The Hierarchical Algorithms –Theory and Applications

Thesis by

Zheng-Yao Su

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1995

(Defended July 29, 1994)

Acknowledgments

It is a great pleasure to publicize my gratitude to those who have helped make this work possible. First, I would like to thank Professor Geoffrey C. Fox, my advisor, for the continuous support throughout my graduate study. I am especially grateful to him for his guidance and for giving me the freedom to explore many different topics which have fascinated me. My deepest thanks also go to Professor Steven Frautschi, who has given me advice and encouragement that have enhanced my understanding of the character necessary for a successful scientist. Special gratitude also goes to Professor Thomas Prince and Professor Michael Cross, my thesis committee members, who have made certain crucial suggestions for this work. I am greatly indebted to Professor Nai-Chang Yeh, who has played the role of a mentor and has given me constant encouragement after I came back to Caltech from Syracuse. More importantly, from her, I learned the elements a scholar should acquire. Thanks are due similarly to Professor Peter Weichman for agreeing to be my on-campus advisor during the term of detached-duty from Caltech. His many critical readings and suggestions have made significant contributions to this thesis.

My special thanks also go to Professors John Schwarz and Yaser Abu-Mostafa who have helped broaden my vision for the future course of academic pursuit. Profs. Tin-Wei Chiu, K.-L. Chang, C.-L. Lin, J. Chen, and H.-C. Chang at National Taiwan University have my most sincere thanks for their invaluable advice during various stages of my academic life. Profs. Pauchy W. Y. Hwang, San-Cheng Chang, Y.-Y Chen at NTU and J.-M. Lee at Academia Sinica deserve my appreciation for their timely encouragement. Heartfelt thanks should go to my teachers at Caltech –Profs. John Preskill, Kip Thorne, Steven Koonin, Ward Whaling, Charles Peck, Mark Wise, Barry Barish, and Mani Chandy. I am thankful also to teachers at Syracuse who helped open my eyes to various parts of different research areas –Profs. Enzo Marinari, Wojtek Furmanski, Christina Marchetti, Harvey Kaplan, Roger Penrose, Jim Royer, and A. P. Balachandran. Many others who have generously shared their ideas with me and provided assistance through this journey of quest –Paul Coddington, Clive Baillie, Ren-yuan Zhu, Paul Messina, Hong Qiang Ding, John Salmon, Roy Williams, Petar Simic, and Seth Lloyd– merit greatly deserved gratitude.

Full-hearted thanks should be acknowledged to many people at Northeast Parallel Architectures Center at Syracuse University, who have made the center a fun and excellent environment for pursuing research. Among them, I would particularly like to deliver my appreciation to Deborah Jones, Peter Crockett, Lisa Deyo, and Diane Sanderson for their constant consideration, warm cheering, ever-present sense of humor –despite the status of weather. Also, I am deeply grateful to the wonderful people of Caltech Concurrent Supercomputing Facilities –Mary Maloney, Chip Chapman, Heidi Lorenz-Wirzba, Jack Stewart, and Doug Freyberger– for their kindness and generosity in providing resources and support for me to finish this work. Donna Driscoll, Frances Spalding, Karen Fox, Gregory Dunn, and Kate Finigan –thank you all for the inspiring words and for the help I always needed in the last moments. To those at Millikan Library –Tess Legaspi, Sandy Garstang, Henry Frederick– I would say you will feel relieved of a heavy workload after I leave.

To those whom I have been so lucky to know on the west and east coasts –Meng-Chien Yang, Shuo-Hsien Hsiao, Lee-Wu Yang, John Apostolakis, Leewen Chen, Alvin Leung, Stuart Anderson, Mark Muldoon, Alex Ho, Isaac Wong, James Oyang, Miloje Makivic, Leping Han, Martin Bucher, and so many other people too numerous to name here– thank you all for sharing experiences which have revealed different facets of life and cultures.

These thanks hardly begin to embrace all the friends and colleagues who have stimulated and advised me over the years. For those not specifically mentioned, please understand my gratitude includes you.

Finally and most importantly, I would like to thank my parents. It is beyond words how grateful I am for their love, continuous encouragement, and unfailing support. It is to them I dedicate this thesis.

Abstract

Monte Carlo simulations are one of the most important numerical techniques for investigating statistical physical systems. Among these systems, spin models are a typical example which also play an essential role in constructing the abstract mechanism for various complex systems. Unfortunately, traditional Monte Carlo algorithms are afflicted with "critical slowing down" near continuous phase transitions and the efficiency of the Monte Carlo simulation goes to zero as the size of the lattice is increased. To combat critical slowing down, a very different type of collective-mode algorithm, in contrast to the traditional single-spin-flip-mode, was proposed by Swendsen and Wang in 1987 for Potts spin models. Since then, there has been an explosion of work attempting to understand, improve, or generalize it. In these so-called "cluster" algorithms, clusters of spin are regarded as one template and are updated at each step of the Monte Carlo procedure. In implementing these algorithms the cluster labeling is a major time-consuming bottleneck and is also isomorphic to the problem of computing connected components of an undirected graph seen in other application areas, such as pattern recognition.

A number of cluster labeling algorithms for sequential computers have long existed. However, the dynamic irregular nature of clusters complicates the task of finding good parallel algorithms and this is particularly true on SIMD (single-instruction-multiple-data) machines. Our design of the Hierarchical Cluster Labeling Algorithm aims at alleviating this problem by building a hierarchical structure on the problem domain and by incorporating local and nonlocal communication schemes. We present an estimate for the computational complexity of cluster labeling and prove the key features of this algorithm (such as lower computational complexity, data locality, and easy implementation) compared with the methods formerly known. In particular, this algorithm can be viewed as a generalized *scan* scheme applicable to problem domains of any high dimension and of arbitrary geometry (*scan* is an important primitive of parallel computing). In addition, from implementation results, the hierarchical cluster labeling algorithm has proved to work equally well on MIMD machines, though originally designed for SIMD machines.

Based on this success, we further study the hierarchical structure hidden in the algorithm. Hierarchical structure is a conceptual framework frequently used in building models for the study of a great variety of problems. This structure serves not only to describe the complexity of the system at different levels, but also to achieve some goals targeted by the problem, i.e., an algorithm to solve the problem. In this regard, we investigate the similarities and differences between this algorithm and others, including the FFT and the Barnes-Hut method, in terms of their hierarchical structures.

Contents

List of Figures

Chapter 1

Introduction

1.1 Physics and Computer Science

The connections between physics and computer science have been a fascinating subject attracting the research efforts from scientists in many other areas as well as in these two disciplines. It is interesting to ask if we could apply the methodology learned from computer science to the problems arising in some physical systems and obtain better understanding by rephrasing or reformulating the problems.

Put another way, we may ask whether or not there is an alternative algorithm or computational model derived from well-known routines used in physics to attack some intractable problems existing in other application domains. These questions have stimulated a great deal of studies in several directions. Recent advances are stirring even more interest and makes this subject more important than it was previously recognized.

To answer the first question in part, the most visible progress may be that the concept of *complexity* has been brought to the attention of a large number of physicists when they discuss problems in physics. There are principally two types of complexity that are employed in studying physical systems. The first one is devoted to the discussion of *randomness* observed in the systems where the degree of order or disorder needs to be defined and classified. The work in this direction was initiated in the 1960's by G. Chaitin, A. N. Kolmogorov, *et al.* [23], who first introduced the notion of "*algorithmic complexity*." In brief, the algorithmic complexity of a pattern is the length of the shortest program on a general-purpose computer, a *universal Turing machine*, required to generate the pattern, divided by the size of pattern itself (taking the limit of infinitely large patterns is assumed). This definition of algorithmic complexity seems to be the quantity most closely related to the intuitive notion of randomness. A considerable effort has been made to extend this concept such that the other factors affecting the system can be sufficiently taken into account and the randomness or information of the system can be effectively described (e.g., [12], [116], and the references therein).

The second type of complexity is essentially imported from the notion of *computational complexity* [1], [46] in computer science. In the theory of computational complexity one correlates the complexity of a problem with the time it takes to solve the problem on a computer, normally an abstract computational model like a Turing machine. If a problem can be solved on a Turing machine in a number of steps less than a certain polynomial $P(M)$ of the size of the problem M, assuming the algorithm is found, we say this problem is a P problem. Furthermore, an NP problem is a problem that can be solved in a polynomial time on a *non*deterministic Turing machine. A nondeterministic Turing machine is a Turing machine that allows the verification of several guesses to proceed in parallel. From the experience, an NP problem is an intractable problem with no efficient algorithm and whose solution rapidly becomes impractical with increasing problem size. Whether or not an NP problem can be transformed into a P problem has been an important question seeking answers in mathematics for more than two decades. Many problems in statistical physics have been proved to be NP (or NP-complete, more precisely speaking) [10], [14], [45], [115]. Locating the global minimum energy of some spin glass-like systems is a well-known example that still lacks an efficient algorithm [11], [7].

Using physical analogies to attack some complex problems, which are often NP-complete, has been used in many areas, but originated in physics. This approach is inspired by the recent success of applying artificial neural-net type algorithms to problems in pattern recognition and combinatorial optimization [50], [54]. The feasibility of this methodology is further enhanced by the rapid progress in current parallel computation capabilities. *Physical computation* [42] is introduced as the use of physical methods to describe general complex systems by encompassing a variety of ideas, including statistical physics, simulated annealing, information theory, etc. Confronted with the situation that there is no universally good approach to optimization, this comparatively novel methodology aims at being a general strategy to deal with this sort of very large scale problems [89], [96], [111]. Though it resembles the probabilistic nature of randomized algorithms [3], [61] in mathematics, only physical computation is able to address the problems in a more uniform prescription based on the theories and formalism of physics. More details of its state-of-the-art can be read in [42].

It is a thesis stated by A. Turing and A. Church that Turing machines are capable of solving any effectively solvable algorithmic problems [71]. Namely, Turing-computable functions are the closure set of all functions computable by any mechanical procedure. Worse than the intractable problems like NP-complete ones, there exist infinitely many problems that are "*undecidable*" or uncomputable by any algorithm. Not restricted to the realm of theoretical computer science, a number of problems in the dynamical systems theory have also proved to be undecidable [27], [59], [72], [73]. Frustrated and encouraged by this reality, physicists have speculated (e.g., [83]) and attempted to envisage computational models more powerful than Turing machines, despite the relentless criticism from computer scientists (e.g., [97]). *Quantum computer* [13], [22], [32] and *emergent computation* [40] are two starting steps toward this goal that have received much attention in recent years. The major advance anticipated from quantum computer is that, owing to the nature of quantum superposition, this theoretical machine offers the possibility for massive parallelism within a single piece of hardware. Though this theoretical computer does not surpass Turing machines, it reveals promising prospects by showing more efficient performances than Turing machines on several problems. The realization of this design awaits the breakthrough of future technology. Emergent computation exploits the potentially higher computational capabilities exhibited by a big set of computers interacting collectively and cooperatively. It is an interesting investigation *per se* to see if any computation with better efficiency emerges in the vicinity of a phase transition, if any, in the system. Yet the usefulness of this type of model still receives much skepticism.

1.2 Hierarchical Structures

Hierarchical structure is a conceptual framework frequently used in building models for the study of a great variety of phenomena. The differences in definition, interpretation, and utilization of structures of this sort may appear, at first glance, impossible to reconcile for different problems, though the main underlying ideas are very likely to be the same. In a sense, the hierarchical structure can be visualized as an ordered configuration abstracted from an ensemble of interacting units, and, by some specific governing rules, the interacting units are successively decomposed into nested sub-components at every level of the hierarchy. This structure serves not only to describe the complexity of the system at different levels, but also to achieve some goals targeted by the problem, i.e., an algorithm to solve the problem.

Spin glasses [38], [69] are a very interesting topic in condensed matter physics that provide an inspiring example for hierarchical structures. These types of systems are relatively new (developed in the mid-1970's), and yet have become a rich paradigm for the study of several characteristic behaviors shared by a great many complex systems. Besides randomness, the crucial ingredient to the construction of a spin glass system is frustration. By frustration, we mean the inability of a system to accommodate a configuration that satisfactorily complies with all constraints, i.e., the competition among conflicting interactions. Owing to this paradoxical feature, the system exhibits a multitude of nearly satisfactory accommodations, which correspond to a very large number of near equilibrium states at low temperature.

Ultrametricity [85] entered physics through recent investigations of some spin glass systems in the context of mean field theory. This concept has appeared now and then in the mathematical literature under the name of p-adic numbers, originating in 1897. In mathematical terms, an ultrametric space is a space replacing the triangular inequality

$$d(A,C) \leq d(A,B) + d(B,C) \tag{1.1}$$

by a stronger inequality

$$d(A,C) \leq max\{d(A,B), d(B,C)\}. \tag{1.2}$$

The pictorial explanation of this metric is depicted in Fig.1.1. A very short time after its introduction into spin glass theory, the concept is discussed in several fields: statistical physics of disordered systems, combinatorial optimization, neural networks, conformational structure of proteins, evolution and diffusion (see the reviews in [85] and [114]).

There is a considerable amount of indirect evidence that the free energy surface of a spin glass is *rough on many scales* [99]. Specifically, the free energy surface of a spin glass system may be filled with valleys of different heights, and valleys within valleys (Fig. 1.2). In other words, the local minima of the free energy surface may undergo multifurcation over and over again, which as a result gives rise to a hierarchical organization on the free energy surface. A number of toy

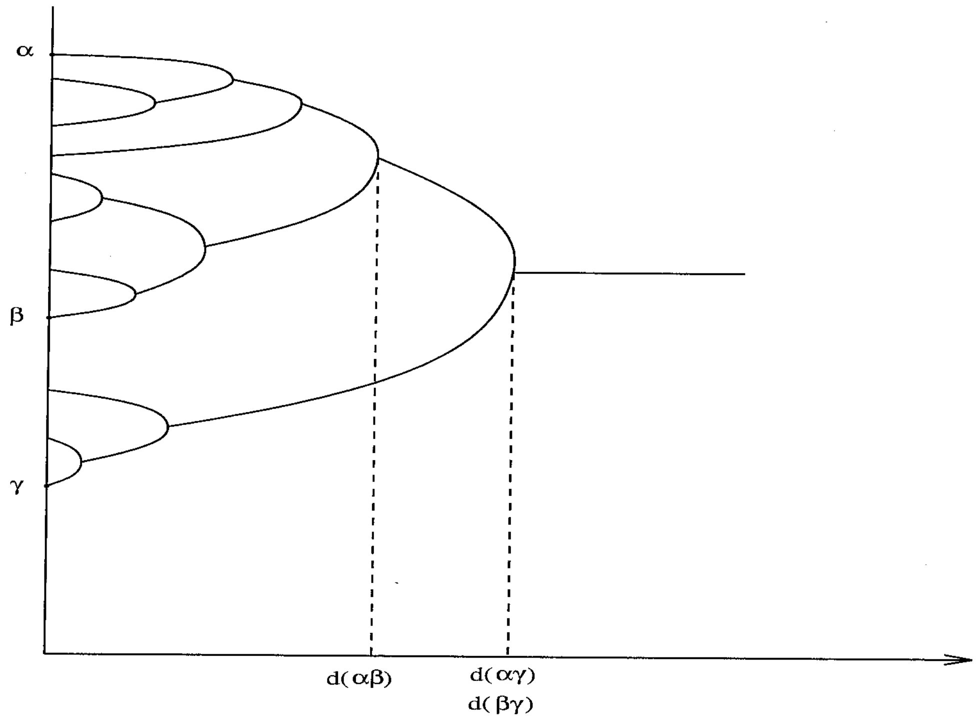


Figure 1.1: An ultrametric tree.
In an ultrametric space distance between two states is defined as the number of levels which must be traced back before they merge.

