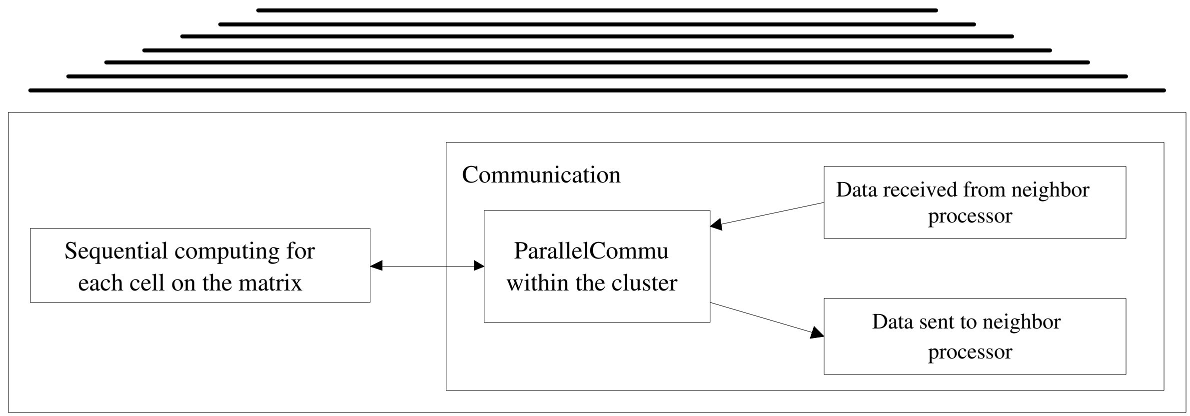

Figure 5.2: The two parts of parallel algorithms for (a) Parallel DP algorithms, (b) HPGAs

According to the characteristics of the parallel algorithms shown in Chapter 4, we have divided parallel DP algorithms and the HPGA into two parts: the sequential computing part and the communication part (see Figure 5.2). The sequential computing part deals with the local behavior of a parallel algorithm, such as the computation of each element on the matrix in DP, the mutation, variation and crossover in GAs. The communication part processes the communication behavior between all processors participating in the parallel algorithm. By dividing parallel algorithms in such a way, both parts of a parallel algorithm can evolve independently. This allows for the rapid prototyping of parallel programs and facilitates the mapping of them onto parallel architectures.

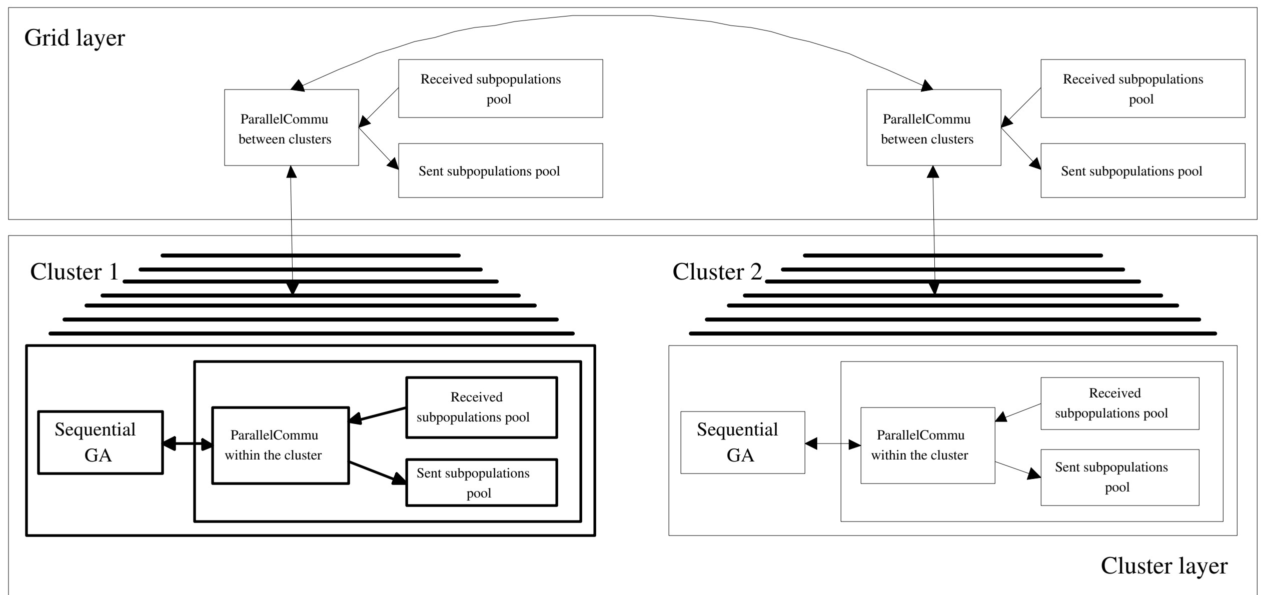
Figure 5.3 shows our way to map DP algorithms and the HPGA onto parallel architectures. Two kinds of parallel communication modules (ParallelCommu), the module within the cluster and the module between clusters, are used to implement the intra-cluster and the inter-cluster communication separately. For the HPGA, the high level

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library






(a)




(b)

Figure 5.3: (a) Mapping of parallel DP algorithms onto a cluster, (b) Mapping of HPGAs onto the computational grid environment

part of it is mapped onto the grid layer and the low level part is mapped onto the cluster layer.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



### 5.2.3 Development of An Extensible and Reusable Framework

Traditional OOP techniques provide a good environment for code reuse, such as the techniques of inheritances and virtual functions. For example, we can use pure OOP techniques to design the framework for parallel wavefront pattern as shown in Figure 5.4. The virtual function `parallelcommunication()` contains a loop that iterates over all matrix elements in the partition and invokes the `sequentialcomputation()` method on each element.

```
class Wavefront{
Public:
    void launch(){
        preprocess();
        while(the cyclic loop is not completed){
            parallelcommunication(){
                ...
                sequentialcomputation();
                ...
            }
        }
        postprocess();
    }
    virtual void preprocess();
    virtual void postprocess();
    virtual void parallelcommunication();
    virtual void sequentialcomputation();
};
```

Figure 5.4: Pure OOP design for the parallel wavefront pattern.

By inheriting the base class, and overriding virtual functions, new parallel applications can be implemented. In order to reuse components in the base class, users need to construct other derived classes for specific applications. That is, the code reuse in traditional OOP is lengthways. If some common parts among virtual functions need to be changed for specific applications, users have to rewrite them from scratch. For instance,

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



the data type `int` and `MPI_INT` may be used in the base class `Wavefront`. However, the data type `float` and `MPI_FLOAT` are needed in the derived class `SmithWaterman`. In this case, users have to override all data type-related codes. This is a time-consuming and error-prone work. Figure 5.5 illustrates how to use available patterns to develop new applications in traditional OOP.

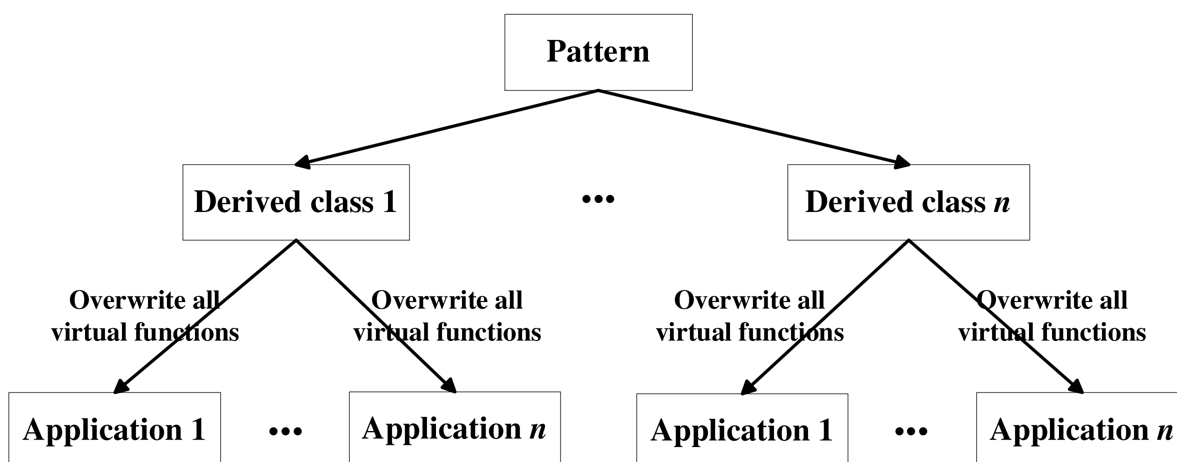



Figure 5.5: Using inheritance and overriding virtual functions to develop new applications in traditional OOP.

GP techniques provide a new mechanism for code reuse. The code reuse in GP is obtained by template specialization rather than inheritance. It is much more flexible than OOP. For example, we can use the code in Figure 5.6 to construct the connection between the parallel communication part and the sequential code, i.e. the relationship between the data type `int` and the data type `MPI_INT`. Here, we use the partial specialization technique to specialize the class template `MPItype<class datatype>` with `int` (line 2). Similarly, other data types can be specialized in this way. Thus, the program can automatically identify which MPI type will be used according to the data type specified by the user (lines 4, 5).

This kind of code reuse in GP is transverse and can be used to solve multi-strategy problems efficiently. The multi-strategy problem is that a problem computation routine

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



```
1. Template<class datatype> class MPItype;
2. Template<>class MPItype<int>{enum{TypeID=MPI_INT};};
3. ...
4. Template<class datatype> class Application{
5.         mpitype=MPItype<datatype>::TypeID;}
```

Figure 5.6: The connection between the sequential data type and the MPI data type.

is determined by many parameters. Each parameter has different possible values or types. Most computational problems are multi-strategy. This is because computation routines of them are usually determined by many associated factors. It is very difficult for traditional OOP techniques to implement the code reuse for multi-strategy problems. This is because the combination explosion of all the possible values of different parameters can make this implementation by OOP very difficult. GP techniques can make it easily for that the compiler will automatically determine the specific computation routine. That is GP provides a higher level abstraction than traditional OOP. It makes use of the power of compiler to generate the specific program routine automatically.

The code in Figure 5.7 shows the general structure of our framework. Concrete parallel applications can be implemented by extending and instantiating the template parameters of `GenericPattern`. These parameters encapsulate the abstract structure and behavior of a set of parallel patterns in an application independent manner. The parameter `AlgorithmIni` initializes some parameters and defines the data structure of the communication message. `SequentialComp` processes the sequential computation. `ParallelCommu` consists of the communication behavior between all processors participating in the parallel computation. By defining these three parameters, the parallel part (`ParallelCommu`) is separated from the sequential application parts (`AlgorithmIni` and `SequentialComp`). Thus, both parts of a parallel application can evolve independently. This allows for the rapid prototyping of parallel applications, and permits users to experiment with alterna-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



```
template<class datatype,
         class AlgorithmIni,
         class SequentialComp,
         class ParallelCommu>
class GenericPattern{
    virtual void HpcComputing(){
        AlgorithmIni::PreProcess();
        while the cyclic loop is not completed{
            ...
            /*For parallel DP applications, SequentialComp is
            invoked in the following method*/
            ParallelCommu::Launch();
            ...
            /*For parallel GAs applications, ParallelCommu is
            invoked in the following method*/
            SequentialComp::Launch();
        }
        AlgorithmIni::PostProcess();
    }
};
```

Figure 5.7: The structure of class template `GenericPattern`.

tive communication structures conveniently. Currently, we have integrated the tunable coarse-grained partitioning and communication scheme for DP algorithms into the framework. As for the HPGA, the island and stepping stone communication models have been pre-implemented. Figure 5.8 is the UML class diagram for our framework.

An important aspect of our framework is the generic representation for a set of patterns, i.e. a generic pattern. With this generic pattern, we mainly focus on the extensibility of the framework rather than how many limited patterns it can support.

Figure 5.9 and 5.10 are UML class diagrams that show extensions of the `GenericPattern` to develop parallel programs for DP algorithms and GAs. The user only needs to specify the relevant template parameters according to the characters of the algorithms. In

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



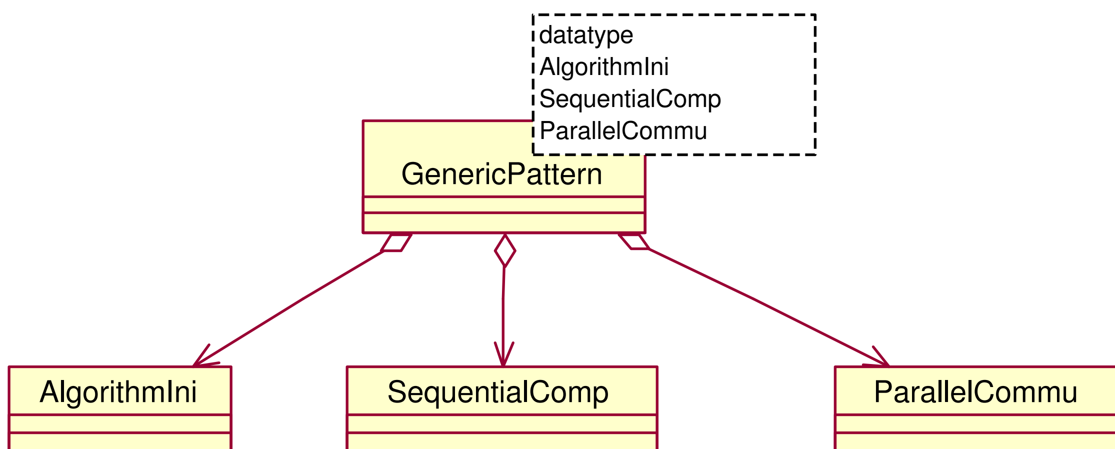



Figure 5.8: The UML class diagram for `GenericPattern`

GP techniques, this extension is also called the template specialization. Our framework provides users with some predefined, efficient and reusable components for HPC programming, thus relieving users of the need to rebuild all the error prone parts that are common in parallel and distributed program code.

From Figure 5.9 and 5.10 we can see that users can extend the generic pattern by specifying the application-dependent template parameters. Different specialization will lead to different policies, i.e. different implementation strategies for a concrete parallel application. A policy can be further instantiated in order to generate the concrete parallel program (see Figure 5.11). Each template parameter is defined independently from other parameters. Yet different template parameters can interact with each other via standard interfaces. Consequently, the framework has a good flexibility. In Figure 5.11, policy3 and policy1 share the same parallel characters. Thus we can entirely reuse the overall design of `ParallelCommu1` to develop policy3. The user can therefore reuse the components in existing patterns to develop new applications in a flexible way.

For instance, two groups of DP algorithms: the Nussinov algorithm and the MCOP; the Smith-Waterman algorithm with the linear and the affine gap penalty, algorithms in

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



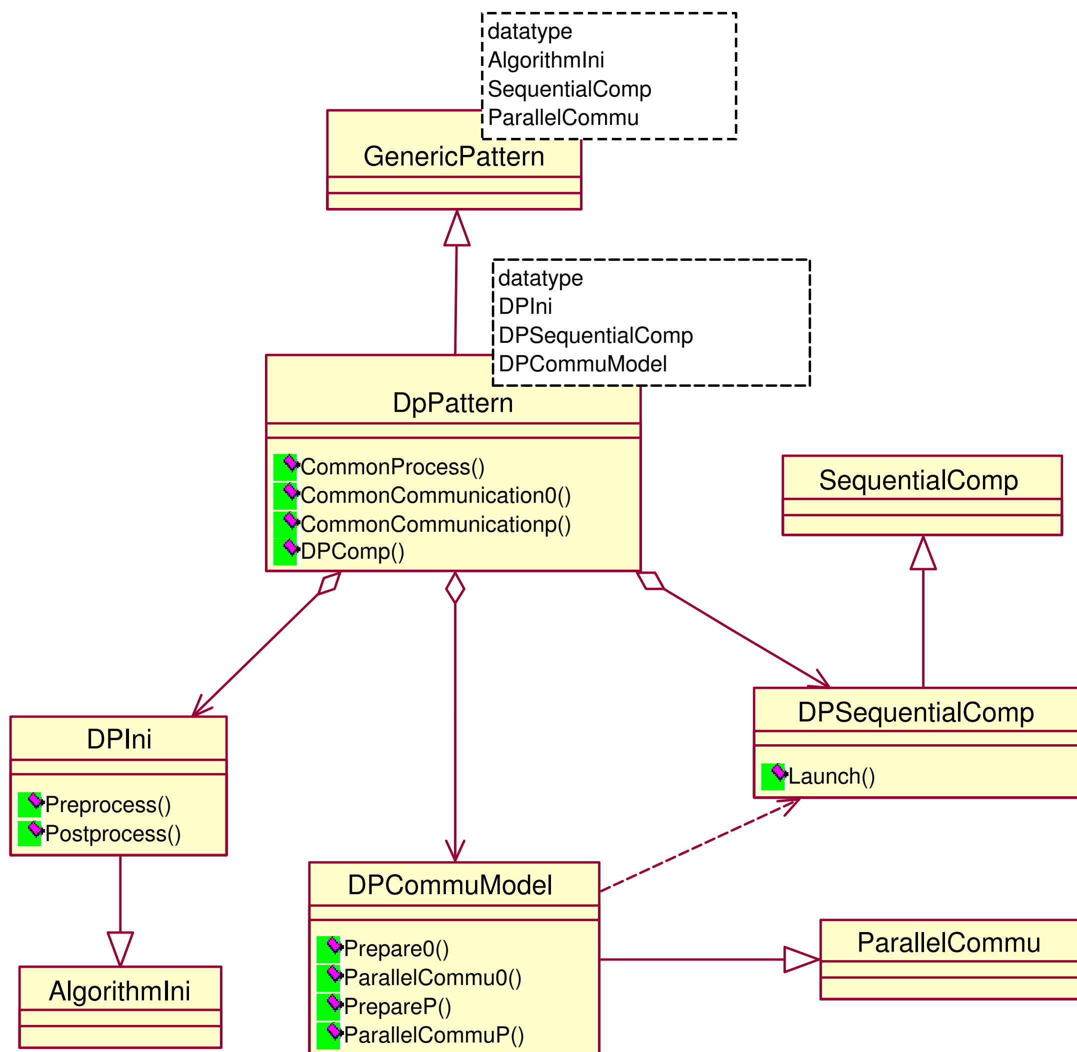



Figure 5.9: The UML class diagram of an extension of the `GenericPattern` to implement parallel DP programs

each group share same parallel characteristics. Thus we can reuse the same component `ParallelCommu` to develop parallel applications for them. As to the HPGA for the PFP, we can reuse existing communication patterns in the cluster level and the grid level, such as the stepping stone model in the cluster level and the island model in the grid level or vice versa, to generate new HPC applications.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



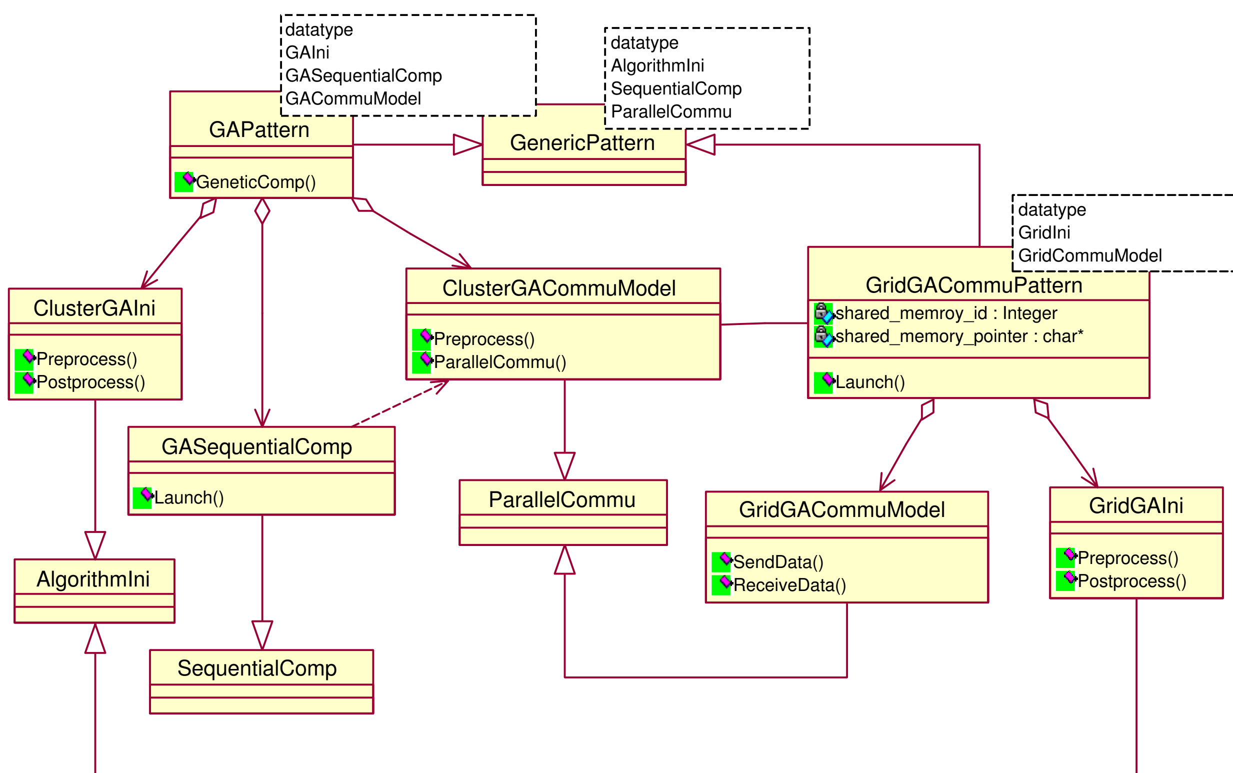



Figure 5.10: The UML class diagram of an extension of the `GenericPattern` to implement HPGA programs

In summary, compared to other pattern-based systems mentioned in Chapter 2, our framework has the following advantages:

(1) CB related parallel patterns, such as the parallel wavefront pattern and the parallel GA pattern, have been integrated into the framework.

(2) It has good extensibility and reusability. Users can reuse the components in the framework to develop new applications in a flexible way. The integration of new parallel patterns is also allowed under the extensible mechanism of the framework.