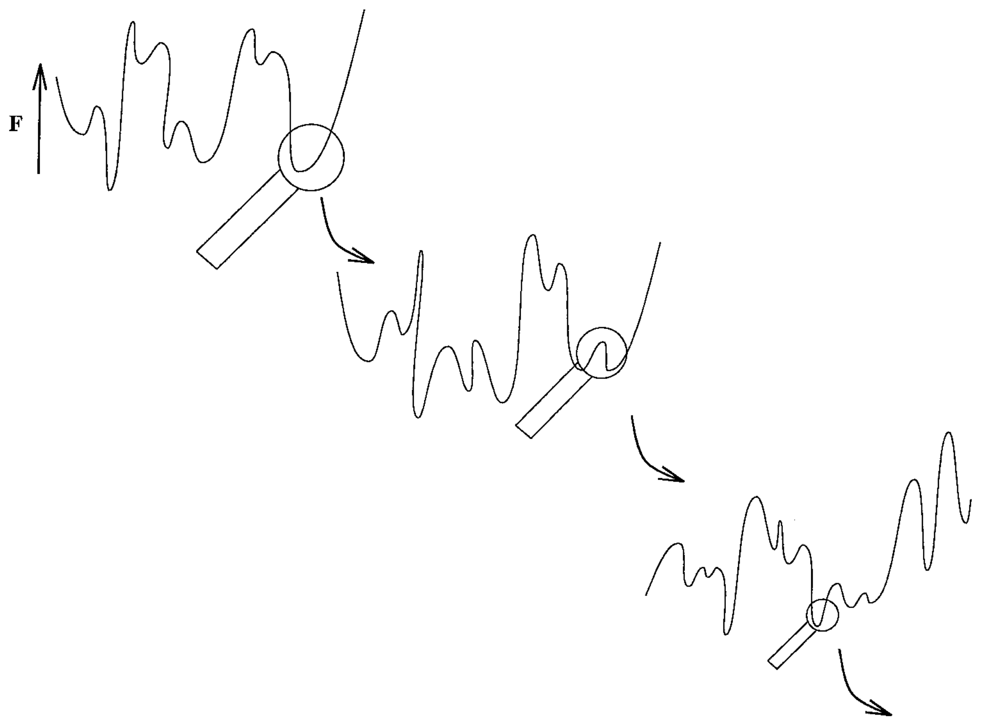


Figure 1.2: The Multiscale resolution.
The free energy surface at three different resolutions, which can be controlled by temperature and length scale.

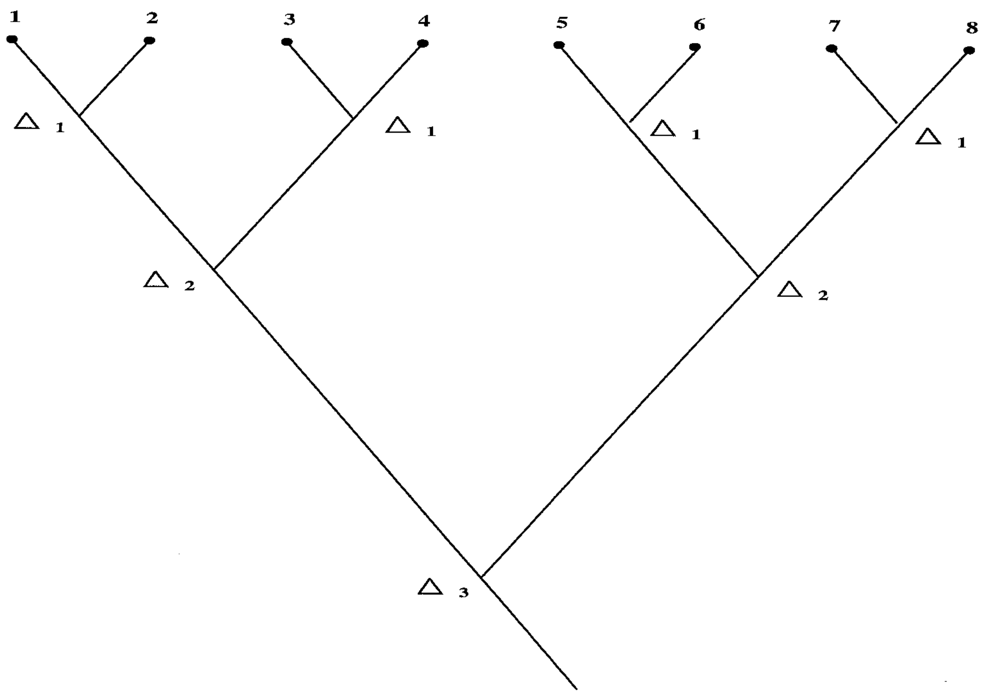


Figure 1.3: A toy model of hierarchy-type.
A sketch of a toy model, where the distance between sites is the number of steps toward the root of tree before paths from two sites converge. Different probability weights can be specified at different levels of transition.

models [53], [55], [76], [80], [93] have been invented, based on different types of hierarchical organization (e.g., Fig. 1.3), to imitate some interesting yet not totally understood phenomena such as irreversibility, slow relaxation, relaxation of different time scales, etc., observed in many complex systems.

Another excellent device used to study complex systems are *cellular automata* [109]. This model was originally introduced in 1948 by John von Neumann as a possible idealization of biological systems. In an attempt to "abstract the logical structure of life," he developed an automaton network that could exhibit one of the major features of life: self-reproduction. Mathematically, cellular automata are a skeletal scheme of natural systems in which space and time are discrete, and the physical quantities take on the finite set of discrete values; the automaton network evolves in discrete steps, the sites being simultaneously updated by a deterministic or nondeterministic rule. Typically, only a finite number of neighbors are involved in the updating of any site.

The interest in cellular automata was reawakened when Frisch, Hasslacher and Pomeau proposed their "lattice gas automata" for the 2-dimensional Navier-Stokes equations in 1986 [44]. Their lattice gas automata are a clear-cut model with a oversimplified "microdynamics," in which particles move and collide with certain

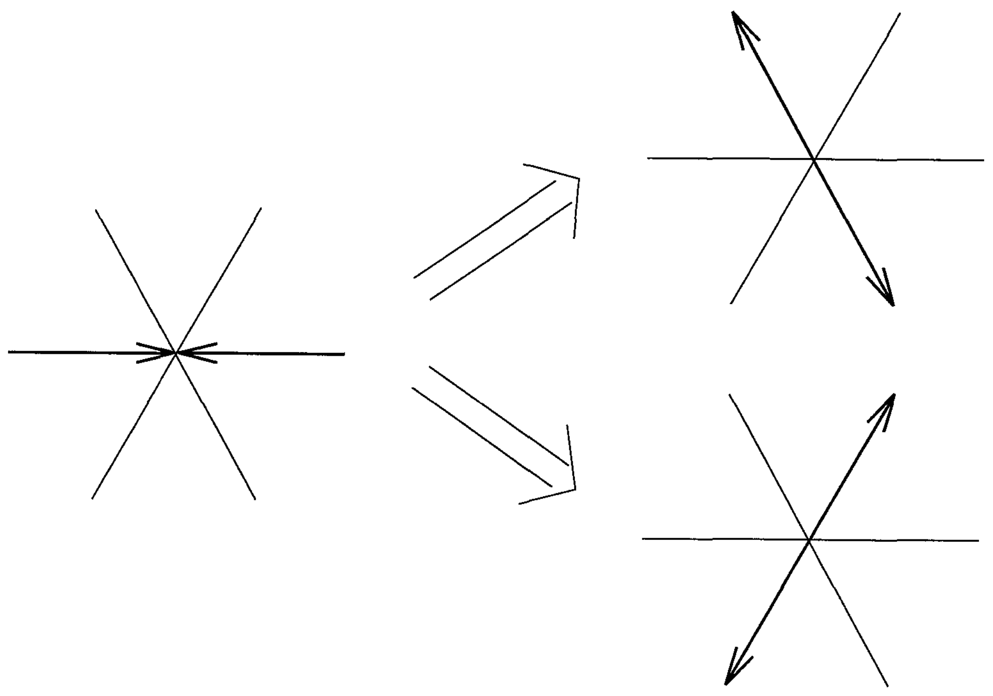


Figure 1.4: A collision rule in a CA model.
Performing a simple collision rule on a triangular lattice can simulate complicated fluid dynamics.

deterministic or nondeterministic rules on a triangular lattice, e.g., Fig. 1.4. By taking an appropriate continuum limit of this microdynamics, the macrodynamic description of fluids, the Navier-Stoke equations, is recovered. Unfortunately, even after several variant models were proposed, the cellular automaton fluid is still confined to the low Reynold's number regimes and offers no help to the study of turbulence, the most interesting phenomenon in fluid dynamics. However, cellular automata may still provide insight into the investigation of models which are as simple as possible in construction, yet capture the essential mathematical features necessary to reproduce the deserved complexity. It is now believed that cellular automaton models of hierarchy-type are well suited to more accurately simulate the essential features of complex systems (see [58] and the references therein).

These hierarchical cellular automaton models can be roughly understood as cellular automata of several layers organized in a tree structure, rather than of only one layer, as their predecessors. In the system, a unit interacts strongly with the other units of the same level; A unit at the lower level is "slaved" to the unit of its upper level, while there is a small feedback from the lower level to the higher level. Obviously, this arrangement is a phenomenological imitation of the dynamics seen in many natural or social systems. The design of hierarchical cellular automata is mostly inspired by the work on toy models mentioned earlier (e.g., [53], [55], etc.). However, the hierarchical cellular automata have much more freedom to vary the governing dynamics. A hierarchy of coupled iteration rules are from the typical recipe for the dynamics, where one has the freedom to decide the form of iterations and couplings with neighbors of the same level and sites at different levels, yet still to keep the rules as mathematically simple as possible. Complex systems as diverse as biological information processing, economic activities, and turbulence of fluids are currently the problems that the researchers are most interested in approaching with these modeling schemes.

1.3 Organization of This Work

In spite of the exciting progress in strengthening the connections between physics and computer science and the significant value of studying the hierarchical models, we will only concentrate our attention on the discussion of the hierarchical algorithms in this thesis. As mentioned above, if a hierarchical structure is designed to fulfill the goal described by a problem, this structure is in fact an algorithm to solve the problem, and its structure is that of a hierarchical algorithm. [1]

The main algorithm to be studied in the thesis is called the *hierarchical cluster labeling algorithm.* This study is essentially motivated by the inefficient performance of the *cluster Monte Carlo algorithms* for spin models on parallel computing

[1]Quite often, a model simulating a physical system is called an algorithm to unfold the properties of the system, like Frisch *et al.*'s lattice gas automata also known as the cellular automaton fluid algorithm. But here, we will call it a model to avoid any possible confusion.

systems. In Chapter 2, we describe the cluster Monte Carlo algorithms, mostly based on the work of Swendsen and Wang, and elaborate the motivation to introduce a better parallel algorithm. Chapter 3 and Chapter 4 are entirely devoted to the discussions of the hierarchical cluster labeling algorithm. We emphasize the *divide-and-conquer* methodology employed in this algorithm and its recursive nature. Owing to these characteristics, we are able to have a clear picture of the whole course of the algorithm. The algorithm can easily be applied in any dimension and to arbitrary geometry. Based on this success, the hierarchical cluster algorithm can be even further extended as a generalized scheme to perform any parallel *prefix* calculation on an arbitrary problem domain. This scheme is more flexible and powerful than the *scan* mechanism previously proposed. Chapter 5 is a detailed exposition of fast Fourier transform (FFT) and a brief review of the Barnes-Hut method. The purpose of this chapter is to discern the similarities and differences between these two and the hierarchical cluster labeling algorithm, in terms of the scenarios of divide-and-conquer and shuffling. Since the technique of shuffling used in the algorithm design here is similar to the idea of encoding in information theory and this theory has become very useful and popular in elucidating many notions in physics, we briefly describe the information theory formalism in the Appendix. The discussion is centered on the Huffman code which plays a primary role in information theory and is an algorithm of the hierarchy-type as well. Finally, in Chapter 6, we conclude the thesis by pointing out the prospects and further research directions for the hierarchical algorithms.

Chapter 2

Cluster Labeling

2.1 Motivation

Cluster labeling has been an important problem existing in a wide variety of studies in natural sciences and engineering applications. It is an unavoidable problem whenever the specification of a whole cluster of elements with the same assigned properties in the target configuration arises during the problem-solving procedure. This problem has been studied for a long time in different disciplines and referred to by various names, like *connected component labeling, cluster identification*, and *undirected graph computation* [92], [106]. Earlier, scientists in different disciplines only studied this problem based on the experiences within their own discipline, without knowing much about progress in other areas. Along with the increased use of high performance computers, the exchange of know-how between different disciplines has become crucial.

Since Swendsen and Wang proposed the "*cluster*" Monte Carlo simulation algorithm for Potts spin models [104], the problem of finding connected components on a physical configuration has steadily attracted more attention among physicists. In contrast to the traditional single-spin-flip-mode (Metropolis) algorithm, Swendsen and Wang's method is a very different type of collective-mode updating method. Specifically, instead of updating the spins one by one, the cluster algorithm considers connected clusters of spins with the same orientation as one template, and updates the whole template as one single spin. The main purpose of adopting the collective-mode updating is to combat the effect of *critical slowing down* which afflicts the single-mode Metropolis method by reducing the efficiency of the algorithm to zero near the phase transition as the size of the lattice increases

[8].

In this section, we briefly review the theoretical foundations of the Swendsen-Wang algorithm. Instead of following the original exposition of Swendsen and Wang, we will proceed in a form and use the notations first introduced by Sokal and Edwards [35] and later modified by Binder *et al.* [79], which turn out to be more flexible and useful in addressing physical properties associated with clusters in the system. Given a lattice with Potts spins $\sigma_i = 1, \ldots, q$ on the site and bond variables $n_{ij} = 0$ (open, i.e. disconnected), $n_{ij} = 1$ (closed, i.e. connected) on the edges, the *joint probability distribution* of a certain realization of Potts and bond variables is defined by

$$P(\sigma, n) = Z^{-1} \prod_{<i,j>} \left[(1 - p_{ij})\, \delta_{n_{ij},0} + p_{ij}\, \delta_{\sigma_i,\sigma_j}\, \delta_{n_{ij},1}\right] \tag{2.1}$$

with

$$Z = \sum_{\{\sigma\}} \sum_{\{n\}} \prod_{<i,j>} \left[(1 - p_{ij})\, \delta_{n_{ij},0} + p_{ij}\, \delta_{\sigma_i,\sigma_j}\, \delta_{n_{ij},1}\right] . \tag{2.2}$$

The p_{ij} are given by the couplings J_{ij} between the spins, $p_{ij} = 1 - \exp(-J_{ij})$, where $J_{ij} \geq 0$ for all i, j ("ferromagnetism"). This is often called the *Fortuin-Kasteleyn-Swendsen-Wang (FKSW) model* (in [35], $P(\sigma, n)$ is denoted as $\mu_{FKSW}(\sigma, n)$).

Summing over all bond configurations yields

$$\begin{aligned} P(\sigma) &= \sum_{\{n\}} P(\sigma, n) && (2.3) \\ &= Z^{-1} \prod_{<i,j>} \sum_{n_{ij}=0,1} [(1-p_{ij})\ \delta_{n_{ij},0} + p_{ij}\ \delta_{\sigma_i,\sigma_j}\ \delta_{n_{ij},1}] \\ &= Z^{-1} \prod_{<i,j>} [(1-p_{ij}) + p_{ij}\ \delta_{\sigma_i,\sigma_j}] \\ &= Z^{-1} \exp\left[\sum_{<i,j>} J_{ij}\,(\delta_{\sigma_i,\sigma_j} - 1)\right] \\ &= Z^{-1} \exp[-H(\sigma)] && (2.4) \end{aligned}$$

with $Z = \sum_{\{\sigma\}} \exp[-H(\sigma)]$, respectively, and $H(\sigma) = \sum_{<i,j>} J_{ij}\ (1 - \delta_{\sigma_i,\sigma_j})$ the Hamiltonian of the Potts model (thus $P(\sigma)$ is denoted by $\mu_{Potts}(\sigma)$ in [35]). As expected, the Potts model is "recovered" from the FKSW model by "smearing" the bond configurations.

On the other hand, evaluating the sum over all spin configurations leads to another distribution

$$\begin{aligned} P(n) &= \sum_{\{\sigma\}} P(\sigma, n) \\ &= Z^{-1} \sum_{\{\sigma\}} \left[\prod_{<i,j>,n_{ij}=1} p_{ij}\ \delta_{\sigma_i,\sigma_j} \prod_{<i,j>,n_{ij}=0} (1\ -\ p_{ij}) \right]. && (2.5) \end{aligned}$$

Noticing that all the terms in the sum with a closed bond between two spins in distinct states vanish, one can further obtain

$$P(n) = Z^{-1} \sum_{\{\sigma^n\}} \left[\prod_{<i,j>, n_{ij}=1} p_{ij} \prod_{<i,j>, n_{ij}=0} (1 - p_{ij}) \right], \tag{2.6}$$

where σ^n denotes a spin configuration compatible with the restriction for two spins to be parallel, i.e. in the same Potts state, if connected by a closed bond. Hence, the terms in the sum are now independent of the spin configuration. Namely, given the bond configuration, the sum merely counts the number of compatible spins configurations. If a cluster is defined as the set of bond-connected spins, it follows that

$$P(n) = Z^{-1} \prod_{<i,j>, n_{ij}=1} p_{ij} \prod_{<i,j>, n_{ij}=0} (1 - p_{ij})\, q^{(n)} \tag{2.7}$$

with q the number of possible spins orientations, i.e. Potts states, and (n) the number of clusters of the given bond configuration $\{n\}$; equivalently,

$$Z = \sum_{\{n\}} \left[\prod_{<i,j>, n_{ij}=1} p_{ij} \prod_{<i,j>, n_{ij}=0} (1 - p_{ij})\, q^{(n)} \right]. \tag{2.8}$$

This is simply the partition function of the *random-cluster model* first proposed by Fortuin and Kasteleyn [41] ($P(n)$ is thus denoted by $\mu_{RC}(n)$ in [35]).