(3) It supports computational grid oriented CB applications. With the increased availability of grid computing platforms, grid-enabling of pattern-based systems are of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



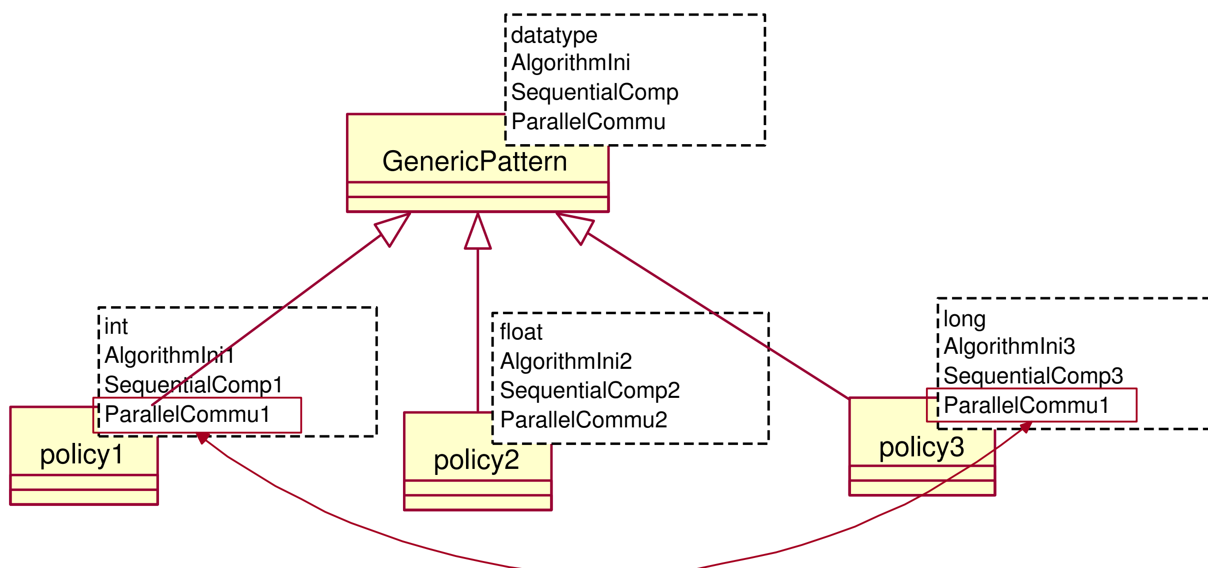



Figure 5.11: Reusability of components in the framework for different policies

high importance to the future research in the high-performance CB area.

## 5.3 Performance Evaluations

### 5.3.1 Experimental Results for Parallel Dynamic Programming Algorithms on PC Clusters

We have implemented the framework using standard C++ [15] and the MPI library provided by MPICH [9, 8]. We have used it to develop parallel applications for several DP algorithms and evaluated them on two different parallel architectures. One is a SMP Alpha cluster with Tru64 UNIX installed. It comprises eight ES45 nodes. Each node contains four Alpha-EV68 1GHz processors and 16GB RAM. Each processor has 8 MB L2 cache. All the nodes are connected with each other by a Gbit/sec quadrics switch. Another test bed is a Beowulf PC cluster with Linux 7.3 installed. It contains sixteen Intel Pentium IV Xeon 2.6GHz processors, 512MB RAM and 512KB cache for each processor. All the processors are connected with each other by a Gbit/sec Myrinet switch.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Since the framework has been implemented using multi-paradigm techniques (GP and OOP), it is interesting to compare its performance with a framework implemented only using OOP techniques. In order to investigate this, we have implemented the framework in both ways. Table 5.1 presents the performance comparison for the sequential applications between these two frameworks.

Table 5.1: Performance comparison between two frameworks(framework1: only using OOP techniques; framework2: using GP and OOP techniques)

| **Sequential Applications** | **Runtimes of Framework1(sec)** | **Runtimes of Framework2(sec)** | **Runtime Reduction** |
|---|---|---|---|
| Smith-Waterman with linear gap penalty (100000×100000) | 551 | 503 | 8.7% |
| Smith-Waterman with affine gap penalty (100000×100000) | 917 | 868 | 5.3% |
| Syntenic alignment algorithm (100000×100000) | 1543 | 1319 | 14.5% |
| Smith-Waterman with general gap penalty (5000×5000) | 1262 | 1217 | 3.6% |
| Skyline matrix (5000×5000) | 1761 | 1708 | 3% |
| MCOP (5000×5000) | 965 | 926 | 4% |
| Nussinov (5000×5000) | 821 | 795 | 3.2% |
| Arbitrary Viterbi (700×700) | 1396 | 1371 | 1.8% |
| Spliced alignment (4000×4000×1000) | 730 | 642 | 12.1% |

From Table 5.1 we can see that the code developed by the framework using multi-paradigm techniques is faster. This is because the GP relies on static polymorphism, which resolves interfaces at the compile time. On the other hand dynamic polymorphism uses inheritance and virtual functions, the determination of which virtual function to use cannot be made at the compile time and must instead be made during the run time. Thus, more overhead is associated with the invocation of a virtual function. In this thesis, we have used HPC applications developed by the framework using multi-paradigm techniques to do all the following experiments.

First, we have run all HPC DP applications on the Beowulf cluster to see their

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



speedups. The speedup of a parallel application is a well-accepted way of measuring its performance. According to the conventional definition, the speedup can be defined as the ratio of the execution time of the sequential algorithm, $T_S$, and the execution time of the parallel program, $T_P$ [22]. So, we can compute the speedup as:

$$S_P = \frac{T_S}{T_P} \tag{5.3.1}$$

Table 5.2: Speedups on the Beowulf cluster for several DP algorithms with corresponding matrix size.

| **Number of Processors** | **2** | **4** | **8** | **16** |
|---|---|---|---|---|
| SW with linear gap penalty(90000×90000) | 2 | 3.99 | 7.76 | 14.16 |
| SW with affine gap penalty(70000×70000) | 1.92 | 3.89 | 7.58 | 13.96 |
| Syntenic alignment(60000×60000) | 2.02 | 3.89 | 7.93 | 15.79 |
| SW with general gap penalty(5000×5000) | 1.86 | 3.76 | 7.32 | 13.43 |
| Skyline Matrix(5000×5000) | 1.92 | 3.6 | 7.13 | 13.59 |
| MCOP(5000×5000) | 1.97 | 3.91 | 7.46 | 13.43 |
| Nussinov(5000×5000) | 1.93 | 3.87 | 7.4 | 13.65 |
| Arbitrary Viterbi(700×700) | 1.97 | 3.85 | 7.23 | 13.14 |
| Spliced(4000×4000×1000) | 2 | 4.04 | 8.12 | 16.2 |

Table 5.2 shows best speedups for different number of processors using the tunable coarse-grained partitioning and communication scheme on the Beowulf cluster. For irregular DP applications, the *division* is set from 1 to 150 and the *rowwidth* is set from 5 to 50. It is important to note that these applications are implemented using different methods. The linear space method is used to reduce the RAM needed by the Smith-Waterman algorithm (with the linear gap penalty and the affine gap penalty) and the Syntenic alignment algorithm for long sequences. Similar space-saving methods are used for the spliced alignment algorithm. As for the Smith-Waterman algorithm with the general gap penalty, the skyline matrix, the matrix chain, the arbitrary Viterby, and the Nussinov algorithm, we store and compute the whole matrix. From Table 5.2 we can see that speedups for the spliced alignment algorithm are super-linear. This is because

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



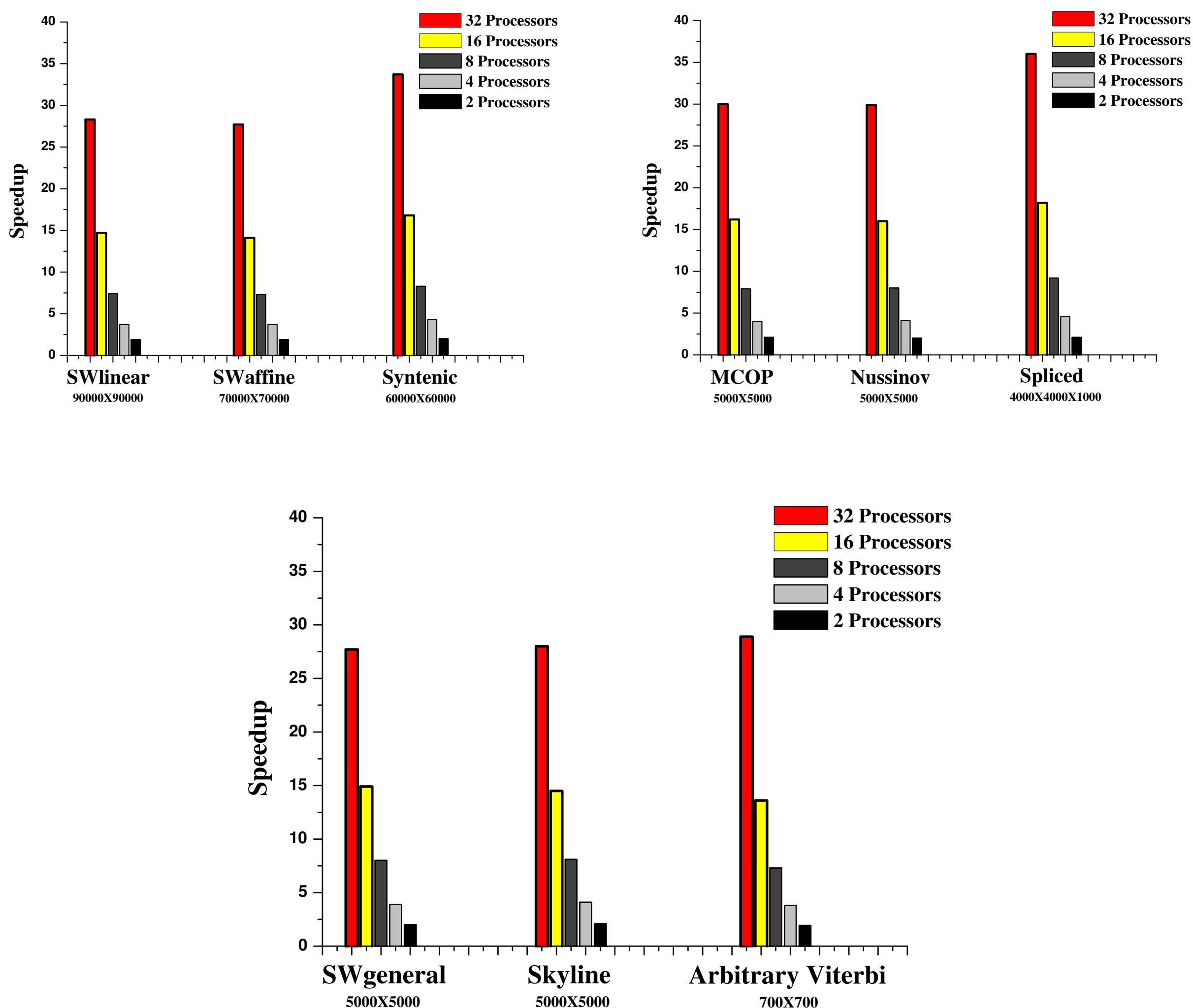



Figure 5.12: Speedups on the Alpha cluster for several regular and irregular DP CB algorithms with corresponding matrix size.

when more processors are put into the parallel computation, more high-speed caches will be used in the data communication and access. These caches can improve the hit rates of processors and thus causes extra performance increase. As to the other regular DP algorithms, the size of the message for communication is quite small and so the impact of caches is not significant.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Then, we have run all parallel DP applications on the Alpha cluster. Figure 5.12 shows the best speedups for different number of processors when *division* is set from 1 to 150 and *rowwidth* is set from 5 to 50. Notice the super-linear speedups are observed in some applications. This is because of the effects due to better caching.

Table 5.3: Speedup comparison using different *division* $d$ and *rowwidth* $r$ for the Nussinov algorithm. The matrix size is 5000×5000. The number of processor is 32

| | $\boldsymbol{r = 5}$ | **10** | **15** | **20** | **30** |
|---|---|---|---|---|---|
| $\boldsymbol{d = 1}$ | 10 | 10.5 | 9.8 | 9.6 | 8.5 |
| **40** | 22.2 | 24.5 | 21.9 | 21.2 | 20.6 |
| **60** | 26.2 | 27.5 | 26.7 | 25.2 | 24.9 |
| **65** | 27.7 | **29.8** | 26.9 | 26.4 | 25.7 |
| **70** | 24.2 | 26.2 | 22.5 | 22.2 | 19.9 |
| **75** | 24.2 | 26.1 | 23.3 | 21.6 | 21.5 |

Table 5.4: Speedup comparison using different *division* $d$ and *rowwidth* $r$ for the Smith-Waterman algorithm with the general gap penalty function. The matrix size is 5000×5000. The number of processor is 32

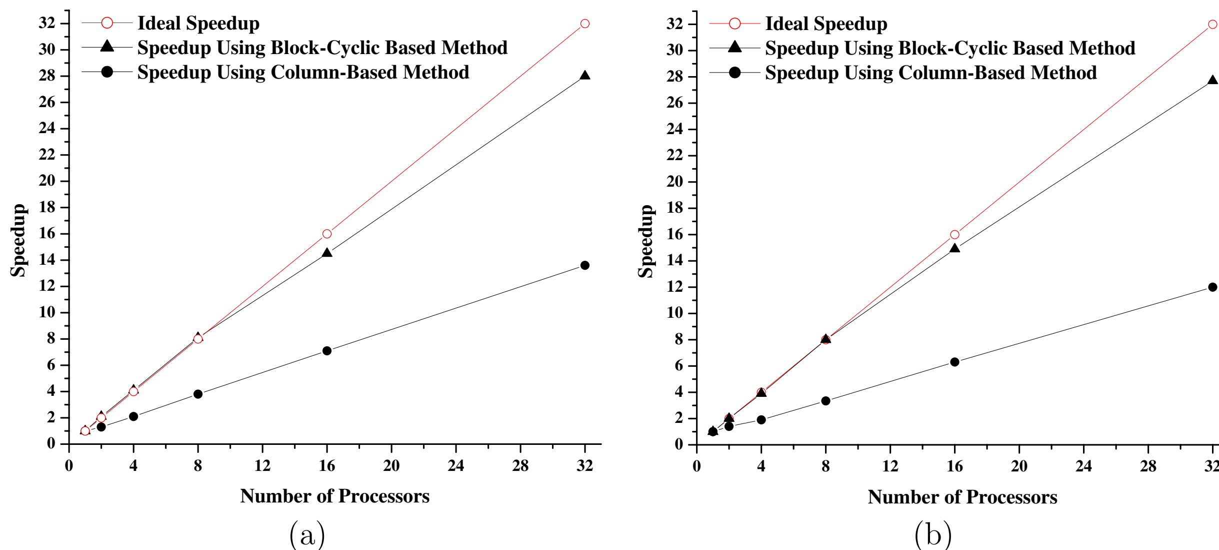
| | $\boldsymbol{r = 5}$ | **10** | **15** | **20** | **30** |
|---|---|---|---|---|---|
| $\boldsymbol{d = 1}$ | 10.5 | 12.8 | 11.3 | 10.1 | 8.5 |
| **40** | 20.4 | 23.6 | 21.3 | 20.6 | 19.1 |
| **60** | 22.1 | 24.9 | 22.2 | 21.2 | 19.4 |
| **65** | 25.8 | **27.9** | 26.2 | 25 | 23.4 |
| **70** | 23.2 | 26.1 | 23.2 | 22.3 | 21 |
| **75** | 23 | 24.7 | 21.9 | 20.7 | 20.3 |

For irregular DP algorithms we have observed the impact of *division* and *rowwidth* to get the optimal block size for these applications. Two examples in Table 5.3 and 5.4 show the average speedups using different *division* and *rowwidth*. From these two tables we can see that the best speedups are obtained when *division* is set around 65 and *rowwidth* is set around 10 for these two DP algorithms. That is, in order to get the best performance, the choice of these two parameters is a trade-off between the load balancing and communication overhead.

Also, we have compared the performances between the block-based partitioning scheme

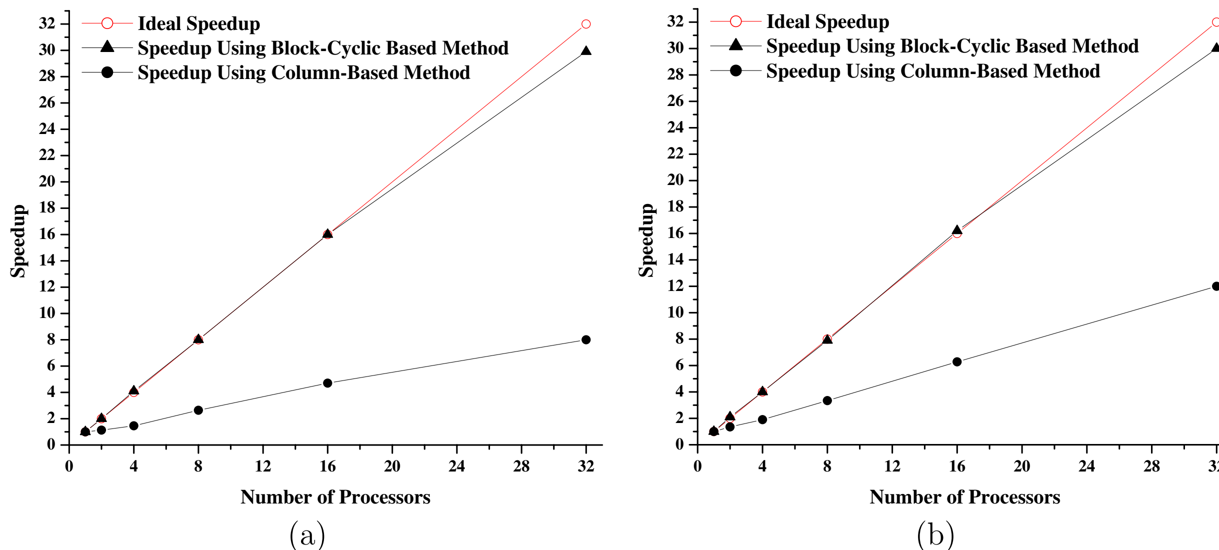
ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library






(a) (b)

Figure 5.13: Speedups of irregular DP applications using different partitioning schemes for (a) the skyline matrix problem, (b) the Smith-Waterman algorithm with general gap penalties.




(a) (b)

Figure 5.14: Speedups of irregular DP applications using different partitioning schemes for (a) the Nussinov algorithm, (b) the MCOP problem.

and the tunable coarse-grained partitioning scheme. Figures 5.13, 5.14 and 5.15 show the speedups of irregular DP applications using different partitioning schemes. We can see

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



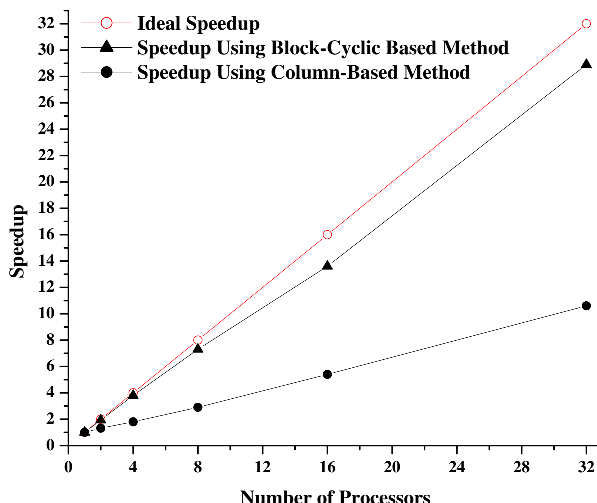



Figure 5.15: Speedups of irregular DP applications using different partitioning schemes for the arbitrary-order Viterbi algorithm.

that the results using the tunable coarse-grained partitioning scheme are much more better due to better load balancing.

### 5.3.2 Experimental Results for Parallel Genetic Algorithms

Two test beds are used in experiments for our parallel GAs. One is the Alpha cluster. Another test bed is a heterogeneous cluster environment. It contains four high-performance clusters. These clusters contain 8 Intel Xeon 2.6 GHz, 8 Intel Xeon 3.0 GHz, 8 Intel Pentium 731MHz and 8 Intel Itanium 733MHz respectively. A 1Gbit/sec Myrinet switch connects each cluster internally and a 100Mbit/sec Ethernet switch is used as an inter-cluster connection. Globus Toolkit™ is installed on the head node of each cluster. Ganglia and Globus MDS [3] are used to collect and present resource information for users.

According to Eq. (5.3.1), the speedup of a parallel GA can also be defined as the ratio of $T_S$ and $T_P$. Because $T_P$ can be further described as the sum of the time used by one subpopulation in the computation ($T_{COMP}$) and the time it used to communicate

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



information to its neighbors ($T_{COMM}$), we can compute the speedup as:

$$S_P = \frac{T_S}{T_P} = \frac{T_S}{T_{COMP} + T_{COMM}} \tag{5.3.2}$$

Although speedup is very common in the deterministic parallel algorithms field, in the GA community the topic of parallel speedups has raised significant controversy. The main reason is that the execution time of the serial and parallel GAs are compared without considering the quality of the solutions found in each case. Because of the limited number of individuals and the inherited selection, the sequential GA has much more tendency to be trapped in a local minimum, without enough genetic diversity to help itself out. Communicating individuals between different evolutions in parallel GAs can help in keeping the genetic diversity of the population, thus greatly reducing the probability to be trapped into local minimum. Assuming the GA is seeking the maximum of some fixed real-valued function, the parallel GA will have an unfair advantage to work out the result much faster. In fact, many researchers have achieved super-linear speedups when using a parallel GA. Shonkwiler [131] has proven this theoretically and provided the following formula to compute the speedup for parallel GAs:

$$S_P = P \times S^{P-1} \tag{5.3.3}$$

In Eq. (5.3.3) he introduced an acceleration factor $S$ that can explain the super-linear speedup ($S > 1$), $P$ is the number of processors. It is also shown that for the "deceptive" problem where the time to reach the goal can be infinite, very large speedups are possible. Eq. (5.3.3) provides us a precise way to explain and predict the speedups for parallel GAs. However, the computation of the acceleration factor $S$ is too complex. And because for any time $T$, there is a non-zero probability that the algorithm will take time $T$ to work out the final result, in order to get the expected speedups by Eq. (5.3.3),

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



the user may wait a very long time $T$ [131]. This is not feasible in practice. During an experiment, if the time to reach the goal tends to be infinite, we think the algorithm is trapped into local minimum. We therefore define the *hit rates* of a running parallel GA as the percentage of the number of outcomes achieved in finite time relative to the total number of experiments. With the concept of *hit rates*, we introduce an approximate way to compute the speedups for parallel GAs:

$$S_P = P \times (k \times \frac{H_P}{H_S}) \tag{5.3.4}$$

where $H_S$ and $H_P$ are the hit rates for the sequential and parallel GAs respectively. $k$ is an algorithm-dependent constant. In our experiments we have chosen the value of $k$ to be 1.

The following sequence with the length 64 has been used in the experiments: *hh-hhhhhhhhhhphphpphhpphhpphpphhpphhpphpphhpphhpphphhhhhhhhhhhh*. This sequence is based on the hydrophobic-hydrophilic (HP) model. The HP model reduces a protein instance to a string of $h$'s and $p$'s that represents the pattern of hydrophobicity in the protein's amino acid sequence.