Up to now, the following facts about the FKSW model have been verified [35], which are of both analytic and numerical interests:

a) $Z_{Potts} = Z_{FKSW} = Z_{RC}$.

b) The probability distribution of $P(\sigma, n)$ (or μ_{FKSW}) on the Potts variables $\{\sigma\}$ (integrating out the $\{n\}$) is precisely the $P(\sigma)$ for the Potts model (or $\mu_{Potts}(\sigma)$).

c) The probability distribution of $P(\sigma, n)$ on the bond variables $\{n\}$ (integrating out the $\{\sigma\}$) is precisely the $P(n)$ for the random-cluster model (or $\mu_{RC}(n)$).

The conditional distributions of $P(\sigma, n)$ are also simple:

d) The conditional distribution of the $\{n\}$ given the $\{\sigma\}$ is as follows: independently for each bond $\{i, j\}$, one sets $n_{ij} = 0$ in case $\sigma_i \neq \sigma_j$, and sets $n_{ij} = 0, 1$ with probability $1 - p_{ij}$, p_{ij}, respectively, in case $\sigma_i = \sigma_j$.

e) The conditional distribution of the $\{\sigma\}$ given the $\{n\}$ is as follows: independently for each connected cluster, one sets all the spins σ_i in the cluster to the same value, chosen equiprobably from $\{1, 2, \ldots, q\}$.

In brief, exploiting facts (b)–(e), the Swendsen-Wang algorithm (SW), simulates the joint model (2.1) by alternately applying the conditional distributions (d) and (e) — that is, by alternately generating new bond variables (independent of the old ones) given the spins, and new spin variables (independent of the old ones) given the bonds. Therefore, Swendsen-Wang algorithm consists of assigning a new random Potts spin value to each cluster, i.e., with all the sites in a given cluster getting the same value, after the clusters are constructed by introducing bonds with probability $p = 1 - \exp(-J)$ (J_{ij} is normally set to be constant J in most of the implementations) that connects sites with the same spin. By erasing bonds we are left with a new Potts spin configuration. This new configuration can differ substantially from the original one since large clusters can be altered in one single step, thereby introducing strongly nonlocal moves in the system. The critical slowing down is ameliorated by this nonlocal updating algorithm through a drastic reduction in the value of the *critical dynamical exponent.* The critical dynamical exponent, normally denoted as z, is a measurement of the efficiency of an updating method near the point of phase transition. It can be proved that the statistical error in a Monte Carlo-type algorithm is approximately proportional to $\xi^{z/2}$ near criticality [68], where ξ is the spatial correlation length. The experimental data show, for instance, that z is reduced to at most 0.35 by Swendsen and Wang's algorithm from 2.125 measured from the traditional (Metropolis) Monte Carlo simulations on the two-dimensional Ising model [104].

After the success of Swendsen and Wang's cluster Monte Carlo algorithm, there has been an explosion of work devoted to understanding and improving the algorithm [79], [87], [88], [105]. Another successful variant of the cluster algorithm has been designed by Wolff [110]. In this algorithm only a single cluster is generated by a randomly chosen spin, applying the same bond probabilities as for the Swendsen-Wang algorithm. Since only one cluster of spins is updated at each step of the Monte Carlo procedure, Wolff's algorithm achieves the best performance on a sequential computer. Nevertheless, Swendsen and Wang's method is better suited for parallelization. In addition, many cluster algorithms for more complicated spin systems, e.g., the fully frustrated [37], [56], [57], the spin glass-like [26], etc., have also been introduced and intensively studied. Although these Swendsen-Wang-type algorithms perform impressively well, we understand very little about *why* the critical dynamical exponents take the values they do [98].

In implementing these cluster algorithms, whether sequentially or in parallel, the major time-consuming bottleneck is in identifying the cluster(s). Thus labeling the clusters in a physical configuration has turned out to be a problem of importance to physics, rather than confined only to engineering and computer science. A couple of cluster labeling algorithms for sequential computers have long existed and been used by physicists when dealing with percolation-like problems [9]. However, we are now interested in designing their parallel versions in order to take advantage of the large gain in speed provided by parallel computers currently being developed.

2.2 Graph Algorithms

The first way of designing a parallel algorithm is to detect and exploit any inherent parallelism in an existing sequential algorithm. In spite of its straightforwardness, blindly transforming a sequential algorithm to parallel form is often a mistake. Some problems have strong sequential tendency and consequently their sequential algorithms have no obvious parallelization. The algorithm adapted from such a sequential algorithm for this kind of problems will exhibit poor speedup. Unfortunately, the cluster labeling in the spin models is just such a problem. The growing of clusters in the spin configuration is very similar to the *information passing procedure* seen in many phenomena, and thus is inherently sequential in nature. In addition to these obstacles, the hardware architecture often demands a new approach. Therefore, it is often better to "start from scratch." Before doing so, we would like to see what has been previously accomplished on this problem.

As mentioned earlier, the cluster labeling problem is sometimes called *undirected graph computing*, and the algorithms developed to tackle this problem generally require the aid of graph theory. In a broader context, these algorithms are always referred to as *graph algorithms* [64]. Alternatively speaking, the image component labeling problem is a special case of the graph connected component labeling problem. In these kinds of problems, the idea is, for a given graph, to label each vertex in such a way that two vertices get the same label number if and only if they are connected by a path in the graph.

There are three common approaches to find the connected components of an undirected graph. The first one is to use some form of search, such as depth first or breadth first. This kind of search seems to be an inherently sequential process, since searching always occurs along a single edge from a single vertex, restricting opportunities for parallelism. In fact, it has been discussed and conjectures that this type of searching algorithm is hardly parallelizable [48]. The second approach is to solve the problem by finding the transitive closure of the adjacency matrix of the graph.[1] This method is very straightforward by simply calculating the product of adjacency matrix of a graph. However storing the information of the adjacency matrix can become very costly in memory and computing its closure always consumes too much processor time, when the graph is very large. This method is not considered the best option in searching for the parallel version.

The third approach collapses vertices into larger and larger sets of vertices until each set corresponds to a single connected component and it has proved to be the best suited strategy allowing parallelization. The most important progress on this study may be the algorithm presented by Shiloach and Vishkin [95]. Given an undirected graph $G = (V, E)$ and using a special parallel computation model, their algorithm takes $O(\log\, n)$ time steps to complete the labeling, employing $n + 2m$ processors where $n = |V|$ and $m = |E|$. The computation model used in the algorithm requires that all the processors have access to a common memory and are capable of simultaneous reading and writing from the same location. The

[1]An unweighted graph can be uniquely represented by an $n \times n$ *adjacency matrix* A, with one row and one column for each vertex. The element of A at row i and column j is equal to 1 if and only if there is an edge from vertex i to vertex j; the value is 0 otherwise. The closure of a graph is represented by an $n \times n$ matrix, denoted as C, whose elements c_{ij} are defined to be 1 if there is a path from vertex i to vertex j and defined to be 0 otherwise. It can be proved [1] that $C = (I + A)^m$, where $m = 2^{\lceil \log(n-1) \rceil}$. Notice that the matrix product here is performed according to the rule of *Boolean matrix multiplication* defined as follows: If X, Y, and Z are $n \times n$ Boolean matrices where Z is the Boolean product of X and Y, then, for $i, j = 1, 2, \ldots, n$, $z_{ij} = (x_{i1}$ **and** $y_{1j})$ **or** $(x_{i2}$ **and** $y_{2j})$ **or** ... **or** $(x_{in}$ **and** $y_{nj})$.

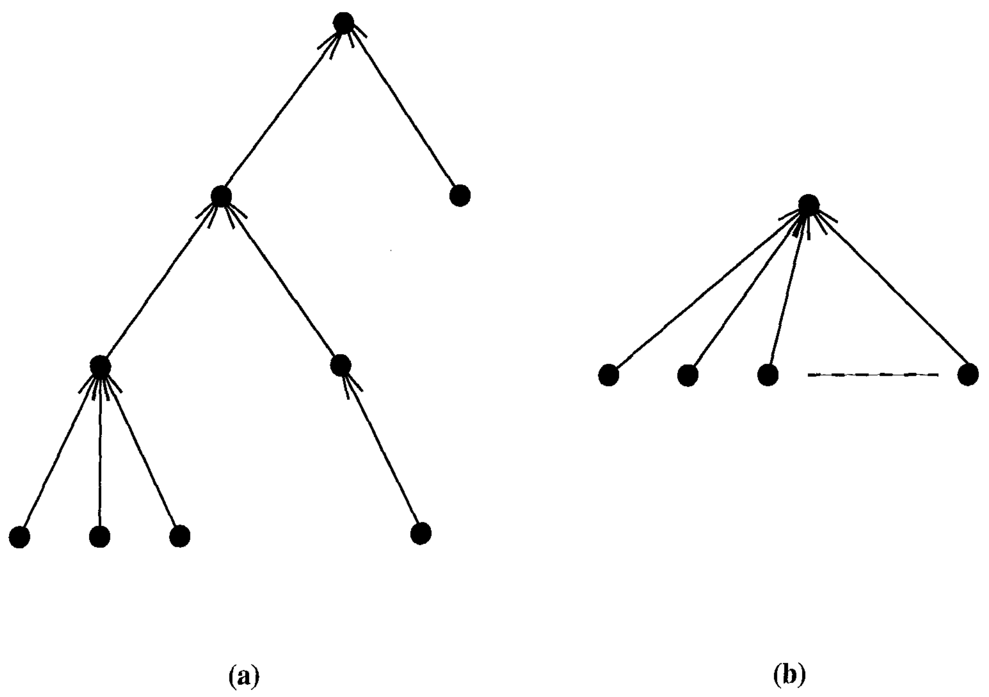


Figure 2.1: The rooted tree and star.
(a) An example of rooted tree; (b) an example of rooted star.

algorithms invented prior to that of Shiloach and Vishkin only achieve the time efficiency of $O(\log^2 n)$ or worse and use just as many number of processors, see e.g., [52], [65], [77].

Though Shiloach and Vishkin's algorithm is intuitively simple, it is worthwhile to introduce the jargon used in the algorithm for later convenience.

Definition.

(1) A *rooted tree* is a directed graph satisfying:

(a) its underlying undirected graph is a tree;

(b) it has a vertex r called the *root* such that there exists a directed path from each vertex to r (e.g., Fig. 2.1a).

(2) A *rooted star* is a rooted tree in which each vertex is connected directly to the root, i.e., a rooted tree of height 1 (e.g., Fig. 2.1b).

During the whole course of the algorithm, associated with each vertex ν has a pointer $D(\nu)$ which points to another vertex or to itself. The pair $(\nu, D(\nu))$ pairs can be viewed as a directed edge and this defines a set of rooted trees, with self loops at the root of the trees. The set of $(\nu, D(\nu))$ is called the *pointer graph* which is always a forest of rooted trees plus self-looping that only occurs in the roots. The entire algorithm essentially consists of intermixed applications of two primitive operations.

Definition.

(1) The *shortcut operation*, $D(\nu) \leftarrow D(D(\nu))$, replaces the value $D(\nu)$ by $D(D(\nu))$ (see Fig. 2.3).

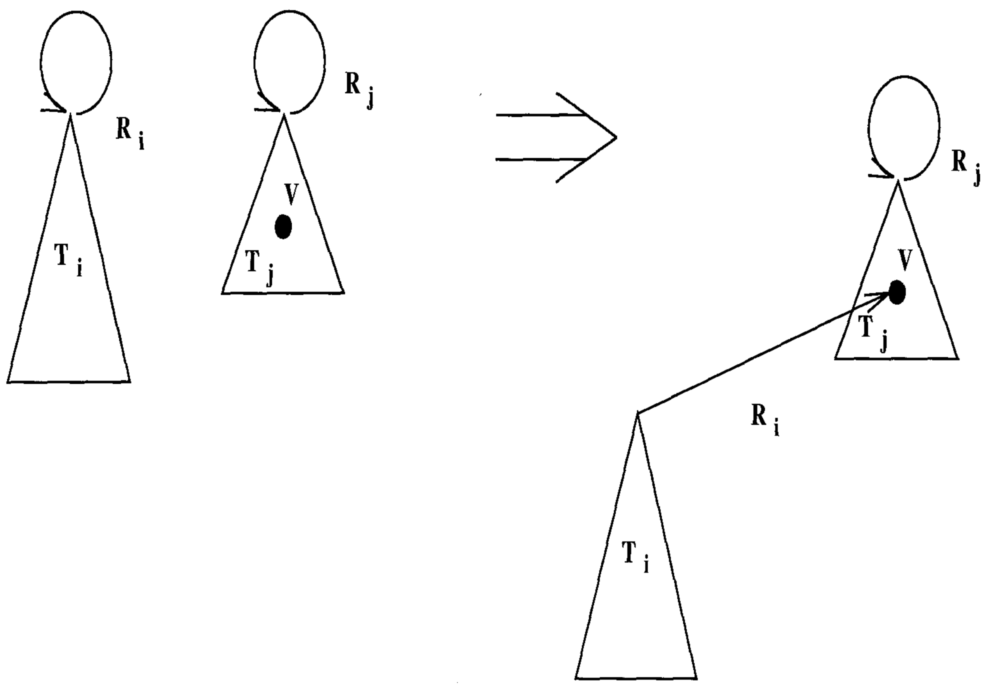


Figure 2.2: The tree hooking operation.

(2) Let $P = (V, D)$ be a pointer graph with rooted trees $T_1, ..., T_k$, $k \geq 2$, and let r_i be the root of T_i. The operation

$$D(r_i) \leftarrow \nu, \qquad \text{where} \quad \nu \in T_j \quad \text{and} \quad j \neq i,$$

is called *hooking* of T_i onto T_j (see Fig. 2.2).

As the algorithm proceeds, the *hooking* operation hooks one tree onto another. This operation makes the number of trees decrease while individual trees expand or even disappear. The trees are also subject to the *shortcut* operation that decreases their height. At the end of the algorithm the vertices of each connected component form a rooted star in the pointer graph. Thus, the question of the form "Do ν_i and ν_j belong to the same connected component?" can be answered in constant time. The computation model used in the algorithm is normally called CRCW PRAM (*Concurrent-Read Concurrent-Write Parallel Random Access Machine*) [64], by which these operations take $O(1)$ time. Under this assumption, one finds that the total time taken is $O(\log n)$, or $\lfloor \log_{3/2} n \rfloor + 2$ precisely speaking. Since the *shortcut* operation is applied to each vertex ν, the height of the tree containing ν reduces by at least a factor of 2/3 whenever the tree is not a rooted star (e.g., Fig. 2.3).

CRCW PRAM is the most powerful parallel computation model [63], [82], though it is impractical to implement physically. Nevertheless, this model furnishes a valuable theoretical paradigm for the discussion of parallel algorithms, and algorithms designed on it can be translated into algorithms on real, more feasible machines with less power. A considerable amount of research has been undertaken to translate Shiloach and Vishkin's algorithm onto existing parallel machines with various architectures (e.g., [28] and the review in [2]). In fact, the

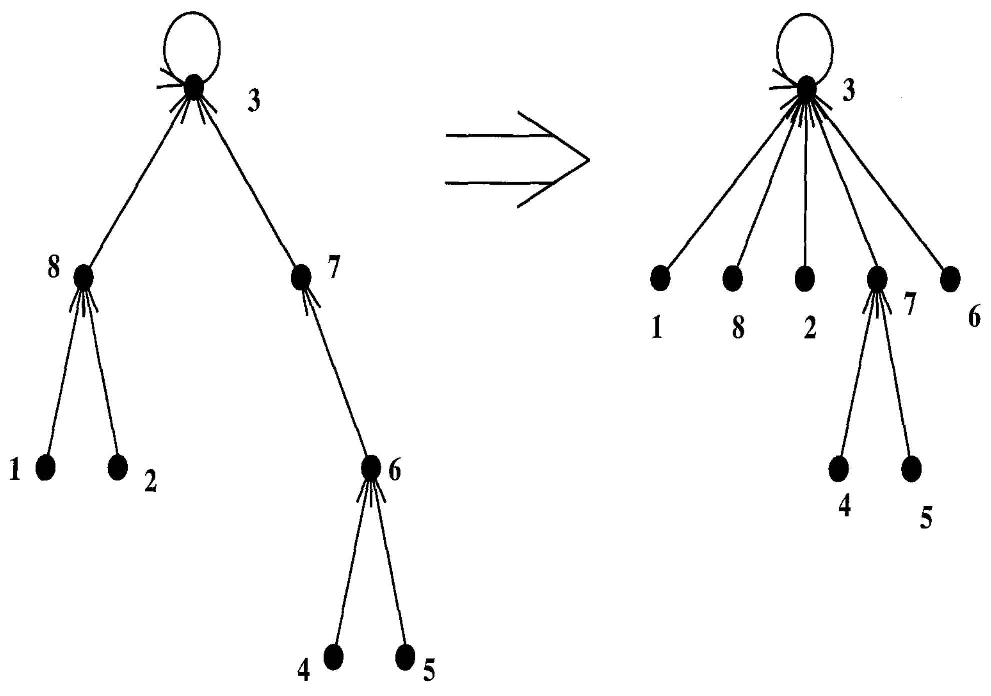


Figure 2.3: The shortcut operation.

cluster identification problem discussed here belongs to a larger family of problems called *clustering.* In the clustering problems [86], [92], one is concerned with partitioning a set into groups according to certain properties of the elements. These properties and the partition rules are usually more broadly defined and the problems are often much more complicated than the cluster identification problem. In contrast to the purely deterministic scenario used in the cluster identification, most clustering problems involve different forms of statistical considerations as the partition proceeds. The clustering routines that play a key role in the *unsupervised learning* of artificial neural-net theory [50], [92] are a very good example. Designing faster probabilistic partition schemes in order to generate better learning capabilities is one of the major goals in this subject.

Another interesting approach to cluster identification using *randomized algorithms* has recently been presented by Gazit [47]. The main idea of the algorithm is to find a way to separate vertices with a large number of incident edges, called *extrovert* vertices, from those with a small number of incident edges, called *introvert* vertices. To separate them by counting the edges takes too much time. Therefore a statistical test is used: a sample of edges is taken and only the vertices they hit are considered. Obviously, an extrovert vertex is more likely to be chosen by this method because it has more edges than an introvert vertex. In a randomized algorithm, each processor has access to a random-number generator which makes the statistical test possible. Through rigorous probabilistic analysis, Gazit proves that his algorithm has an expected running time of $T = O(\log n)$ with $P = O((m+n)/\log n)$ processors, where m and n are again the numbers of edges and of vertices, respectively. The probability that the algorithm runs longer than expected is at most $(2/e)^{n/\log^k n}$ for some $k \leq 4$. In other words, Gazit's probabilistic algorithm is *optimal* in the sense that the product $P \cdot T$ is a linear function of the input size. The algorithm requires $O(m+n)$ space, which is again just the input size, so it is *optimal* in space as well. This progress in some way addresses Shiloach and Vishkin's conjecture that the barrier of $\log n$ cannot be surpassed by using any polynomial number of processors. If this conjecture is true, Gazit's algorithm has achieved the lower bound for running time with an optimal number of processors. The question of whether these bounds can be achieved by a deterministic algorithm remains open.

The research into randomized algorithms [3], [49], [61], [84] is a relatively new yet increasingly active area. In general, a randomized algorithm is one that receives, in addition to its input data, a stream of random bits that it can use for the purpose of making random choices. Even for a fixed input, different runs of a randomized algorithm may give different results. Thus it is not surprising that a description of the properties of a randomized algorithm will involve probabilistic statements in terms of one or more random variables. By now it is recognized that, in a wide range of applications, randomization is an extremely important tool for the construction of algorithms. There are two principal advantages that randomized algorithms often have [61]. First, the execution time or space requirement of a randomized algorithm is generally smaller than that of the best deterministic algorithm for the same problem. But even more surprisingly, if we look at the various randomized algorithms that have been invented, we find that they are in fact very simple to understand and to implement; often, the introduction of randomization suffices to convert a simple and naive deterministic algorithm with bad worst-case behavior into a randomized algorithm that performs well with high probability on every possible input. In this regard, it is desirable to find the corresponding randomized algorithms for the NP-complete problems such that these problems can be solved in a "reasonable" amount of time, i.e., polynomial time. In a certain sense, randomized algorithms share similar ideas with the physical computation mentioned in Chapter 1. The most important advantage of randomized algorithms over their deterministic counter parts may be their close relation with physical statistical mechanics [115]. The exact form of this relation has not been made very clear yet, and is therefore worth more devoted attention. A great reduction of computational complexity using a certain number of processors with a cleverly designed randomized algorithm is not the whole story. Implementing an algorithm in a real computing environment is also one of the essential steps needed to evaluate the method in practice. Thus, to analyze the performace of algorithms of this category on existing computers is an important task which deserves further attention.

Chapter 3

The Hierarchical Cluster Labeling Algorithm

3.1 Divide and Conquer

The dynamic irregular nature of clusters in a spin configuration or a pattern recognition domain complicates the task of finding good parallel algorithms. This is particularly true on SIMD machines [43] where the high degree of parallelism conflicts with the inherent sequential process, i.e. cluster growing, which defines the problem. The essential idea behind the algorithm we are going to describe in this chapter is the construction of a static hierarchical structure on the configurations of the target problem. Through this hierarchical structure, the possible configurations are recursively decomposed into many components. The connected components of clusters are labelled by a unique number, which, during the course of the algorithm, is passed through calling the nonlocal communication routines provided by the machines. The computation proceeds in a recursive manner upwards through the hierarchical structure until all connected components have been labeled. One can easily see that this hierarchical scenario has invoked a concept rooted in the technique of *divide-and-conquer* [1].

Divide-and-conquer (DC) is a problem-solving methodology that involves partitioning a problem into subproblems, solving the subproblems, and then combining the solutions of the subproblems into a solution for the original problem. The methodology is recursive; that is, the subproblems themselves may be solved by the DC technique. Many important problems in scientific computing are known to have efficient DC solutions. Examples are fast Fourier transformation (FFT) [78], Strassen's matrix multiplication [102], prefix algorithms [64], Cuppens method for solving matrix eigenvalues [33], and several sorting algorithms, which include quick sort, bitonic sort [62], etc. All of these are essentially some manifestation of the divide-and-conquer methodology. Recursively partitioning the corresponding problem configuration, like matrix elements in matrix multiplication and sequences of numbers in sorting, the hierarchical structure is defined for each configuration. By this tree-like structure, a drastic speedup algorithm is obtained, e.g., from N^2 steps of Fourier transformation to $N(\log N)$ steps for the FFT. The idea is not confined to seeking better algorithms. The notion of hierarchical structure has also been applied to the design of computer architectures. The well-known *hypercube* architecture [43] is an embodiment of a hierarchical structure inside the computer hardware. The concept has been pushed much further, so that a large number of computational activities of highly *heterogeneous* nature [34] can proceed in a single computation environment. The so-called *hierarchical multiprocessors* [29] meeting these needs may, in the near future, be a present day mainstream high performance computer. The hierarchical structure is also a central idea in the construction of any conventional programming language. It is also a widely-held opinion that divide-and-conquer is an effective programming paradigm for parallel computers. In particular, a new parallel programming language, *Divacon* [74], has also been invented based on the scheme of divide-and-conquer. It is expected that parallel programming languages of this type may revolutionize the way of thinking and programming on parallel computers.

Returning to the major subject of this chapter, in the next section we will introduce the cluster labeling algorithm in one dimension, and its extension to higher dimensions will be presented in the following section.

3.2 The Algorithm in One Dimension

The primary portion of the algorithm is composed of a special initialization of the label number on each site of the target problem configuration. This initialization installs an appropriate hierarchical structure on the problem from the very beginning. In addition to the special initialization, another elementary part of the algorithm consists of mixed applications of local and nonlocal communication routines furnished by the parallel computer being used.

Suppose we are given a strip of length N as Fig. 3.1a, where N is at first assumed to be a power of 2, $N = 2^k$, for pedagogical simplicity. In this illustration

($k = 4$), the strip is to grow into one single cluster after the whole identification procedure is finished. In this array, from left to right, every site has a coordinate, *coord*, ranging from 0 through $N - 1$. To begin, each site is assigned a unique label number, *label_value*, by the following formula.

$$\mathit{label\_value} = a_{k-1} + 2a_{k-2} + 2^2 a_{k-3} + \ldots + 2^{k-1} a_0 + (2^k a_k - 1), \qquad (3.1)$$

where $a_0, a_1, \ldots, a_k = 0$ or 1 are defined by the dyadic expansion:

$$\mathit{coord} + 1 = a_k 2^k + a_{k-1} 2^{k-1} + a_{k-2} 2^{k-2} + \ldots + a_1 2 + a_0.$$

The evolution rule is that the label numbers are replaced by smaller ones after checking the label numbers of connected clusters. It is very similar to the ants-in-the-labyrinth method[1] which is the most obvious one for identifying a single cluster of connected sites. The difference is that here we consider each cluster as a template to be identified. At the first step of identification, every site carrying its own unique label number can be regarded as a cluster composed of only one site. Actually, this is the view taken for clusters in the implementation in order to keep the consistency of definition. After the first iteration (Fig. 3.1b), only one half of

[1]The reason for its name is that we can visualize the method as follows [9]. An ant is put somewhere on the problem domain, say a lattice, and notes which of the neighboring sites are connected to the site it is on. At the next time-step this ant places children on each of these connected sites which are not already occupied. The children then proceed to reproduce likewise until the entire cluster is populated. It is easy to see that the time complexity of this method is a linear function of the cluster size.

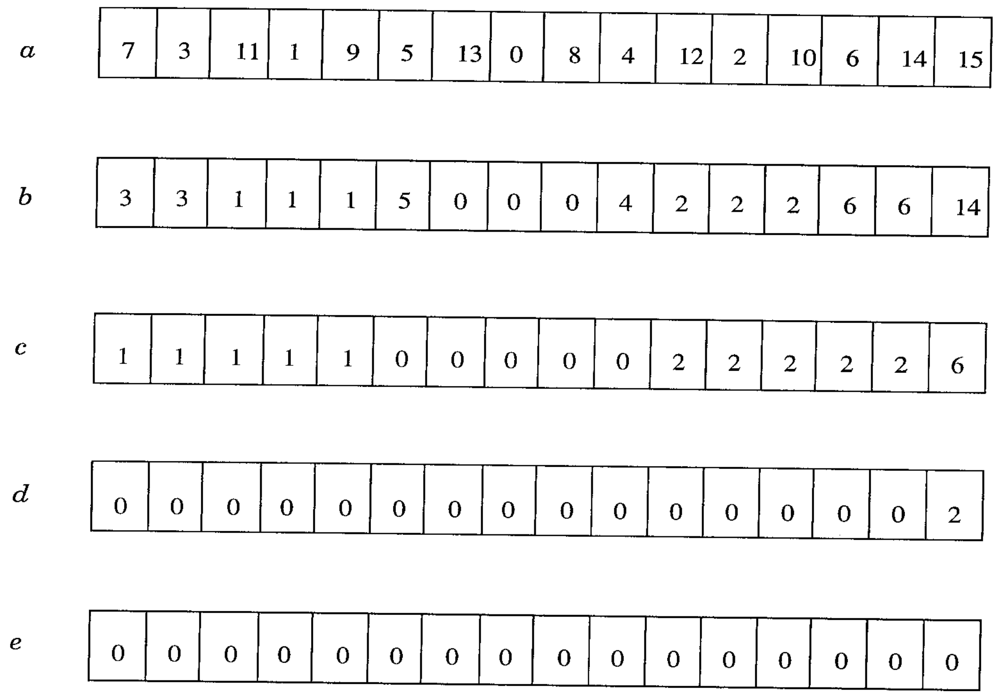


Figure 3.1: The merging of a one-dimensional cluster.

the label numbers survive, and thereby half the number of clusters emerge. After the second iteration (Fig. 3.1c), only one quarter of the label numbers survive. At the same time, only one half of the clusters survive from the first iteration though their lengths may grow twice as long as they were in the previous iteration step. Shown in the illustration of Fig. 3.1, this identification process repeats itself until the growth of one single cluster is completed (Fig. 3.1e). From this evolution, it is not difficult to understand why only log N iteration steps are required to finish the job. Furthermore, the strip can be divided into any number of disconnected clusters. Then only $max\{\log N_i\}$ iteration steps are required to identify those different clusters. Here N_i is the length of i-th cluster.

That the cluster identification ends in a logarithmic time traces back to the architecture defined by initialization process. Instead of assigning consecutive numbers site by site in the array, we initialize the label number according to *3.1* so that the cluster is able to evolve in a hierarchical manner. Fig. 3.2 shows the hierarchical structure embedded in the strip of Fig. 3.1. In the implementation, the sites belonging to the same cluster are identified with a coordinate, pointing to a special point that we call the *center* of the cluster. However, this point doesn't have to be the geometrical center of the cluster. It can be decided by any easy

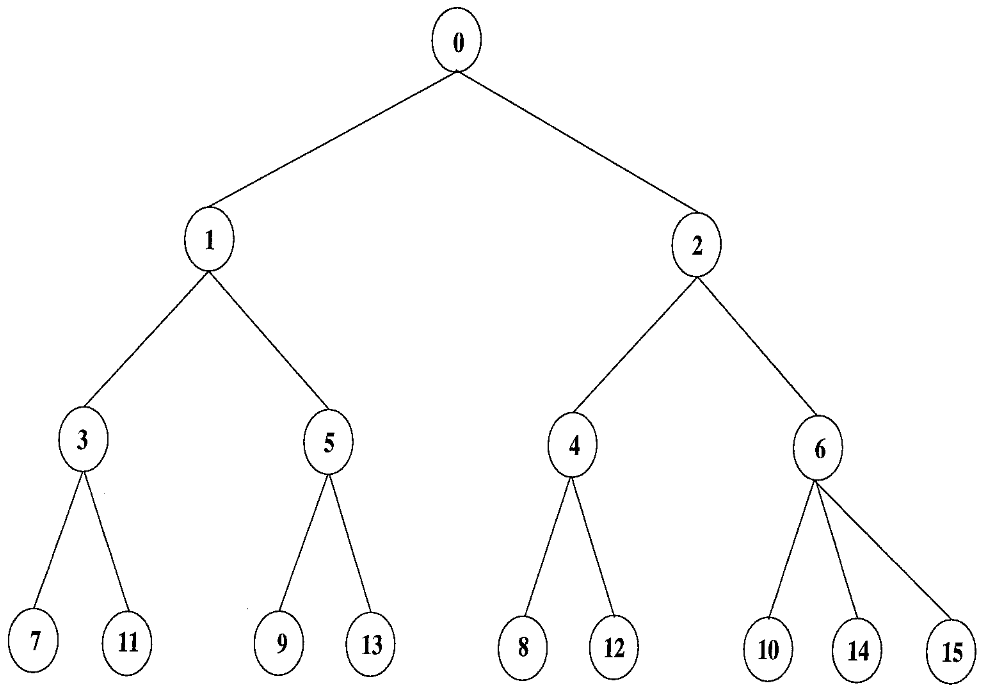


Figure 3.2: The hierarchical tree.
The hierarchical structure hidden in the array of Fig. 3.1.

rule as long as *center* is some point in the cluster. The following is the pseudo code which is applicable to a grid of any dimension. Three parallel structures, *OWN*, *NEIGHBOR*, and *TMP*, are defined for each site on the grid to hold the essential information about the point, e.g., label number and the coordinate of its associated *center*. Since one cluster always connects with more than one other cluster, especially in higher dimensions, *the TMP best satisfying the growth condition* in this problem is that referring to the one having the smallest label number among the connecting clusters.