Figure 5.16 shows the hit rates of the parallel GA with different migration models on the Alpha cluster. Figure 5.17 shows the performance measurements for different numbers of processors on the Alpha cluster. Because GAs are stochastic procedures, we have done more than twenty measurements for each configuration. The total population size is set to be 768. For the mutation stage and the crossover stage, the cooling scheme starts with $C = 2$ and is cooled by a factor of 0.99 for every 4 generations. The communication frequency is set to be every 5 generations, exchanging 10% subpopulations.

Figure 5.18 shows the hit rates of the HPGA with different migration models on the computational grid environment. Figure 5.19 shows the performance measurements for

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



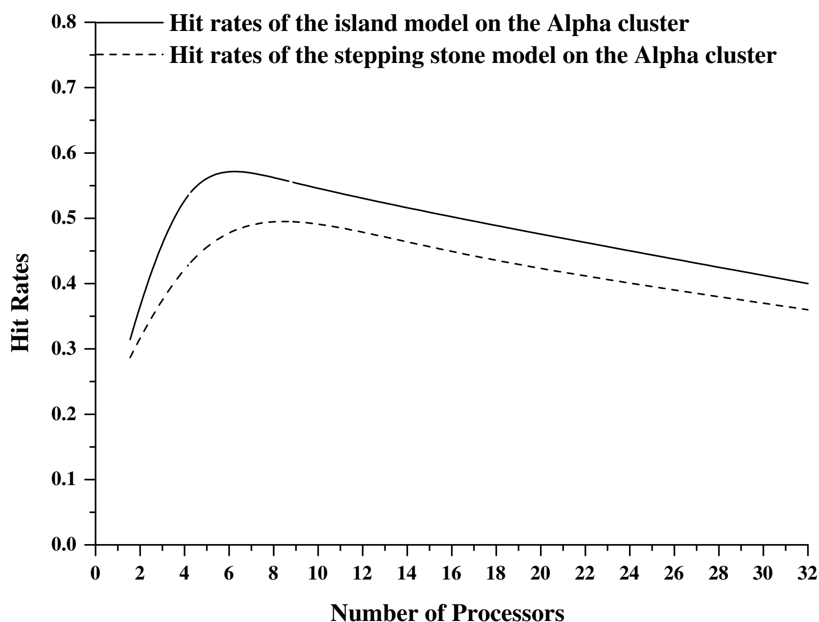



Figure 5.16: Hit rates of the parallel GA on the Alpha cluster

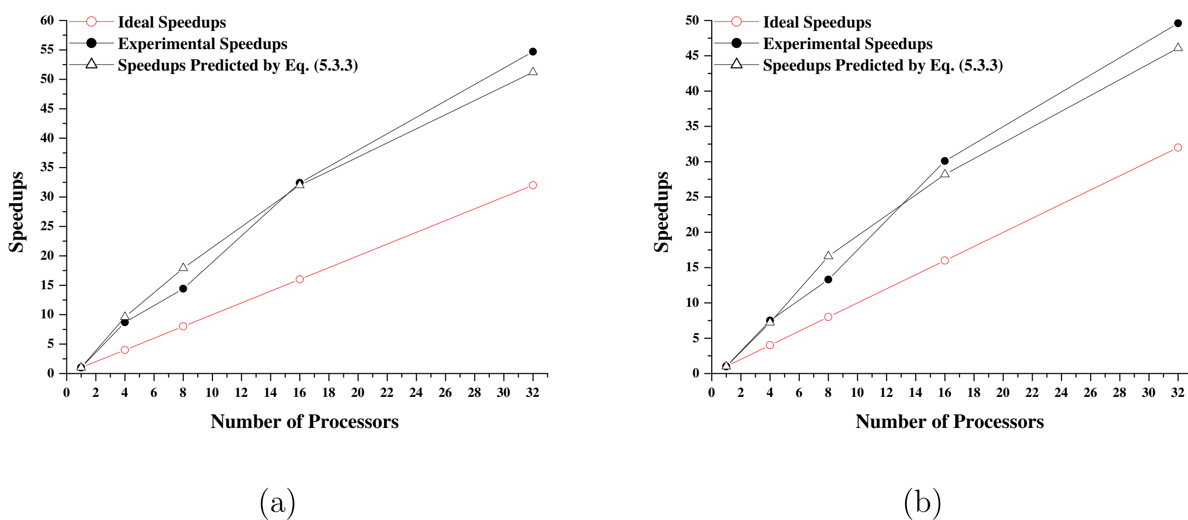



Figure 5.17: Speedups of the parallel GA with (a) the island model and (b) the stepping stone model on the Alpha cluster

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



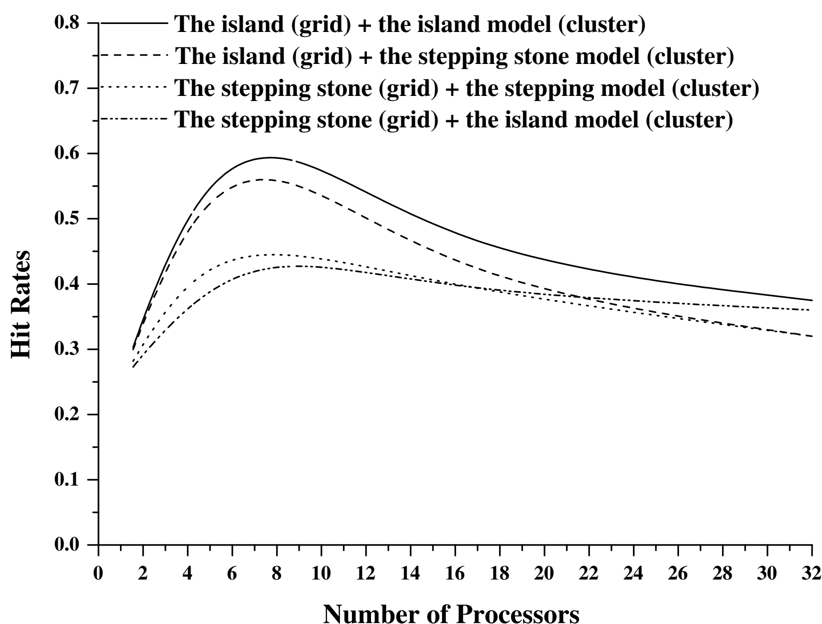



Figure 5.18: Hit rates of the HPGA on the computational grid environment

the protein folding simulations using the HPGA on the computational grid environment. The processors used are distributed evenly on the four clusters. The island model and the stepping stone model are used on the grid level and the cluster level respectively. From these figures we can see that our hierarchical communication architecture in Figure 4.23 and 5.3b for HPGAs can be efficiently applied to the computational grid environment. In our experiments, speedups of the HPGA with the island model on both the grid level and the cluster level are slightly higher. This is because the island model allows more freedom and brings more messages into subpopulations. Thus, the algorithm can work the result out faster. We can also find that the speedups predicted by Eq. (5.3.4) are very close to the experimental speedups.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



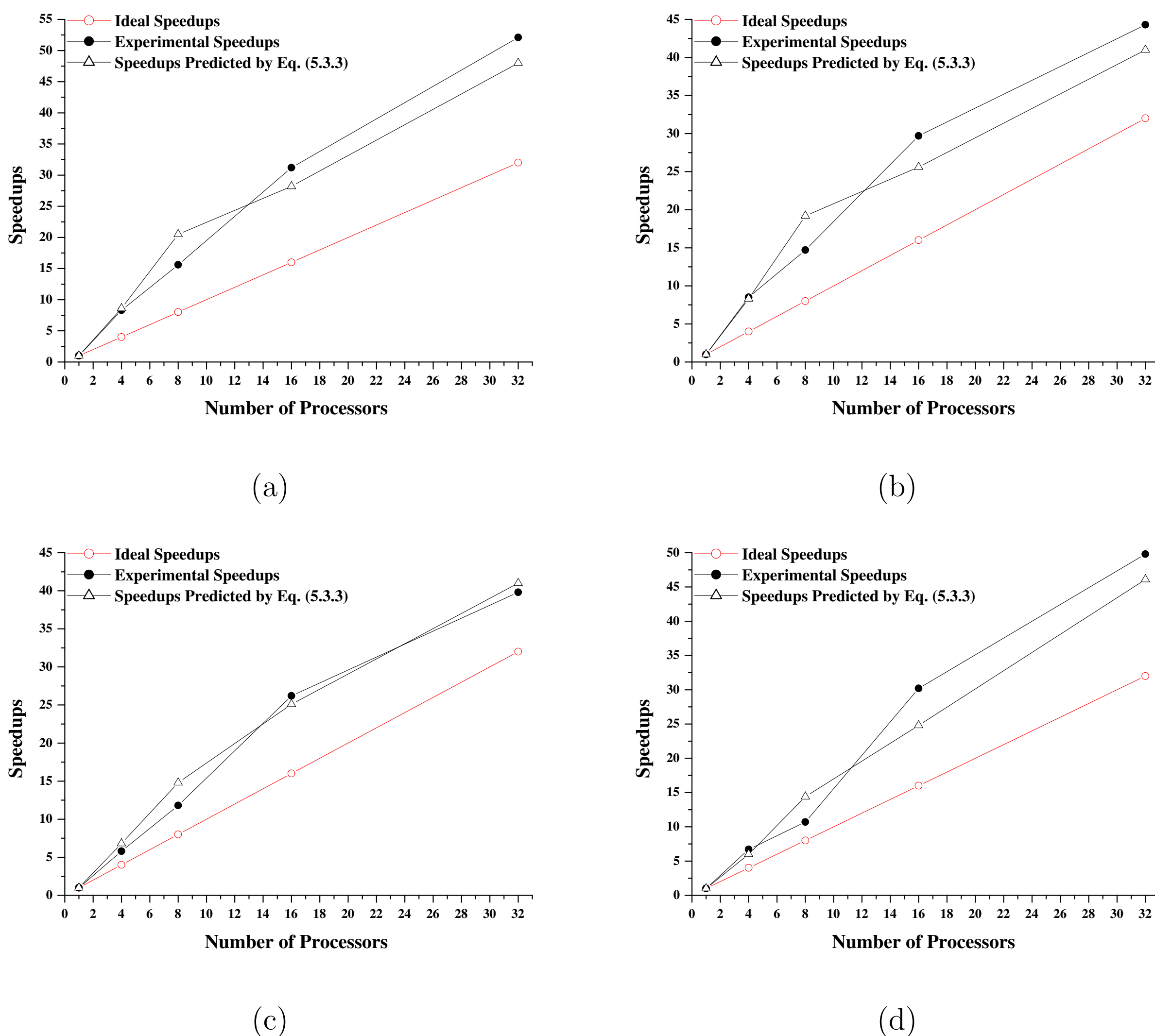



Figure 5.19: Speedups of our HPGA with (a) the island model on both the grid level and the cluster level, (b) the island model on the grid level and the stepping stone model on the cluster level, (c) the stepping stone model on both the grid level and the cluster level, (d) the stepping stone model on the grid level and the island model on the cluster level

## 5.4 Summary

In this chapter, we have presented a new parallel pattern-based framework for the development of high performance CB applications. We have identified common communication patterns by analyzing the characteristics of two popular CB algorithms. By integrating

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



these pattern using multi-paradigms in C++, our framework provides good extensibility and reusability. Moreover, we have integrated the two level communication schemes for grid computing into the framework. So, it can support two level hierarchical applications, such as our HPGA for the PFP on computational grids. Experiments show that our framework can be used to develop HPC applications with substantial performance gains on both PC clusters and computational grid environments.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Chapter 6

# Conclusions and Future Work

The next two sections further highlight our research achievements and suggest possible future research directions.