Pseudo Code

```
1   Initialize OWN with the unique label number on every site of the grid ♠
2   Loop beginning
3       $TMP := OWN$ ♠
4       For all neighbors of each site Do
            $NEIGHBOR := OWN_{neighbor\_i}$ ♠
5           If growth_condition(TMP, NEIGHBOR) = TRUE ♠
              Then
6                 $TMP := NEIGHBOR$ ♠
7           Else do nothing
8       Send the TMP that best satisfies growth_condition in
        each cluster to $OWN_{center}$ ♣
9       For all sites Do
            $OWN := OWN_{center}$ ♠♣
10      Continue this loop if any growth occurs in the previous round
11  End of Loop
```

To exhibit the parallelism in the code, two superscripts, $\spadesuit$ and $\clubsuit$, are affixed to the instructions which are executed in parallel for all sites and for all clusters, respectively. When every site updates its label number, obsolete *center*'s are discarded at the same time. Only the *center* bearing the smallest label number survives after two or more clusters merge. Two superscripts appearing together at the end of line 9 depicts this occurrence.

This pseudo code illuminates the fact that the whole algorithm is essentially the superposition of two parts: first, a special arrangement in the label number initialization; second, the nonlocal cluster updating which occurs by applying parallel routines to the updating of *center* points. It may not be so obvious to perceive the superiority of this algorithm having seen only the example in one dimension. In the next section, the extension to higher dimensions will be presented and the advantages of the hierarchical algorithm will thereby be revealed.

3.3 The Algorithm in Higher Dimensions

As elucidated by the pseudo code in the previous section, the intrinsic work required for the extension to higher dimensions is the enlargement of the hierarchical structure in the label number initialization. That is, only modification of the first part of the algorithm (line *1*) is required. The adaptation of the second part occurs automatically as the configuration (or *shape* in the terminology of the Connection Machine) of the grid is redeclared for higher dimensions in the code header. In the beginning, we show how this enlargement is accomplished in two dimensions and then introduce the general method in arbitrary dimensions.

The basic idea in extending this structure to two dimensions is to store the hierarchical ordering of the initial label numbers in both directions at the same time, horizontally and vertically, from any site on the grid. Once the structure satisfying this criterion is constructed, the clusters will be connected in a logarithmic time, in both horizontal and vertical directions simultaneously (or in all directions on the two-dimensional grid, in fact). The clusters merge in this manner, no matter where they are located on the grid, since all the clusters are updated in a parallel fashion according to the second part of the algorithm.

Now we introduce our method of weaving this structure on the two-dimensional grid. Suppose (x,y) is the coordinate of a certain point on the grid. Making use of

3.1, one can compute $label\_value\_x$ taking $label\_value\_x = label\_value$ with $coord = x$; $label\_value\_y$ is similarly computed simultaneously. Therefore the respective label numbers projected onto the vertical and horizontal axes are derived, and the hierarchical structures expected to reside in each direction are simply encoded in these numbers. Now we still need an appropriate melding of these two numbers to define the actual label number on this site, in such a way that the hierarchical structure is entirely preserved in both directions. There are several ways to combine these two projected numbers and still keep the combined numbers distinct from one another. One natural way is to take $label\_value = p^{label\_value\_x} q^{label\_value\_y}$, where p and q are any two different prime numbers. However, the drawback of this combination is that it may take too much memory when the grid size is large: This combined label number increases exponentially with the size of the projected label numbers.

In fact, there is a linear combination of the label numbers which satisfies the aforementioned criteria, and uses only the same order of the memory storage as that of assigning label numbers consecutively site by site.

Theorem 3.1.

Let

$$label\_value = p * label\_value\_x + q * label\_value\_y, \tag{3.2}$$

then the label numbers on a grid of size $N = N_x * N_y$ are distinct.

Here p and q are two relatively prime positive integers,

$$(p, q) = 1,$$

$$p > 1,$$

$$q > N_x - 1,$$

and N_x and N_y are, respectively, the horizontal and vertical widths of the grid.

Proof.

Suppose z_1 and z_2 are two label numbers that happen to have the same value, yet originate from different combinations, i.e.,

$$z_1 = p * label\_value\_x_1 + q * label\_value\_y_1$$

$$z_2 = p * label\_value\_x_2 + q * label\_value\_y_2$$

and

$$label\_value\_x_1 \neq label\_value\_x_2$$

$$label\_value\_y_1 \neq label\_value\_y_2.$$

Then we have

$$0 = p * (label\_value\_x_1 - label\_value\_x_2) + q * (label\_value\_y_1 - label\_value\_y_2),$$

i.e.,

$$0 = (label\_value\_x_1 - label\_value\_x_2)\ (mod\ q)$$

$$0 = (label\_value\_y_1 - label\_value\_y_2)\ (mod\ p),$$

for $(p,q) = 1$.

We know that this is impossible, since

$$q > N_x - 1 \geq (label\_value\_x_1 - label\_value\_x_2) \neq 0.$$

Therefore, on the grid of $N_x * N_y$, no two label numbers can be identical by this assignment. $\square$

Furthermore, suppose we define $label\_value\_x_i$ the projected label numbers in every dimension, $i = 1, 2, \ldots, n$, from *3.1.* The generalization of *Theorem 3.1* to arbitrary dimensions is as follows:

Theorem 3.2.

Let

$$label\_value = \sum_{i=1}^{n} p_i * label\_value\_x_i, \tag{3.3}$$

then the label numbers on the grid of size N, equal to $\prod_{i=1}^{n} N_i$, must be distinct.

Here

$$(\forall n \geq i > 1) \qquad\qquad p_i > \sum_{j=1}^{j<i} p_j(N_j - 1),$$

and N_i is the width of the grid in the i-th dimension.

The proof is similar to that of *Theorem 3.1*, except more trivial. One may find that the linear combinations above do not reduce to those in *Theorem 3.1* when $n = 2$, for the restriction that p_1 and p_2 be relatively prime has been lifted. However, both methods are legitimate. It was pointed out earlier that the combination rule is not unique.

Notice that there is no theoretical upper bound for the choice of p_1 (or p in *Theorem 3.1*). However, in the implementation, it is usually chosen to be 1, using the assignment of *Theorem 3.2*, or chosen to be 2 using that in *Theorem 3.1*. This way all the label numbers are kept as small as possible. If, for instance, all shuffled label numbers are assigned to a two-dimensional grid by these methods, one can see that the hierarchical structure is preserved on any size patch of any shape, wherever it is located on the grid. Therefore, for any cluster configuration on a grid of arbitrary dimension, this assignment guarantees the asymptotic running time is $max\{\log N_{ij}\}$ (in practice, greater than this time). Here N_{ij} denotes the

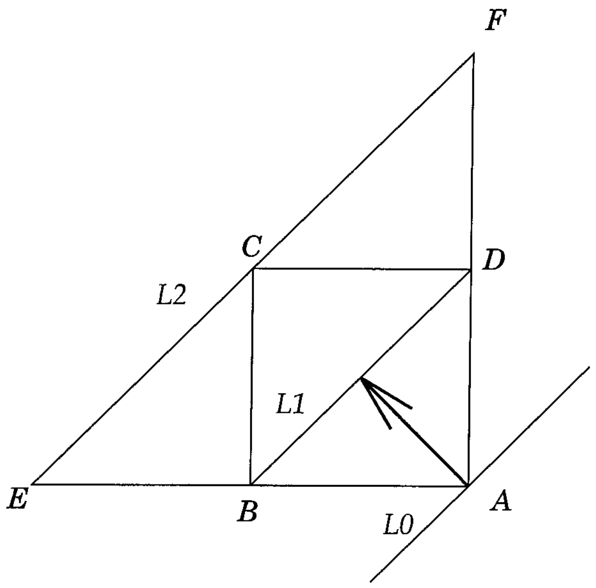


Figure 3.3: The growth sequence in 2-d.

width in the j-direction of the i-th cluster.

In the growth sequence, the clusters actually evolve in diamond shapes or, more generally speaking, by moving diagonally. Due to the identical hierarchical structure along each axis of the grid, the clusters are updated in all axial directions with the equal 'velocity.' Fig. 3.3 demonstrates this process in two dimensions. The heavy vector in this figure indicates the 'superposed' direction of cluster updating from L_0 to L_1, while $\overline{AB}$ and $\overline{AD}$ represent, respectively, the horizontal and vertical directions of the grid. Imagine that the rectangle $ABCD$ is an arbitrary subset of the grid (it could be the whole domain, too). The updating from point A to point C implies a parallel shift of the updating line from L_0

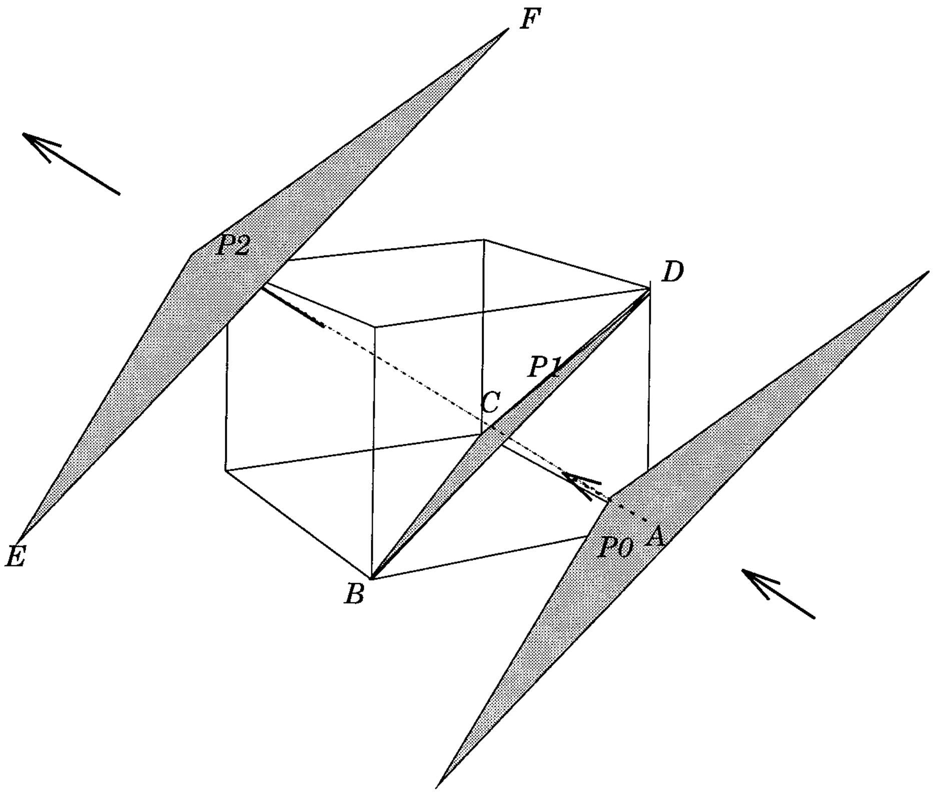


Figure 3.4: The growth sequence in 3-d.

to L_2. As the diagonal updating line crosses over this rectangular domain along the heavy arrow direction, the updating progression along each axial direction has proceeded two lattice sites, i.e., $\overline{AE} = 2\overline{AB}$ and $\overline{AF} = 2\overline{AD}$. For a d-dimensional cube, the updating in each axial direction will proceed d lattice sites when the $(d-1)$-dimensional diagonal hyperplane crosses the whole cube (Fig. 3.4 shows a three-dimensional example).

The actual number of iteration steps required to finish the labeling is very likely to be much smaller than the upper bound stated in the following theorem below, we will not address such subtleties here, postponing the further discussions until the next chapter.

Theorem 3.3.

The time needed to complete the cluster labeling is at most log N, and is the same as the height of the hierarchical structure constructed in *Theorem 3.2* or *Theorem 3.3*. Here N is the number of sites in the grid.

Proof.

The theorem is nearly self-evident. Since the hierarchical structure, or tree, has been defined in the problem configuration, the connected components simply emerge as the tree is described (see Fig. 3.5) from the bottom, and terminate at a particular node at a certain height. This height is the time step at which the cluster stops growing. Since the maximum height of the tree is log N, the temporal upper bound for the algorithm is log N. $\square$

As we see from *Theorem 3.3*, building a suitable hierarchical tree for the problem makes the proof trivial, very much unlike the cumbersome proofs required

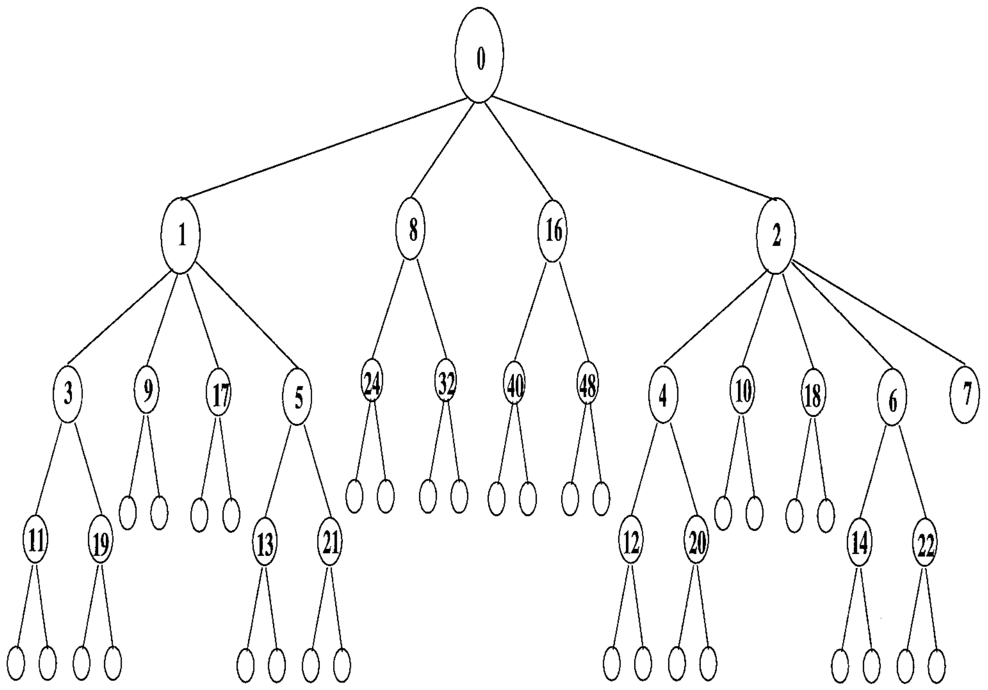


Figure 3.5: The composite hierarchical tree.
The *composite* hierarchical tree embedded on a 2-d grid; only partial label numbers and sites (or nodes in the tree) are shown.

in the graph algorithms described in the last chapter. This is a triumph of *shuffling* techniques, by which the corresponding hierarchical tree is established and then an efficient algorithm is derived. Moreover, through shuffling techniques, this method of cluster labeling highlights a fundamental difference from the previously used techniques [6], [19], [70] for parallel cluster labeling. Some implementation results of the algorithm are described in [103]. It should be pointed out that Rossi and Tecchiolli [90] proposed two options of label number shuffling similar to ours for 1-d clusters (their preprint was shown to the author after this work had been substantially completed). However, they are not the optimal choices in one dimension. In particular, Rossi and Tecchiolli did not discern the most important implication of label number shuffling –hierarchical structures on the problem domain. As a result, the general and systematic shuffling scheme was not discovered in their work.

Quite often, the shuffling or reshuffling of the original problem configuration should be viewed as a different *"representation"* of the original configuration, such that we obtain a better *"perception"* of the problem through which one may be able to seek a better or even optimal solution. Very similar to the codewords in communication theory (see the Appendix), shuffling is a method of *encoding* that conveys the essential information about a problem in an optimal way. This property is particularly useful when the problem has a highly irregular dynamic nature. More elaboration and discussion of these points will be presented in the following chapters.

3.4 The Implementation Results

We have implemented the algorithm on NPAC's (Northeast Parallel Architectures Center at Syracuse University) Connection Machines, CM-2 and CM-5. The Connection Machine 2 (CM-2) is a fine grain SIMD machine, which is better suited for the implementation of the hierarchical cluster labeling algorithm. The average times (typically over 500 sample points) of each iteration (i.e., 1 step of time complexity) taken in simulating the Ising system at criticality are plotted in Figs. 3.6 and 3.8 (16K processors). For the lattice size of interest (1024×1024), our algorithm gives an averaged time per iteration per site of 0.268 microseconds on a 16K CM-2. This measurement is already about 10 times faster than the best

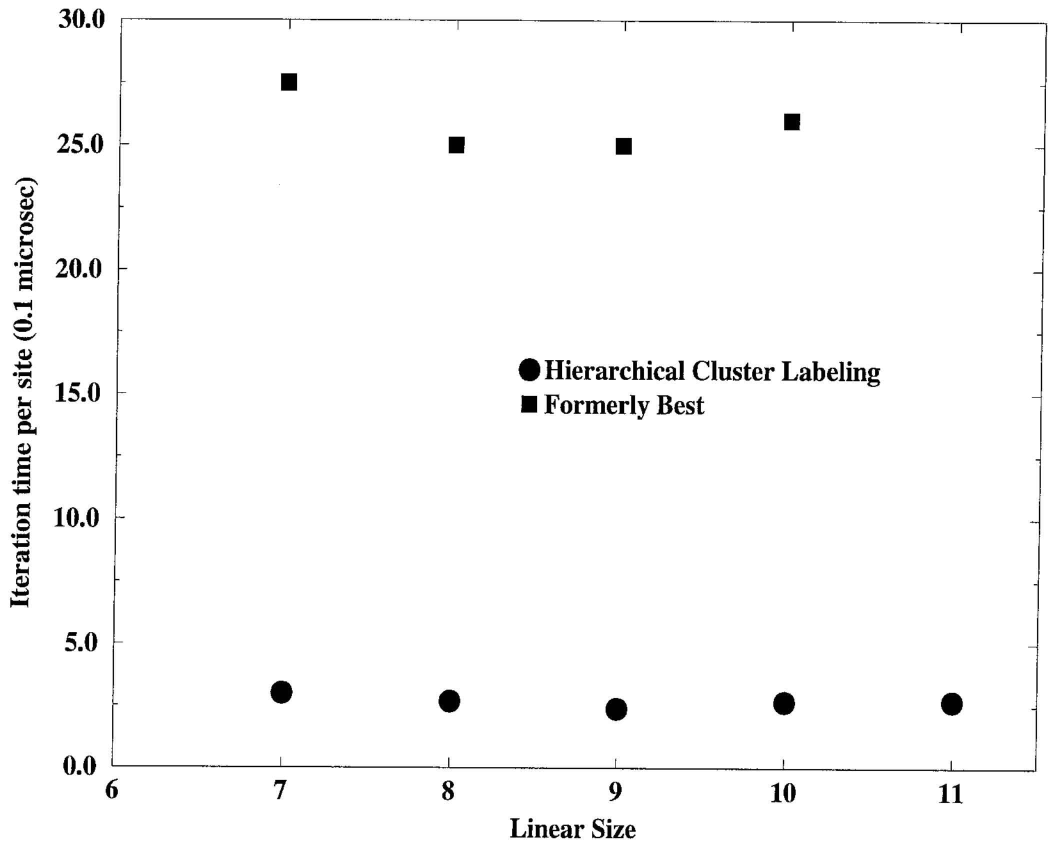


Figure 3.6: The average iteration times (2-d) on CM2.
The numbers on the abscissa are the exponents of the linear size (base = 2).

results previously achieved (Apostolakis *et al.*'s [6] and Brower *et al.*'s [19]), which were 2.6 microseconds for each iteration. Furthermore, owing to the hierarchical scheme employed in the algorithm described above (i.e. nearly in an optimal way), the numbers of iterations to finish the labeling at criticality are always less than the lowest previously reported (see Fig. 3.7). Notice that the numbers on the abscissa of all plots in this section are the exponents (base = 2) of the linear size or of the total site number (more data not at criticality are recorded in Figs. 3.13-3.17).

Besides demonstrating a success in numerical improvement, the hierarchical cluster labeling algorithm also allows a systematic analysis on the implementation, for instance, the estimation of iteration steps to finish the labeling for a cluster of any kind of complicated shape, and the further optimization of the algorithm by reshuffling the label numbers. The analysis is essentially achieved by making use of the hierarchical structure (or recursive decomposition) furnished in the algorithm. In systems of higher dimensions, the algorithm is expected to give even better performance and excel over other algorithms for the same problem (e.g. the performances on 3-d Ising systems at criticality are shown in Fig. 3.8 and Fig. 3.9; those not at criticality are in Figs. 3.18-3.20). In a sense, the hierarchical

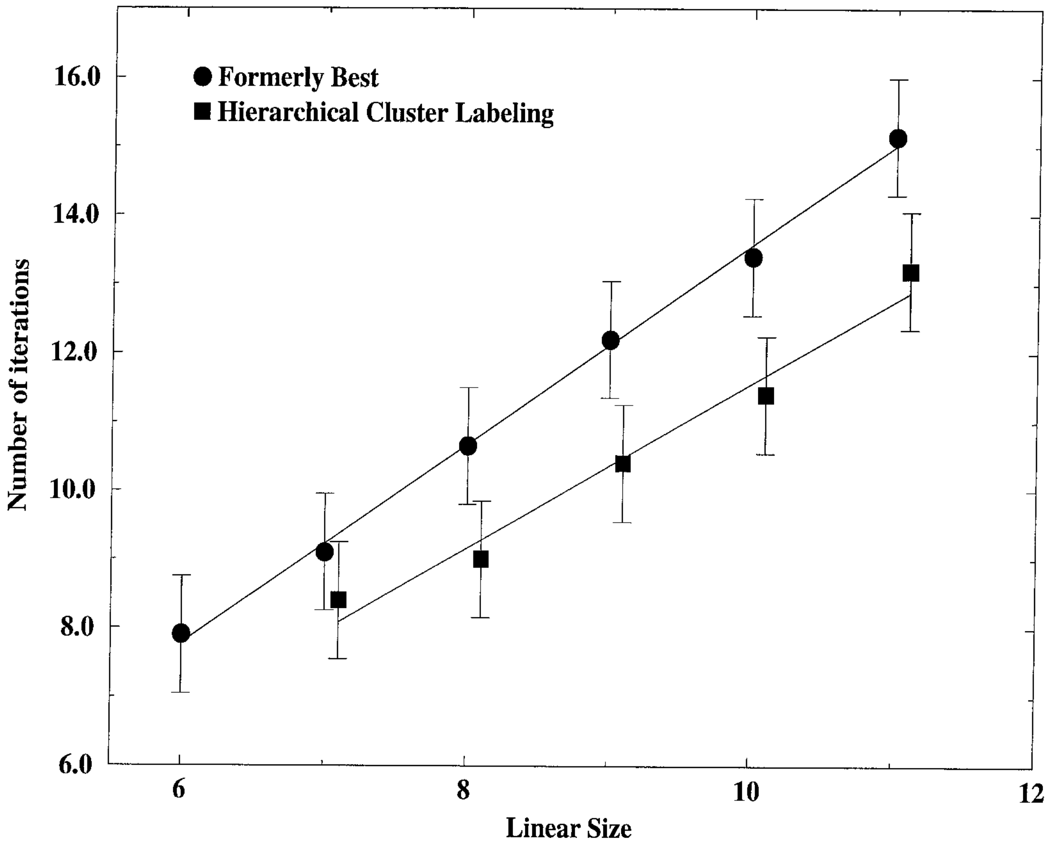


Figure 3.7: The average number of iterations at criticality.

cluster labeling algorithm can be considered as a generalized *scan* scheme [16] applicable to a domain of arbitrary geometry and of any dimension, yet not using any built-in routine provided by the machine to speed up the implementation.

When the same code (i.e. a SIMD version) is implemented on the CM-5, which has only 32 processors available to us and is practically a MIMD machine, the algorithm is unable to exhibit a speedup closely comparable with that on the CM-2 (see Fig. 3.10). However, after an appropriate adaptation which makes the code more "congruent" with the MIMD architecture of the CM-5, the algorithm implemented on the CM-5 is able to perform as well as on the CM-2 (as shown in Fig. 3.10). The basic trick to do this adaptation is that, instead of each site on the grid (i.e. each virtual processor) executing the same command simultaneously as on a SIMD machine, only the boundary sites keep active in the process of checking neighbors' label numbers and exchanging information with the associated center; all the internal sites, whose label numbers remain the same during the above two procedures, stay idle until they need to update their label number. This simple trick (a similar but independent work is reported in [39]), also incorporating the use of a very effective communication primitive, *active message passing*, provided on the CM-5, has greatly improved the performance of the algorithm on

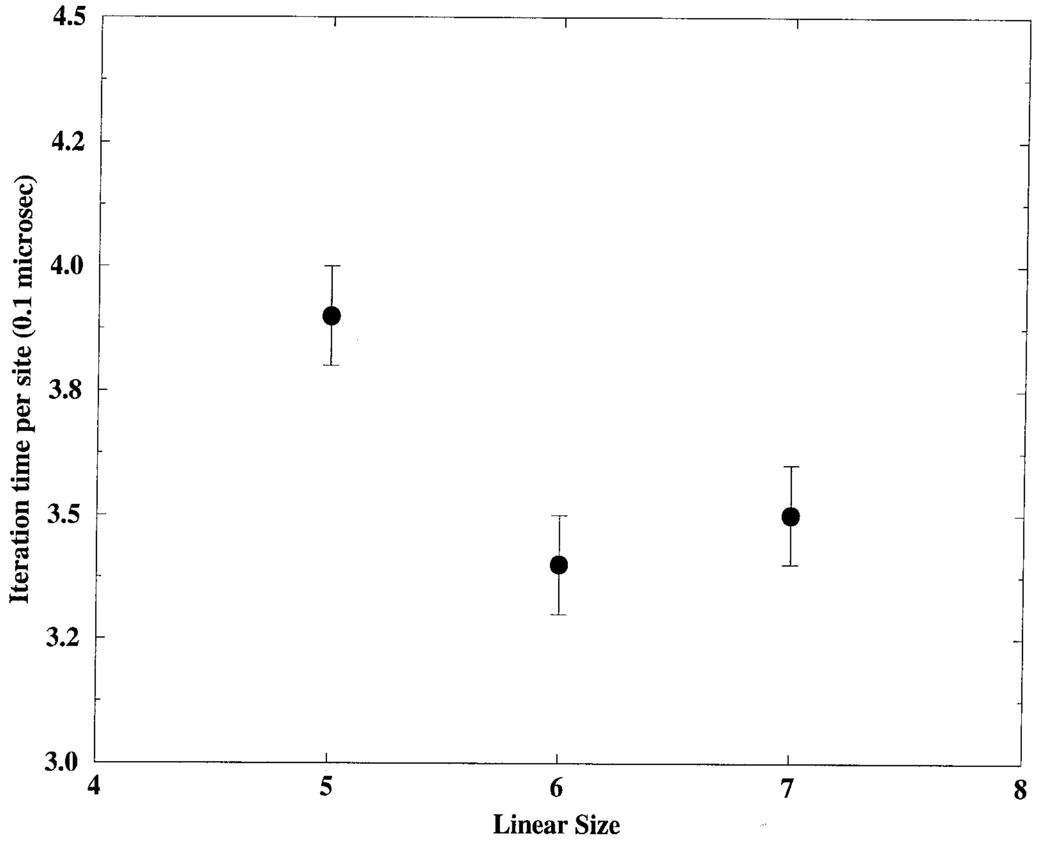


Figure 3.8: The average iteration times (3-d) on CM2.

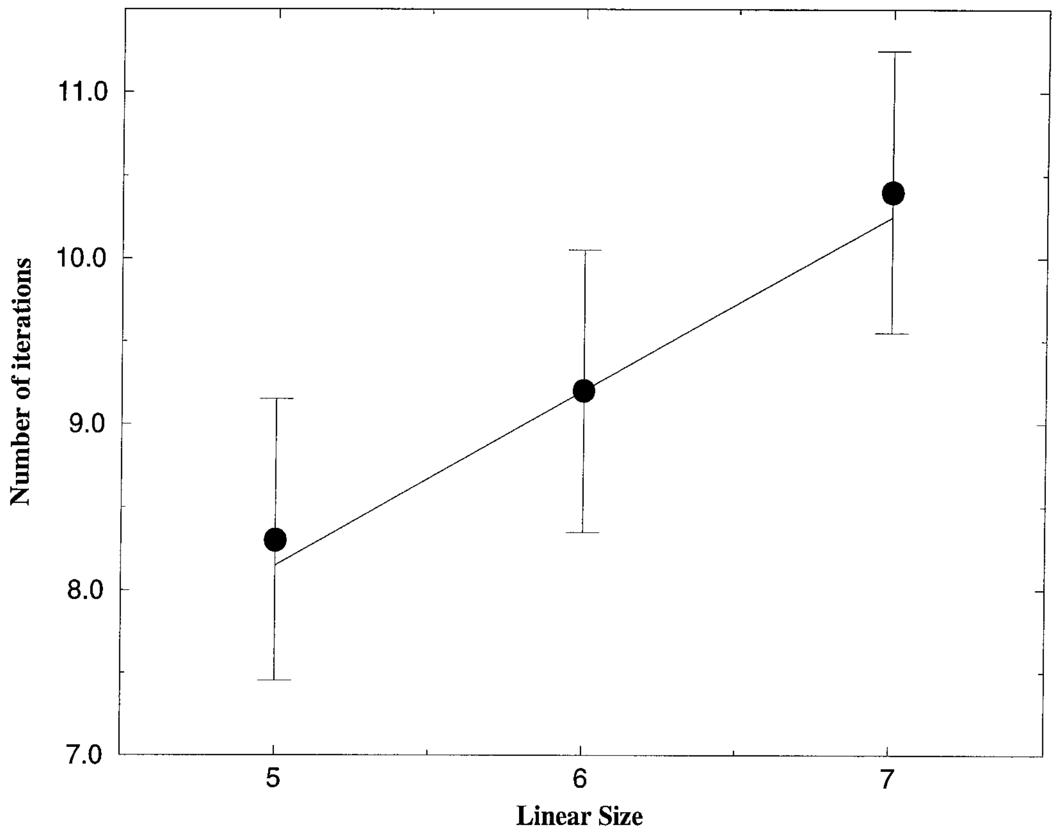


Figure 3.9: The average number of iterations at criticality (3-d).

this machine. The absolute average times taken in each iteration of the adapted algorithm on the CM-5 (32 nodes) are still longer than those on the CM-2 (16K nodes). However, when considering the difference between the peak performance between these two machines (the peak performance of 16K CM-2 is 500 Mflops and that of 32 node CM-5 is only 144 Mflops without vector units), one can safely say that the algorithm, originally designed for SIMD machines, works equally well on MIMD machines. In fact, if the ratio of the peak performance between these two machines is taken into account, the performance of the algorithm on the CM-5 is about 2 times as fast as that on the CM-2. The ratios of speedup (on the CM-5) using this adaptation for different linear sizes of 2-d grid are shown in the first array of the following table,,and the ratios of speedup using the adaptation (on the CM-5) over the original performance (on the CM-2), after scaling with the ratio of peak performance between these two machines with quite different architectures, are in the second array.

Speedup	*Linear Size*			
	128	256	512	1024
CM5(MIMD) vs. CM5(SIMD)	3.27	3.40	3.06	3.77
CM5(MIMD) vs. CM2(SIMD)	1.79	1.99	1.70	2.27

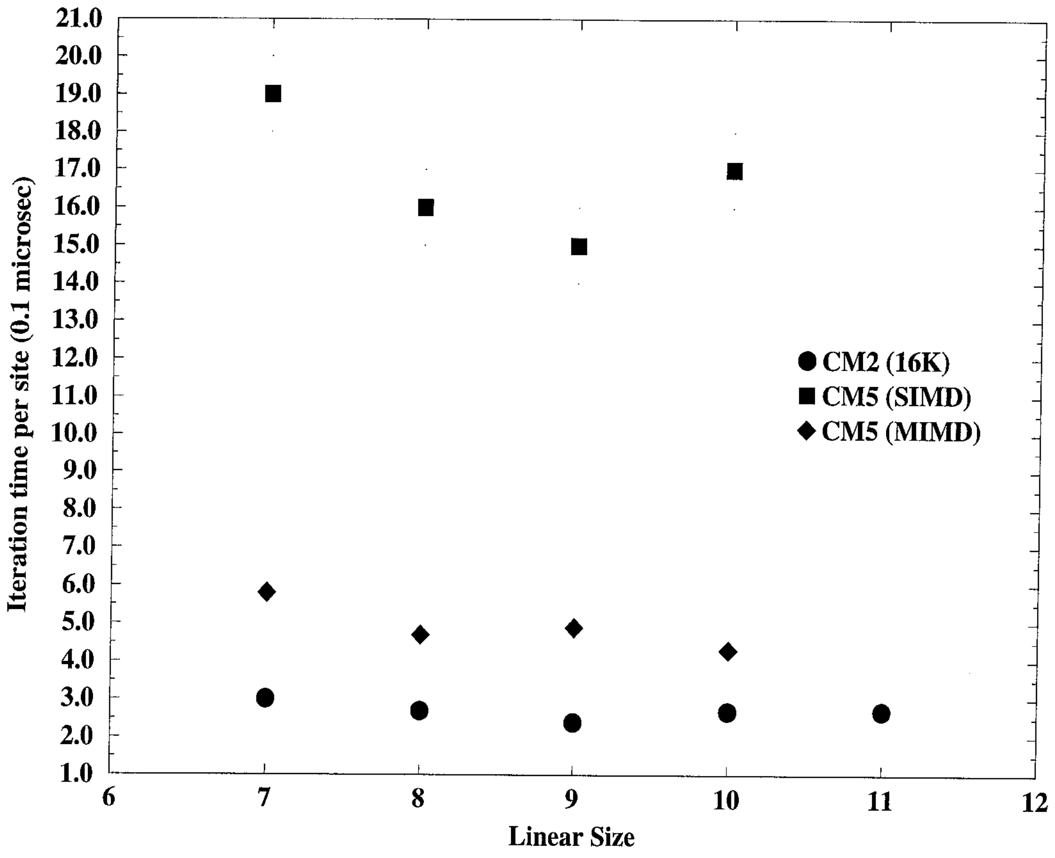


Figure 3.10: The average iteration times (2-d) on CM5.