## 6.1 Conclusions

The string representation of biomolecules allows for a wide range of algorithmic techniques concerned with strings to be applied for analyzing and comparing biological data. We have contributed to this field by analyzing and constructing HPC applications that address problems with relevance to biological sequence analysis and structure prediction.

First, we have analyzed two categories of popular CB problems: DP algorithms and GAs. Analyzing and understanding characteristics of these algorithms will help us to develop efficient parallel applications for them.

According to the characteristics of DP algorithms, we have proposed a tunable coarse-grained partitioning and communication scheme for regular and irregular DP applications. By introducing two performance-related parameters, we can tradeoff between computation time and communication time by tuning these two parameters and thus obtain the maximum possible performance. We have demonstrated how this algorithm leads to

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



substantial performance gains for DP applications.

Next, we have designed a new HPGA for the PFP on PC clusters and computational grids. Our hierarchical approach unites the inter-cluster and intra-cluster parallelism in an efficient way by using a combination of two communication models, i.e. the stepping stone model and the island model. This unique combination achieves super-linear speedups on two different parallel architectures. Based on the concept of *hit rates* we also introduce a mathematical model to explain and predict our experimental results.

At last, we have proposed a parallel pattern-based framework to facilitate the semi-automatic development of HPC programs. By separating the communication structure of a parallel program from the sequential application, parallel patterns can be reused and therefore allow for a rapid development of HPC applications. We show how our parallel pattern-based framework can be deployed to implement parallel DP algorithms and HPGAs effectively and efficiently.

## 6.2 Future Work

The exponential growth of genome data demands even more HPC solutions in the future. As algorithms favored by biologists are not fixed, programmable parallel environments are eagerly required to speed up the compute-intensive tasks in CB. Our future work includes adding more patterns to the framework and identifying more applications that can benefit from it. Now, we are working on the identification of parallel patterns that are frequently used on popular data structures in CB such as sequences, trees and matrices. Also, we will add some functional modules to our framework to facilitate the use of it.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



### 6.2.1 Using Our Framework to Solve Other Dynamic Programming Applications

Although many algorithms in CB can be implemented using DP, they usually exhibit different characteristics according to their application fields. Only the wavefront computation is not enough to reflect the precise characteristics for all DP algorithms. We will study more DP applications and extend our framework to support most DP problems in CB.

#### Phylogenetic Analysis

Phylogenetic analysis provides a conceptual framework for understanding evolution but involves several computational challenges. The generation of evolutionary trees can be seen as two distinct NP-complete problems: multiple sequence alignments and phylogenetic tree searches. Phylogenetic analysis of sequence data depends strongly on accurate multiple alignments. In addition, there are other problems, such as orthologs and paralogs. Orthologs are sequences derived from a common ancestor through vertical descent. In more direct terms, this means the same gene in different species. Paralogs are genes within the same genome that have been generated by duplication. Distinguishing between orthologs and paralogs is important for building accurate phylogenetic trees.

As the number of DNA and protein sequences in databases increases, it is increasingly important to be able to create multiple sequence alignments for very large numbers of sequences. However, given that the underlying alignment algorithm requires $O(n^2)$ steps, where $n$ is the number of sequences to be aligned, it is not surprising that these standard tools soon begin to take many hours to run. If we further consider the problem of simultaneously aligning sequences and finding a plausible phylogeny for them, such as what the Sankoff and Cedergren's gap-substitution algorithm [123] (it is also a multidimensional dynamic programming algorithm), it will take even more time. Parallel computing thus

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



is eagerly needed in this field.

### Prediction of RNA Secondary Structure

The simplicity and the amount of information preserved, when describing complex bi-molecular as sequences of residues is the foundation of most algorithms in computational biology. However, in the real world the biomolecules DNA, RNA, and proteins are not one-dimensional strings, but full three dimensional structures, e.g. the three-dimensional structure of the double stranded DNA molecule that stores the genetic material of an organism is the famous double helix described by Watson and Crick [147].

Secondary structure in RNA is the list of base pairs that occur in a three dimensional RNA structure. According to the theory of thermodynamics the optimal folding of an RNA sequence are those of minimum free energy, and thus the native folding, i.e. the folding encountered in the real world, should correspond to the optimal folding. Furthermore, thermodynamics tells us that the folding of an RNA sequence in the real world is actually a probability distribution over all possible structures, where the probability of a specific structure is proportional to an exponential of the free energy of the structure. For a set of structures, the partition function is the sum over all structures of the set of the exponentials of the free energies. DP algorithms combined with the nearest neighbor model and experimentally determined free energy parameters give rigorous solutions to the problems of computing minimum free energy structures, structures that are usually close to real world optimal folding, and partition functions that yield exact base pair probabilities.

Zuker [152] proposes a method to determine all base pairs that can participate in structures with a free energy within a specified range from the optimal. McCaskill [108] demonstrates how a related DP algorithm can be used to calculate equilibrium partition functions, which lead to exact calculations of base pair probabilities. A major problem

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



for these algorithms is the time required to evaluate possible internal loops. In general, this requires time $O(n^4)$. It is a time consuming task. Much work has been done to reduce the total time to $O(n^3)$ by Waterman [146] and Lyngso [104]. Thus many new algorithms have been presented. Constructing HPC programs using the parallel pattern-based framework for these algorithms is also one of our future works.

### 6.2.2 Using Our Framework to Solve Tertiary Structure Prediction for Real Proteins

In this thesis, we have presented an HPGA for the protein folding problem (PFP) using 2D HP lattice models. Although it is a useful demonstration of the potential advantages of a GA for structure prediction, this model is so simple that it leaves open the question of its applicability to real proteins.

In the first attempt to apply GAs to reproducing the tertiary structure of real proteins, Sun [139] used a description of a protein molecule that consisted of a full backbone and one virtual atom per side chain. A potential of mean force derived from known protein structures was used to assess fitness. A library of peptide fragment conformations 2–5 residues long was used to construct initial conformations and to perform mutational changes. The library was constructed from known protein structures. An additional constraint was the experimental radius of gyration. A population size of 90 was used. Low final root mean square deviations from the experimental structures are reported. The significance of the results is hard to assess. Fragments were selected from the library on the basis of sequence similarity and the library contains the two larger structures that were reproduced [120].

Bowie and Eisenberg [34] constructed initial conformations of a small protein using a similar method with Sun. Nine-residue segments were selected from a library of fragment conformations on the basis of the environment codes. A similar procedure was used for

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



some larger fragments 15-25 residues long. Care was taken to exclude homologous structures from the database. The method of selecting initial conformations did enhance the local structure accuracy to a value higher than that expected by chance alone. Structures were then improved by a GA procedure in which each gene is the set of dihedral angles of a structure and mutations are changes to one angle. For recombination, segments of one gene were replaced with segments of another. Mutations and cross-overs had a high probability of occurring at the fragment junctions. The fitness was evaluated with a function containing contributions from the profile fit, hydrophobicity, accessible surface area, atomic overlap and the sphericalness of the structure. The weighting of the terms in the potential was strongly biased by the experimental structure.

Pedersen and Moult have used a GA to predict the structure of small fragments (12-22 residues long) of proteins in a blind test [119]. The procedure used was an extension of an earlier torsion space MC method [27]. A full heavy atom and polar hydrogen representation of the chain is used. Conformations with excessive steric overlap were rejected. Fitness was evaluated using a potential that was based on point-charge electrostatics and accessible surface area. Terms in the force field were parameterized with a potential of mean-force analysis of experimental structures. A gene is a string of $\phi$,$\psi$ and $\chi$ angles representing a conformation. Cross-over points were weighted towards positions where the conformation was most varied in the current population. The extensive annealing of side-chain conformations was performed at cross-over points before evaluating the fitness of the new gene. The population size was 200–300, and 40–50 generations were performed. Parameters of the search were optimized systematically on a set of fragments with known structure. One of the three blind predictions did produce a native-like structure for a 22-residue fragment. Experience with this procedure shows it to be substantially more effective than the MC procedure at generating low-energy structures.

At present, however, most of mentioned methods are limited to relatively small protein

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



fragments. This is mainly because very compute-intensive tasks are needed for long protein sequences. To design and develop efficient HPC algorithms on computational grids for the PFP are thus an interesting and worthy task in our future work.

### 6.2.3 Extending Our Parallel Pattern-based Framework

Now, our parallel pattern-based framework presented in this thesis can be used to develop HPC applications semi-automatically. That is, users still need to provide some application-related code in order to get the final HPC application.

A major part of our future work is to develop user interfaces and an algorithm parser. Users only need to provide necessary descriptions of a specific algorithm through a text-script parser and then, the parallel application can be generated automatically. Figure 6.1 shows an illustration.

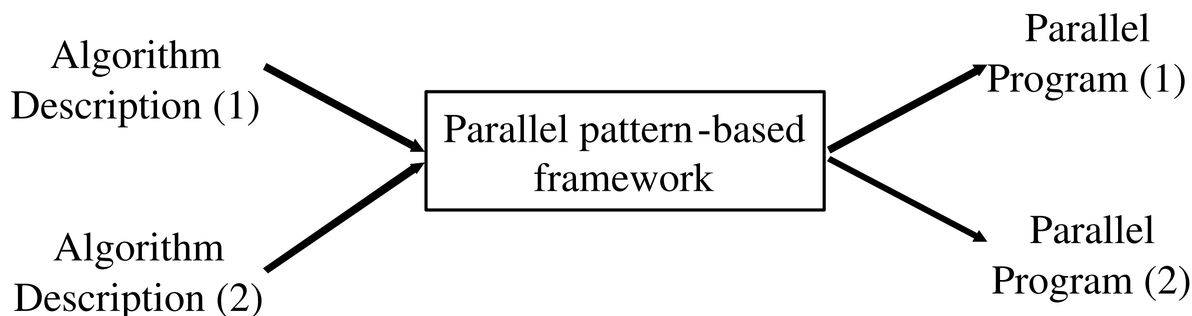



Figure 6.1: The new framework can generate HPC applications automatically.

The new parallel pattern-based framework will have these special features: high performance, easy to use and good extensibility. And also because a successful HPC programming environment should be amenable to performance prediction, a cost prediction module will be further developed so as to provide users a convenient tool. It is shown in Figure 6.2.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



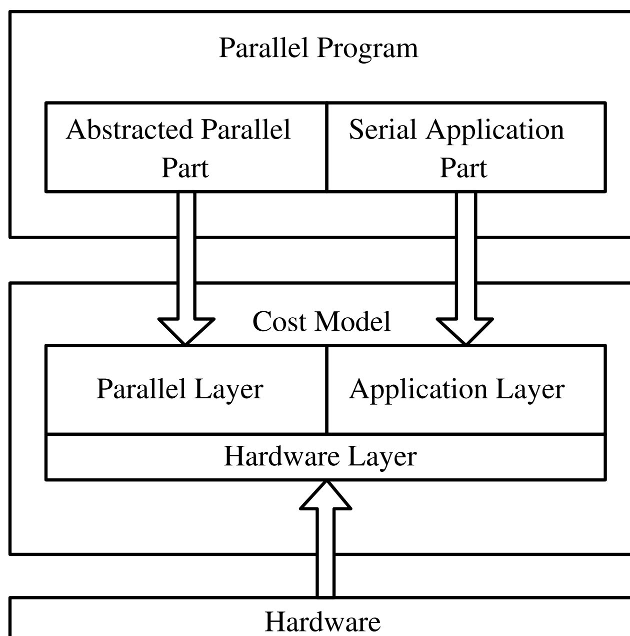



Figure 6.2: Framework of the cost module.

- Parallel Layer – describes the parallel characteristics of subtasks in terms of expected computation-communication interactions between processors.
- Application Layer – describes the sequential part of every subtask within an application that can be executed in parallel.
- Hardware Layer – collects system specification parameters, micro-benchmark results, statistical models, analytical models, and heuristics to characterize the communication and computation abilities of a particular system.

According to the layered framework, a cost prediction module is built up from a number of separate objects. Each object is of one of the following types: application, parallel framework and hardware. A key feature of the object organization is the independent representation of computation, parallelization, and hardware. This is possible due to strict object interaction rules. Using the cost model, users can understand the factors

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



that affect the performance well and can get a performance prediction without running the program.

Also, we will integrate more patterns into our framework, such as divide and conquer, branch and bound, simulated annealing, and randomized patterns.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

# Bibliography

[1] GenBank. http://www.ncbi.nih.gov/Genbank/genbankstats.html.

[2] Global Arrays. http://www.emsl.pnl.gov/docs/global/ga.html.

[3] Globus Project. http://www.globus.org/.

[4] IBM LoadLeveler. http://www-128.ibm.com/developerworks/library/l-halinux3/.

[5] Instruction Systolic Array ISA. http://www.iti.fh-flensburg.de/lang/papers/isa/.

[6] Message Passing Interface Forum. http://www.mpi-forum.org/.

[7] Moor's Law from Intel. http://www.intel.com/technology/mooreslaw/index.htm.

[8] MPICH – G2 Project in High-Performance Computing Laboratory of North Illinois University. http://www3.niu.edu/mpi.

[9] MPICH-P4. http://www-unix.mcs.anl.gov/mpi/mpich/.

[10] OpenMP. http://www.openmp.org/.

[11] Platform LSF. http://www.platform.com/products/LSF/.

[12] Portable Batch System. http://www.openpbs.org/.

[13] RCSB Protein Data Bank. http://www.rcsb.org/pdb/.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[14] SETI@home: The Search for Extraterrestrial Intelligence. http://setiathome.ssl.berkeley.edu/.

[15] Standard C++. http://www.open-std.org/jtc1/sc22/wg21/.

[16] Sun Grid Engine. http://gridengine.sunsource.net/.

[17] The Beowulf Cluster Site. http://www.beowulf.org/.

[18] TOP500 Supercomputer Sites. http://www.top500.org/.

[19] N.R. Adiga. An Overview of the B1ueGene/L Supercomputer. In *Proceedings of SC'2002*, 2002.

[20] M. Aldinucci, M. Danelutto, and P. Teti. An Advanced Environment Supporting Structured Parallel Programming in Java. *Future Generation Computer Systems*, 19(5):611–626, 2003.

[21] W. Allcock, J. Bresnahan, I. Foster, L. Liming, J. Link, and P. Plaszczac. GridFTP Update. In *Technical Report, available at http://www.globus.org/alliance/publications/papers.php*, 2002.

[22] G. Almasi and A. Gottlieb. *Highly Parallel Computing*. Benjamin/Cummings Publishing Company, 1994.

[23] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@Home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11):56–61, 2002.

[24] C.B. Anfinsen. Principles that Govern the Folding of Proteins. *science*, 1973.

[25] E.L. Anson and G.W. Myers. Realigner: A Program for Refining DNA Sequence Multi-Alignments. In *1st Conference on Computational Molecular Biology*, pages 9–16, 1997.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[26] J. Anvik, S. MacDonald, D. Szafron, J. Schaeffer, S. Bromling, and K. Tan. Generating Parallel Programs from the Wavefront Design Pattern. In *Proceedings of the* $7^{th}$ *International Workshop on High-Level Parallel Programming Models and Supportive Environments*, 2002.

[27] F. Avbelj and J. Moult. Determination of the Conformation of Folding Initiation Sites in Proteins by Computer Simulation. *Proteins*, 23:129–141, 1995.

[28] A. Bartoli, P. Corsini, G. Dini, and C.A. Prete. Graphical Design of Distributed Applications through Reusable Components. *IEEE Parallel Distrib. Technol.*, 3:37–50, 1995.

[29] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and K. Moore. Hence: A Heterogeneous Network Computing Environment. *Scientific Programming*, 3(1):49–60, 1994.

[30] D.A. Benson, I.K. Mizrachi, D.J. Lipman, J. Ostell, B.A. Rapp, and D.L. Wheeler. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Nucleic Acids Research*, 28(1):15–18, 2000.

[31] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. In *Proc. of IOPADS'99*, 1999.

[32] E. Birney and R. Durbin. Dynamite: Aflexible Code Generating Language for Dynamic Programming Methods. In *Proc. Intelligent Systems for Molecular Biology*, pages 56–64, 1997.

[33] J. Blazewicz and M. Kasprzak. Complexity of DNA Sequencing by Hybridization. *Theoretical Computer Science*, 290(3):1459–1473, 2003.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[34] J.U. Bowie and D. Eisenberg. An Evolutionary Approach to Folding Small $\alpha$-Helical Proteins That Uses Sequence Information and an Empirical Guiding Fitness Function. *Proc Natl Acad Sci USA*, 91:4436–4440, 1994.

[35] J.U. Bowie, R. Luthy, and D. Eisenberg. A Method to Identify Protein Sequences That Fold Into a Known Three Dimensional Structure. *Science*, 253:164–170, 1991.

[36] S. Bromling, S. MacDonald, J. Anvik, J. Schaeffer, D. Szafron, and K. Tan. Pattern-based Parallel Programing. In *Proceedings of the 2002 International Conference on Parallel Processing*, 2002.

[37] J.C. Browne, M. Azam, and S. Sobek. Code: A Unified Approach to Parallel Programming. *IEEE Software*, 6(4):10–18, 1989.

[38] J.C. Browne, S.I. Hyder, J. Dongarra, K. Moore, and P. Newton. Visual Programming and Debugging for Parallel Computing. *IEEE Parallel Distrib. Technol.*, 3:75–83, 1995.

[39] S.H. Bryant and C.E. Lawrence. An Empitrcal Energy Function for Threading Protein Sequence through Folding Motif. *Proteins: Structure, Function and Genetics*, 16:92–112, 1993.

[40] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996.

[41] R. Buyya, K. Branson, J. Giddy, and D. Abramson. The Virtual Laboratory: Enabling on Demand Drug Design with the Worldwide Grid. *Concurrency and Computation: Practice and Experience*, 15(1), 2003.

[42] E. Cantu-Paz. A Survey of Parallel Genetic Algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[43] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proc. of the* 9*th* *Heterogeneous Computing workshop (HCW 2000)*, pages 349–363, 2000.

[44] M. Cole. *Algorithmic Skeletons: A Structured Approach to the Management of Parallel Computations*. MIT Press, 1988.

[45] M. Cole. Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. *Parallel Computing*, 30(3):389–406, 2004.

[46] J. Coplien. *Multi-paradigm Design for C++*. Addison Wesley, 1999.

[47] K. Czajkowski, S. Fitzgerald, and I. Foster C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proc. of the* 10*th* *IEEE International Symposium on High-Performance Distributed Computing (HPDC-10*, 2001.

[48] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proc. of IPPS/SPDP 1998 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[49] K. Czajkowski, I. Foster, and C. Kesselman. Co-Allocation Services for Computational Grids. In *In Proc.* 8*th* *IEEE Symp. on High Performance Distributed Computing*, 1999.

[50] T. Dandekar and P. Argos. Folding the Main Chain of Small Proteins with the Genetic Algorithm. *Journal of Molecular Biology*, 236:844–861, 1994.

[51] M. Danelutto. Hpc The Easy Way: New Technologies for High Performance Application Development and Deployment. *Journal of Systems Architecture*, 49:399–419, 2003.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[52] M. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A Model of Evolutionary Change in Proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1977.

[53] A. Demiriz, K.P. Bennett, and M.J. Embrechts. Semi-Supervised Clustering using Genetic Algorithms. In *Artificial Neural Networks in Engineering (ANNIE-99)*, pages 809–814, 1999.

[54] K.A. Dill, S. Bromberg, K. Yue, K.M. Fiebig, D.P. Yee, P.D. Thomas, and H.S. Chan. Principles of Protein Folding: A Perspective from Simple Exact Models. *Protein Science*, 4:561–602, 1995.

[55] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis–Probabilistic Models of Protein and Nucleic Acids*. Cambridge University Press, 1998.

[56] M.J. Embrechts, D. Devogelaere, and M. Rijckaert. Supervised Scaled Regression Clustering: An Alternative to Neural Networks. In *Proceedings of the IEEE-INN-ENNS International Conference(IJCNN2000)*, pages 571–576, 2000.

[57] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. A Resource Management Architecture for Metacomputing Systems. In *Proc. of* $2^{nd}$ *IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, pages 39–46, 2002.

[58] E. Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley, 1998.

[59] D. Fischer. 3D−Shotgun: A Novel, Cooperative, Fold−Recognition Meta Predictor. *Proteins: Structure, Function and Genetics*, 51:434–441, 2003.

[60] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



In *Proc. of* $6^{th}$ *IEEE Symp. on High Performance Distributed Computing*, pages 365–375, 1997.

[61] M.J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, 21(9), 1972.

[62] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc, 1998.

[63] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/ogsa/.

[64] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proc. of* $5^{th}$ *ACM Conference on Computer and Communications Security Conference*, pages 83–92, 1998.

[65] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal Supercomputer Applications*, 15(3), 2001.

[66] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002.

[67] S. Fraser, K. Beck, G. Booch, J. Coplien, R. Johnson, and B. Opdyke. Beyond the Hype: Do Patterns and Frameworks Reduce Discovery Costs? In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 342–344, 1997.

[68] N. Futamura, S. Aluru, and X. Huang. Parallel Syntenic Alignments. In *International Conference on High Performance Computing*, pages 420–430, 2002.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[69] E. Gabriel, M. Resch, T. Beisel, and R. Keller. Distributed Computing in a Heterogenous Computing Environment. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computing Scicence*, 1998.

[70] Z. Galil and K. Park. Dynamic Programming with Convexity, Concavity and Sparsity. *Theoretical Computer Science*, 92(1):49–76, 1992.

[71] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley, 1995.

[72] A. Geist, A. Begulin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM-Parallel Virtual Machine- A User's Guide and Tutorial for Networked Parallel computing.* MIT Press, 1994.

[73] M.S. Gelfand, A.A. Mironov, and P.A. Pevzner. Gene Recognition Via Spliced Sequence Alignment. *Proc. Natl. Acad. Sci*, 93:9061–9066, 1996.

[74] M.S. Gelfand and M.A. Roytberg. A Dynamic Programming Approach for Prediction the Exon-Intron Structure. *Biosystems*, 30:173–182, 1993.

[75] J. Gerlach. Generic Programming of Parallel Application with JANUS. *Parallel Processing Letters*, 12(2):175–190, 2002.

[76] K. Ginalski, A. Elofsson, D. Fischer, and L. Rychlewski. 3D−Jury: A Simple Approach to Improve Protein Structure Predictions. *Bioinformatics*, 19:1015–1018, 2003.

[77] A. Godzik, A. Kolinske, and J. Skolnick. A Topology Fingerprint Approach to Inverse Protein Folding Problem. *Journal of Molecular Biology*, 227:227–238, 1992.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[78] A. Godzik and J. Skolnick. Sequence-Structure Matching in Globular Proteins: Application to Supersecondary and Tertiary Structure Determination. *National Academy of Science*, 89:12098–12102, 1992.

[79] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[80] O. Gotoh. An Improved Algorithm for Matching Biological Sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[81] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. *MPI: The Complete Reference (Vol. 2).* MIT Press, 1998.