Further performance improvement can be achieved in the CM-5 program by

1. overlapping communication and computation;

2. further reduction in the overhead of processing the active sites and centers.

The basic idea of the MIMD implementation is to take advantage of the large computing power provided by each physical processor (in CM-5, it is a sparc processor) and meanwhile to maximize the usage of the information collected on the current problem domain (or cluster configuration in this particular problem). As the algorithm is implemented on MIMD machines, the center of a site should be separated from the sites. The site uses a pointer to maintain a link to its center. At all times, the site will either point to the center of its cluster or to a representative of its cluster. Here, a representative is designated as a "local center" in a node, for the sites in the same cluster and confined in this node. The point of the representatives is to reduce the occurrence of node-to-node communications that are much more costly. The lifetime of a site passes two stages. At the first stage, the site is a boundary site. It is responsible for exploring the neighboring cluster and reporting the finding to the center or the representative. The site is in the second stage when it serves as an internal site. It is responsible for maintaining the center information during the updating phase, i.e., not attended until the whole cluster needs updating.

The center of a cluster passes through three stages over its lifetime. Stage one of a center is to serve as a center of a cluster. At this stage, it decides the future of the cluster, specifically which neighboring cluster to join. It also serves the center information to the local sites that belong to the cluster. The second stage is the representative stage. At this state, the representative serves as a cache of the center information for an off node center. The representative also serves as a buffer and an arbitrator for the information extracted by the local sites about the neighboring clusters. In other words, each individual cluster is identified with a *local* representative rather than the *remote* center. After all the local sites complete their exploration, the representative then presents the final selected neighboring information to the center for further processing. The final stage of a center is to be removed from the processing list.

More benchmarking data (on a CM-2 with 16K processors) of the algorithm applied to higher dimensional Ising systems are shown in Fig. 3.11 and Fig. 3.12 (this timing is very insensitive to the variance of the coupling constant). It is no surprise that the iteration time per site increases with dimension as the total number of sites is fixed (Fig. 3.11). The major reason contributing to this increase is that the higher the dimension is, the more directions of neighbor-checking are required and thus more communications are involved. It is important to notice that, even in the highest dimensions (19-d and 20-d) allowable on the CM-2, the iteration times per site are only 4.8 microseconds. This time is still shorter than that of 5.8 microseconds obtained by Brower *et al.* on the 2-d Ising system of the same number of total sites, and is less than two times the best result (2.6 microseconds) obtained on a 2-d (1024×1024) grid by Apostolakis *et al.*

In this chapter, we have presented a hierarchical method to tackle the problem of cluster labeling on parallel machines. In addition to successful speedup in the real implementation, the hierarchical cluster labeling algorithm is a very competitive algorithm in its own right. Compared with the methods formerly known, this algorithm is much easier to implement and is more memory efficient. This is because the primary information (or recipe of the algorithm) is encoded as a hierarchical structure that has been built via a shuffling of the label numbers, from the very beginning of the label number initialization. Through this hierarchical structure, the problem domain is recursively decomposed and clusters can thus be recursively labeled in a nearly optimal way. This algorithm demonstrates a clear-cut and uniform method to deal with arbitrarily irregular cluster configurations,

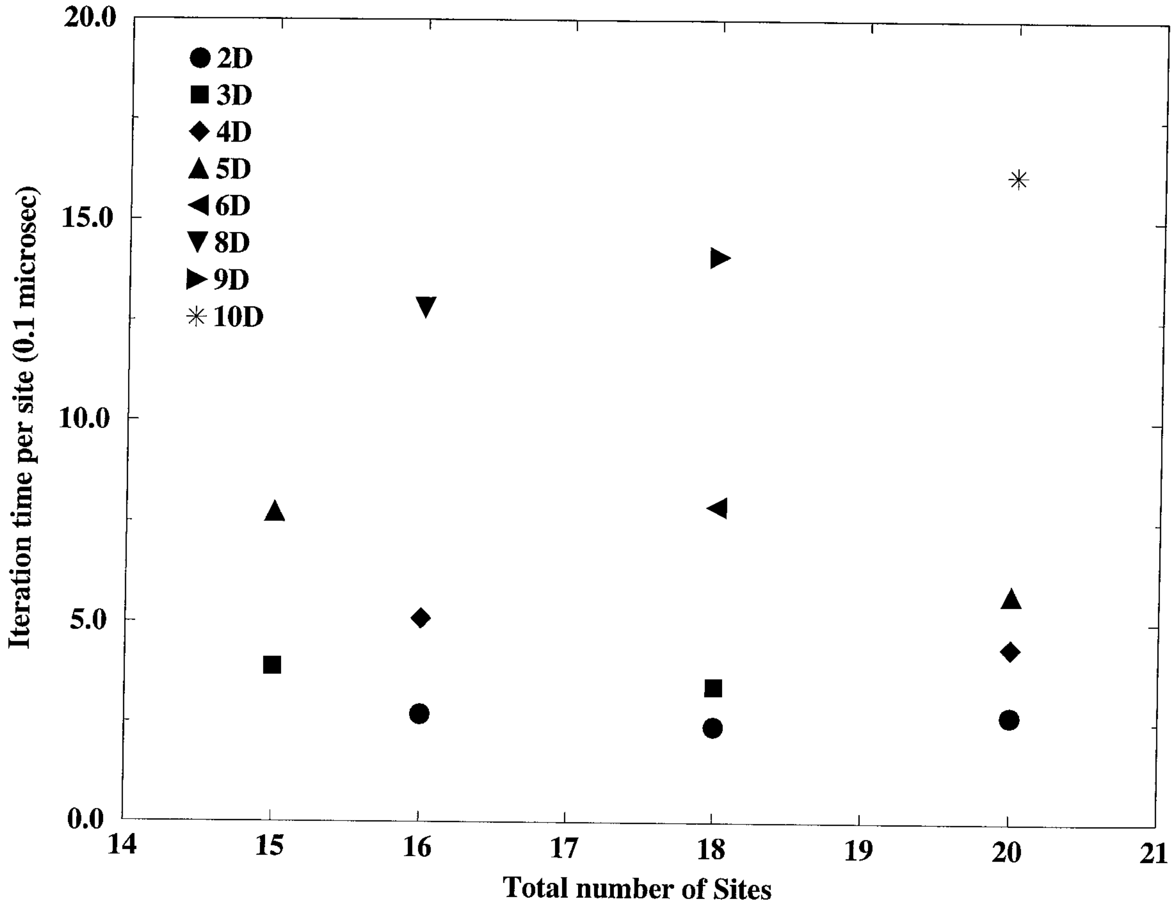


Figure 3.11: The average iteration times (on CM2) in higher dimensions. Only the exponents (base = 2) of the total number of sites are shown on the abscissa.

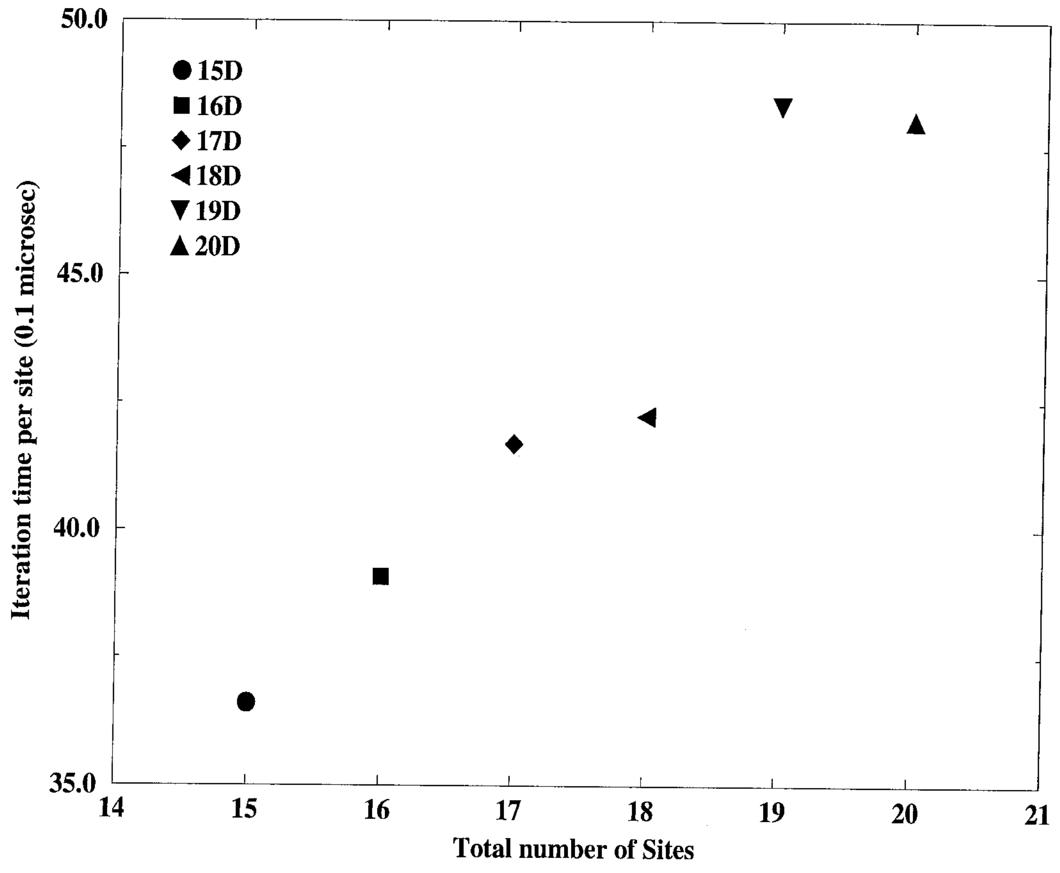


Figure 3.12: The average iteration times (on CM2) in very high dimensions.

without the aid of any *á priori* knowledge or any generic testing. Allowing close estimation of the computational complexity during the course of the algorithm is thereby one of natural merits possessed by this method. In particular, the hierarchical cluster labeling algorithm should be considered as a generalized *scan* scheme applicable to a domain of arbitrary geometry and of any dimension, by simply replacing the neighbor-checking by the operation required. Furthermore, owing to the inherent hierarchical structure of the algorithm, the adaptation of the algorithm to machines of various architectures becomes easier and admits systematic analysis, especially when nowadays parallel computers often have a certain kind or a certain degree of hierarchical organization. Such a useful technique, the combination of shuffling and divide-and-conquer, is not fully exploited in the above-mentioned algorithms addressing the same problem. We will discuss these features in more detail in the next chapter. Yet, this technique has been well utilized in many other efficient scientific algorithms. FFT is the most famous one among them and it will be elaborated in Chapter 5.

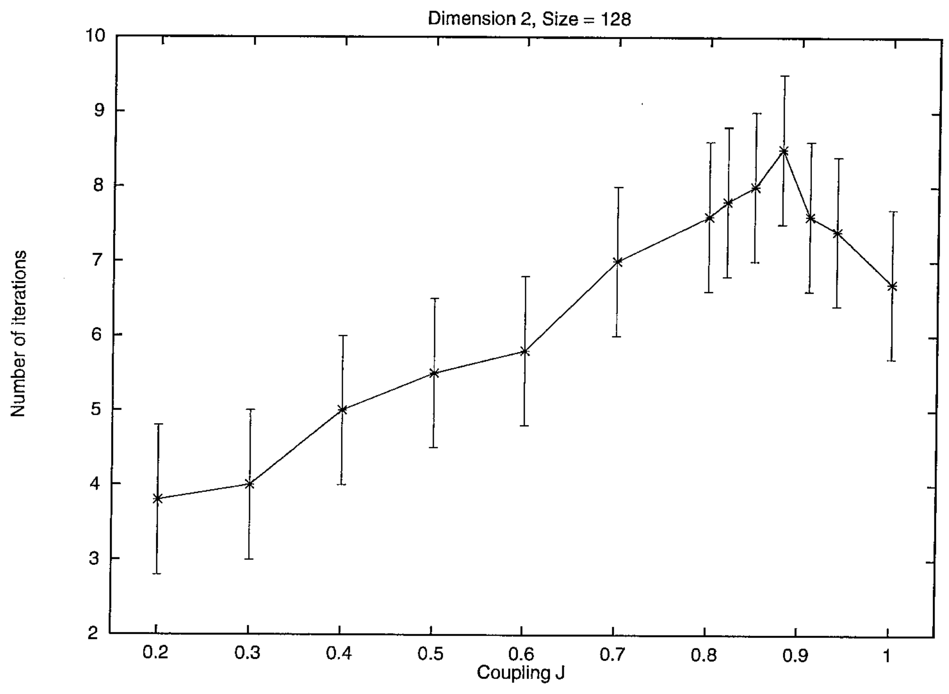


Figure 3.13: The iteration numbers as D=2, S=128.

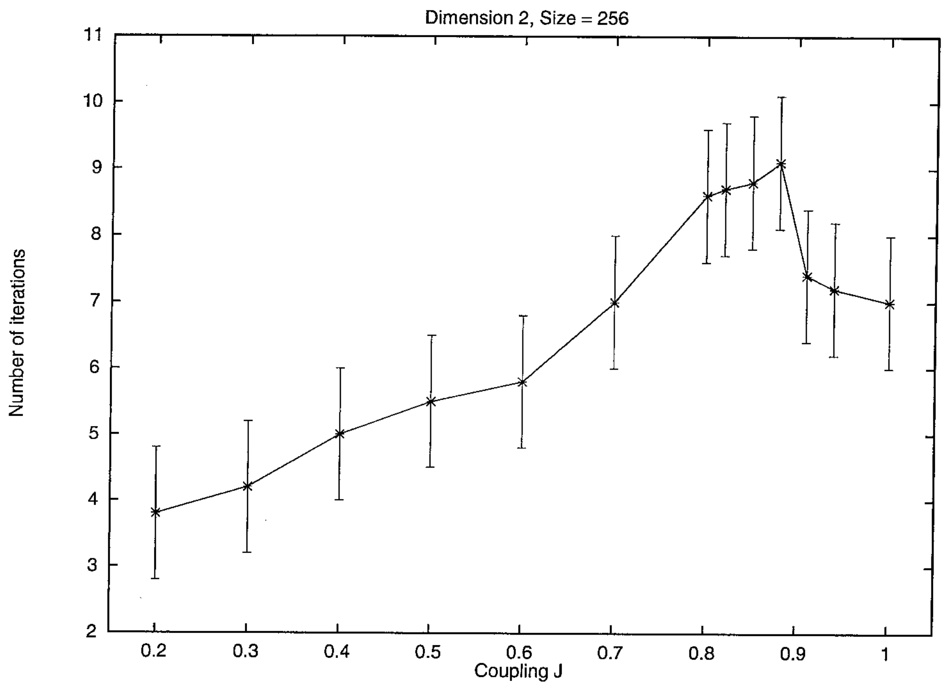


Figure 3.14: The iteration numbers as D=2, S=256.