[82] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message-Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[83] W.W. Hargrove, F.M. Hoffman, and T. Sterling. The Do-It-Yourself Supercomputer. *Scientific American*, 285(2):72–79, 2001.

[84] W.E. Hart and S. Istrail. Robust Proofs of NP-hardness for Protein Folding: General Lattices and Energy Potentials. *Journal of Computational Biology*, 4(1):1–22, 1997.

[85] W.E. Hart and A. Newman. *The Computational Complexity of Protein Structure Prediction in Simple Lattice Models.* CRC Press, 2003.

[86] P. Higgs and T. Attwood. *Bioinformatics and Molecular Evolution.* Blackwell Publishing, 2005.

[87] D.S. Hirschberg. A Linear Space Algorithm for Computing Longest Common Subsequences. *Communications of the ACM*, 18:341–343, 1975.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[88] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[89] X. Huang. A Space-Efficient Algorithm for Local Similarities. *Computer Applications in the Biosciences*, 6:373–381, 1990.

[90] X. Huang and K.M. Chao. A Generalized Global Alignment Algorithm. *Bioinformatics*, 19(2):228–233, 2003.

[91] T. Imamura, Y. Tsujita, H. Koide, and H. Takemiya. An Architecture of Stampi: MPI Library on a Cluster of Parallel Computers. *Lecture Notes in Computer Science*, 1908:200–207, 2000.

[92] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551–563, 2003.

[93] M. Karplus and E. Shakhnovich. Protein Folding: Theoretical Studies of Thermodynamics and Dynamics. *Protein Folding*, pages 196–237, 1992.

[94] T. Kielmann, R.F.H. Hofman, H.E. Bal, A. Plaat, and R.A.F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *Proc. of Ppopp'99*, pages 131–140, 1999.

[95] S.S. Kremer. *Molecular Bioinformatics: Algorithms and Applications*. Walter de Gruyter, 1995.

[96] A. Krogh, M. Brown, I.S. Mian, K. Sjolander, and D. Haussler. Hidden Markov Models in Computational Biology: Applications to Protein Modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[97] H. Kuchen. A Skeleton Library. In *EURO-PAR'2002*, LNCS 2400, 2002.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[98] T. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, 1970.

[99] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. The Benjamin-Cummings Publishing Company Inc., 1994.

[100] M.A. Kurowski and J.M. Bujnicki. Genesilico Protein Structure Prediction Meta-Server. *Nucleic Acids Research*, 31:3305–3307, 2003.

[101] K.F. Lau and K.A. Dill. A Lattice Statistical Mechanics Model of the Conformational and Sequence Spaces of Proteins. *Macromelecules*, 22:3986–3997, 1989.

[102] S. Lee, M.K. Cho, J.W. Jung, J.H. Kim, and W. Lee. Exploring Protein Fold Space by Secondary Structure Prediction Using Data Distribution Method on Grid Platform. *Bioinformatics*, 20(18):3500–3507, 2004.

[103] S.M. LeGrand and K.M. Jr Merz. The Genetic Algorithm and the Conformational Search of Polypeptides and Proteins. *Molecular Simulation*, 13:299–320, 1994.

[104] R.B. Lyngso, M. Zuker, and C.N.S. Pedersen. Fast Evaluation of Internal Loops for RNA Secondary Structure Prediction. *Bioinformatics*, 15:440–445, 1999.

[105] S. MacDonald, J. Anvik, S. Bromling, J. Schaeffer, D. Szafron, and K. Taa. From Patterns to Frameworks to Parallel Programs. *Journal of Parallel Computing*, 28(12):1663–1684, 2002.

[106] T. Madej, J.F. Gibrat, and S.H. Bryant. Threading a Database of Protein Cores. *Proteins: Structure, Function and Genetics*, 23:356, 1995.

[107] T.G. Mattson, D. Scott, and S.R. Wheat. A TeraFLOP Supercomputer in 1996: the ASCI TeraFLOP System. In *Proceedings of IPPS'96, The* $10^{th}$ *International Parallel Processing Symposium*, 1996.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[108] J.S. McCaskill. The Equilibrium Partition Function and Base Pair Binding Probabilities for RNA Secondary Structure. *Biopolymers*, 29:1105–1119, 1990.

[109] S. Miyazaki, H. Sugawara, T. Gojobori, and Y. Tateno. DNA Data Bank of Japan DDBJ in XML. *Nucleic Acids Research*, 31(1):13–16, 2003.

[110] D.W. Mount. *Bioinformatics-Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 2001.

[111] D.R. Musser and A. Siani. *STL Tutorial and Reference Guide*. Addison Wesley, 1996.

[112] E.W. Myers and W. Miller. Optimal Alignments in Linear Space. *Computer Applications in the Biosciences*, 4:11–17, 1988.

[113] J. Nabrzyski, J.M. Schopf, and J. Weglarz. *Grid Resource Management*. Kluwer Publishing, 2003.

[114] S.B. Needleman and C.D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *J. Mol. Biol*, 48:443–453, 1970.

[115] J. Nieplocha, R.J. Harrison, M.K. Kumar, B. Palmer, V. Tipparaju, and H. Trease. Combining Distributed and Shared Memory Models: Approach and Evolution of the Global Arrays Toolkit. In *Proceedings of Workshop on Performance Optimization for High-Level Languages and Libraries (ICS'2002)*, 2002.

[116] J. Nieplocha, R.J. Harrison, and R.J. Littlefield. Global Arrays: A Nonuniform Memory Access Programming Model for High-Performance Computers. *The Journal of Supercomputing*, 10(2):169–189, 1996.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[117] C. Notredame and D.G. Higgins. SAGA: Sequence Alignment by Genetic Algorithm. *Nucleic Acid Research*, 24:1515–1524, 1996.

[118] C. Notredame, E.A. O'Brien, and D.G. Higgins. RAGA: RNA Sequence Alignment by Genetic Algorithm. *Nucleic Acids Research*, 25(22):4570–4580, 1997.

[119] J.T. Pedersen and J. Moult. Ab Initio Structure Prediction for Small Polypeptides and Protein Fragments Using Genetic Algorithms. *Proteins*, 23:454–460, 1995.

[120] J.T. Pedersen and J. Moult. Genetic Algorithms for Protein Structure Prediction. *Curr Opin Struct Biol*, 6(2):227–231, 1996.

[121] D. Pritchard. Mathmatical Models of Distributing Computation. In T. Muntean, editor, *Parallel Programming of Transputer Based Machines*, pages 25–36. 1988.

[122] L. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[123] D. Sankoff, R.J. Cedergren, and W. McKay. A Strategy for Sequence Phylogeny Research. *Nucleic Acids Research*, 10(1):421–431, 1982.

[124] E.E. Santos, E. Santos, and Jr. Reducing the Computational Load of Energy Evaluations for Protein Folding. In *4th IEEE Symposium on Bioinformatics and Bioengineering*, 2004.

[125] J. Schaeffer, D. Szafron, G. Lobe, and I. Parsons. The Enterprise Model for Developing Distributed Applications. *IEEE Parallel Distrib. Technol.*, pages 85–96, 1993.

[126] B. Schmidt and H. Schroder. Massively Parallel Sequence Analysis with Hidden Markov Models. In *Proceedings of IC-SEC*, pages 781–784, 2002.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[127] B. Schmidt, H. Schroder, and M. Schimmler. A Hybrid Architecture for Bioinformatics. *Future Generation Computer System*, 18:855–862, 2002.

[128] B. Schmidt, H. Schroder, and M. Schimmler. Massively Parallel Solutions for Molecular Sequence Analysis. In *Proc. of IPDPS'02*, 2002.

[129] J. Serot and D. Ginhac. Skeletons for Parallel Image Processing: An Overview of the Skipper Project. *Journal of Parallel Computing*, 28(12):1685–1708, 2002.

[130] J.C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, 1997.

[131] R. Shonkwiler. Parallel Genetic Algorithms. In $5^{th}$ *International Conference on Genetic Algorithms*, 1992.

[132] A. Singh, J. Schaeffer, and M. Green. A Template-Based Tool for Building Applications in a Multi-Computer Network Environment. *Parallel Computing*, pages 461–466, 1989.

[133] T.F. Smith and M.S. Waterman. Comparison of Biosequences. *Adv. Appl. Math*, 2:482–489, 1981.

[134] T.F. Smith and M.S. Waterman. Identification of Common Subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[135] A.J. Steen and J.J. Dongarra. Overview of Recent Supercomputers, 2004. http://www.top500.org/ORSC/.

[136] G. Stoesser, W. Baker, A. Van Den Broek, M. Garcia-Pastor, C. Kanz, T. Kulikova, R. Leinonen, Q. Lin, V. Lombard, R. Lopez, R. Mancuso, F. Nardone, P. Stoehr, M. Tuli, K. Tzouvara, and R. Vaughan. The EMBL Nucleotide Sequence Database: major new developments. *Nucleic Acids Research*, 31(1):17–22, 2003.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[137] A. Streit. A Self-Tuning Job Scheduler Family with Dynamic Policy Switching. In *Proc. of the* $8^{th}$ *workshop on job scheduling strategies for parallel*, 2002.

[138] B. Stroustrup. Why C++ is not just an Object-Oriented Programming Language. *OOPS Messenger*, 6(4):1–13, 1995.

[139] S. Sun. Reduced Representation of Protein Structure Prediction: Statistical Potential and Genetic Algorithms. *Protein Science*, 2:762–785, 1993.

[140] J.D. Szustakowski and Z. Weng. Protein Structure Alignment Using a Genetic Algorithm. *Proteins*, 38:428–440, 2000.

[141] R. Thiele, R. Zimmer, and T. Lenguaer. Protein Threading by Recursive Dynamic Programming. *Journal of Molecular Biology*, 290:757–779, 1998.

[142] R. Unger and J. Moult. Genetic Algorithms for Protein Folding Simulations. *Journal of Molecular Biology*, 231:75–81, 1993.

[143] L. Valiant. A Bridging Model for Parallel Computation. *Communication of the ACM*, 33(8):103–111, 1990.

[144] D. Vandevoorde and N.M. Josuttis. *C++ Template: The Complete Guide*. Addison Wesley, 2002.

[145] J. Venter. The Sequence of the Human Genome. *science*, 291(5507):1304–1351, 2001.

[146] M.S. Waterman and T.F. Smith. Rapid Dynamic Programming Methods for RNA Secondary Structure. *Advances in Applied Mathematics*, 7:455–464, 1986.

[147] J. Watson and F. Crick. A Structure of Deoxyribonucleic Acid. *Nature*, 171:737–738, 1953.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



[148] P. Wegner. *Research Directions in Object-Oriented Programming.* MIT Press, 1987.

[149] B. Wilkinson and M. Allen. *Parallel Programming- Techniques and Applications Using Networked Workstations and Parallel Computers.* Pearson Education Inc., 1999.

[150] M. Wilmanns and D. Eisenberg. Inverse Protein Folding by the Residue Pair Preference Profile Method. *Protein Engineering*, 8:626–639, 1995.

[151] G.V. Wilson. Assessing the Usability of Parallel Programming Systems: The Cowichan Problems. In *Proceedings of the IFIP Working Conference on Programming Ecvironments for Massively Parallel Distributed Systems*, 1994.

[152] M. Zuker. On Finding All Suboptimal Foldings of an RNA Molecule. *Science*, 244:48–52, 1989.

[153] M. Zuker and P. Stiegler. Optimal Computer Folding of Large RNA Sequences Using Thermodynamics and Auxiliary Information. *Nucleic Acids Research*, 9, 1981.