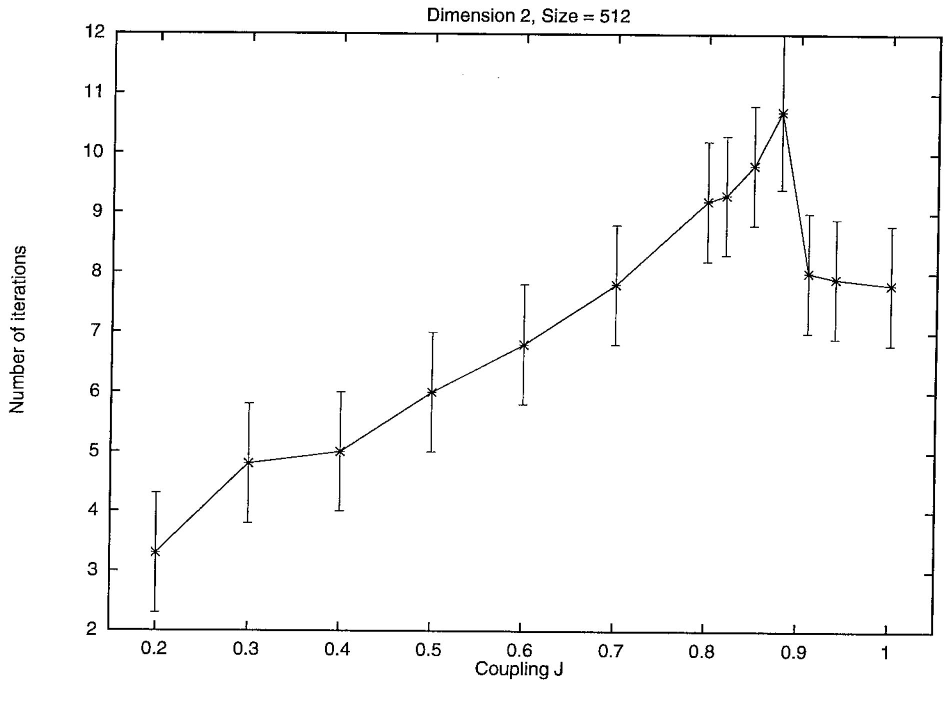


Figure 3.15: The iteration numbers as D=2, S=512.

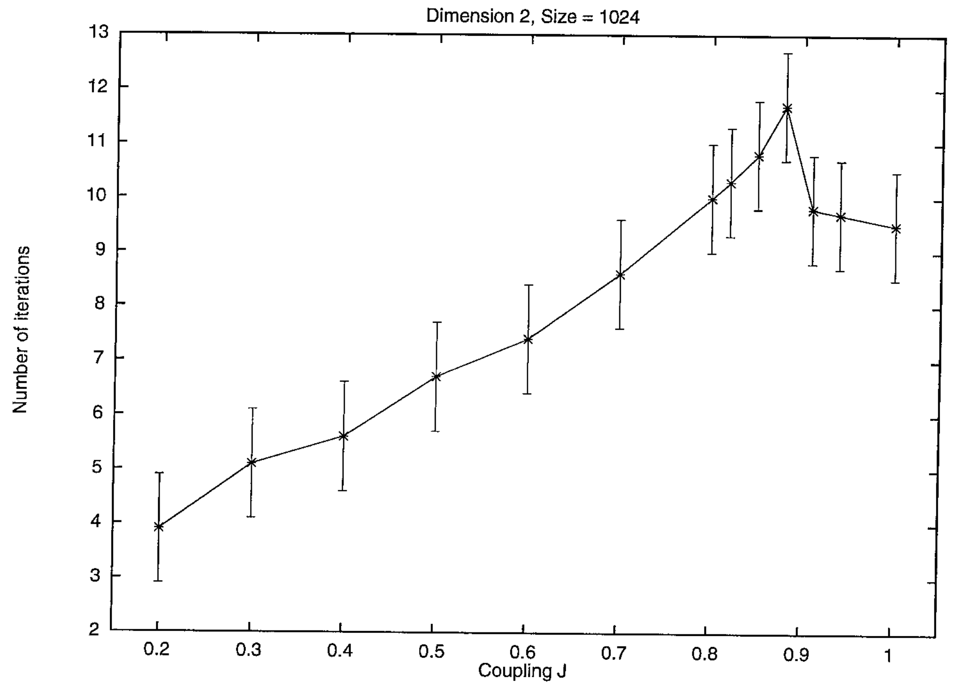


Figure 3.16: The iteration numbers as D=2, S=1024.

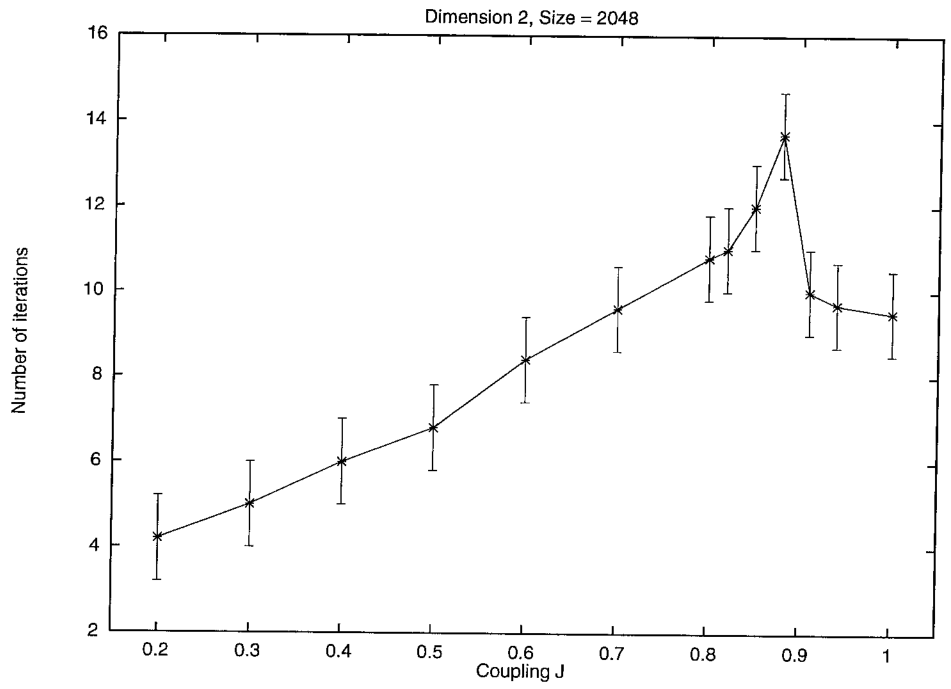


Figure 3.17: The iteration numbers as D=2, S=2048.

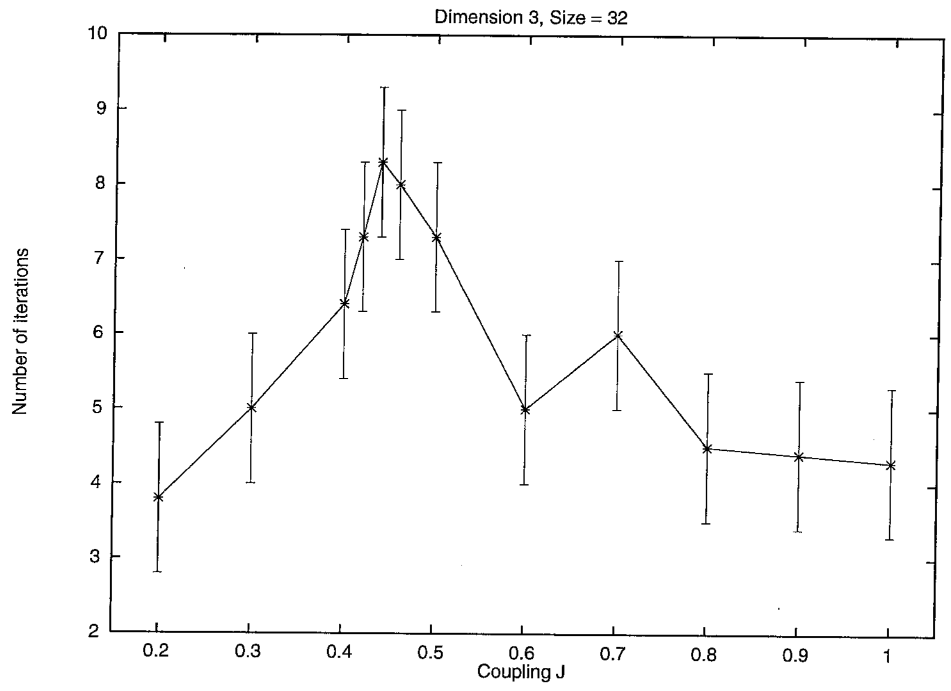


Figure 3.18: The iteration numbers as D=3, S=32.

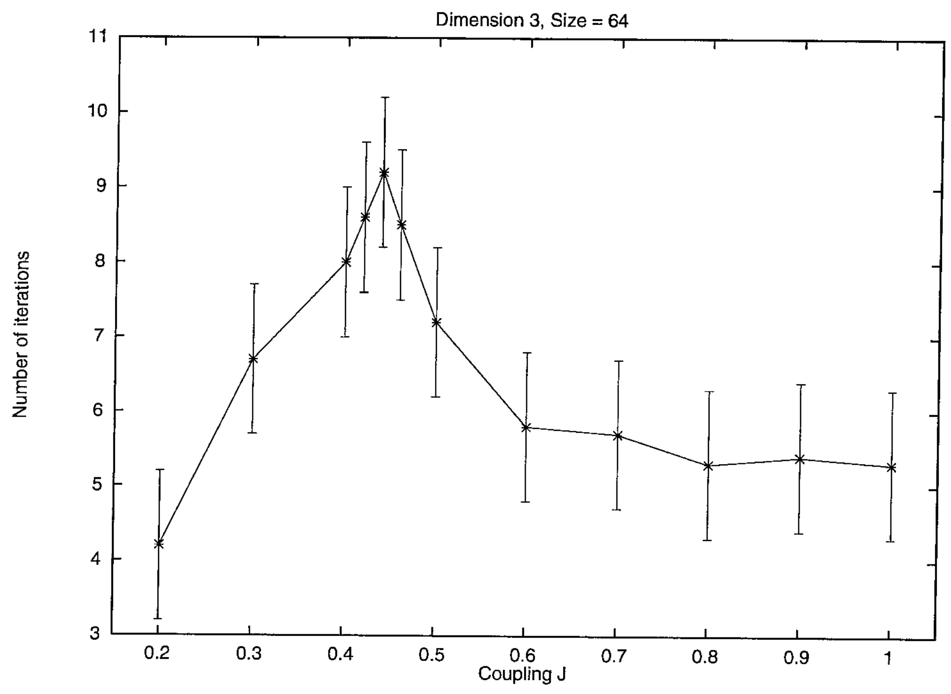


Figure 3.19: The iteration numbers as D=3, S=64.

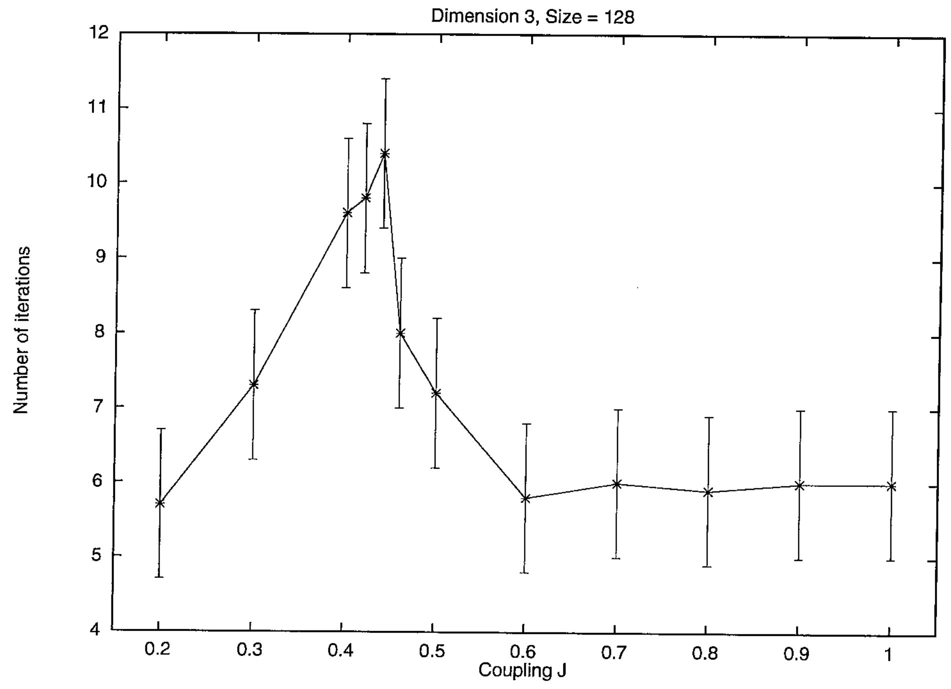


Figure 3.20: The iteration numbers as D=3, S=128.

Chapter 4

More about the Hierarchical Cluster Labeling Algorithm

4.1 The Recursive Relation

In the first part of this section, we would like to discuss the recursive relationship hidden in the shuffling specified by (3.1). This relation will be indispensable in later discussions.

As introduced in the last chapter, the hierarchical tree can essentially be constructed by an appropriate shuffling. Fig. 4.1 displays the feature of *self-similarity* contained in the shuffling obtained using (3.1). The figure shows the label numbers assigned by the formula to an array of 32 sites. One can clearly see that every shuffled label number, except the one at the end (the 32nd number in this case), is either greater than its two nearest neighbors or less than both. This feature is preserved only if the *greater* numbers are extracted and moved to the next *level*. The same is true if, alternatively, the *lesser* numbers are extracted and moved to the next *level*. On the right side of Fig. 4.1, the bold face numbers are all less than their nearest neighbors at each level of the hierarchy; on the left side, the bold face numbers are all greater than their nearest neighbors at each level.

To make these ideas precise, some definitions are needed.

Definition 4.1.

For a given array of length 2^N, the site coordinates range from 0 through $2^N - 1$. Let $(\alpha_N, \alpha_{N-1}, \alpha_{N-2}, \ldots, \alpha_1, \alpha_0)$ denote the binary representation of an arbitrary integer between 1 and 2^N. The *1-d λ-shuffling*, λ_1, is a one-to-one mapping from the coordinates to label numbers which are all integers:

$$\lambda_1 : \quad (a_N, a_{N-1}, a_{N-2}, \ldots, a_1, a_0) \rightarrow (a_N, a_0, a_1, \ldots, a_{N-2}, a_{N-1}). \qquad (4.1)$$

			15	**15**	1	15				
				7	2	**7**	7			
		23	**23**	**23**	3	23				
				3	4	**3**	**3**	3		
			19	**19**	5	19				
				11	6	**11**	11			
	27	**27**	**27**	**27**	7	27				
				1	8	**1**	**1**	**1**	1	
			17	**17**	9	17				
				9	10	**9**	9			
		25	**25**	**25**	11	25				
				5	12	**5**	**5**	5		
			21	**21**	13	21				
				13	14	**13**	13			
29	**29**	**29**	**29**	**29**	15	29				
				0	16	**0**	**0**	**0**	**0**	0
			16	**16**	17	16				
				8	18	**8**	8			
		24	**24**	**24**	19	24				
				4	20	**4**	**4**	4		
			20	**20**	21	20				
				12	22	**12**	12			
	28	**28**	**28**	**28**	23	28				
				2	24	**2**	**2**	**2**	2	
			18	**18**	25	18				
				10	26	**10**	10			
		26	**26**	**26**	27	26				
				6	28	**6**	**6**	6		
			22	**22**	29	22				
				14	30	**14**	14			

Figure 4.1: The recursive nature of the shuffling.
On the right hand side columns, a bold face number must be less than its nearest neighbors; on the left hand side, a bold face number must be greater than its nearest neighbors.

Here

$$coord + 1 = (a_N, a_{N-1}, a_{N-2}, \dots, a_1, a_0),$$

$$label\_value + 1 = (a_N, a_0, a_1, \dots, a_{N-2}, a_{N-1}),$$

and the variables *coord* and *label_value* are the coordinate and label number of the site, respectively.

In short, ignoring the increment by 1, the 1-d λ-shuffling, λ_1, is a bit reversion operation, which leaves the first bit in the coordinate of the array frozen.

The merging of clusters involves a reduction operation defined as follows.

Definition 4.2.

For a sequence of label numbers obtained by applying the shuffling λ_1 on an array of length 2^N, the *h-reduction*, $\Lambda = \{\Lambda_0, \Lambda_1\}$, is an onto mapping which, except for the number $2^N - 1$ at the end, extracts either the even numbers from the sequence (Λ_0), or the odd numbers (Λ_1). Namely, for $(\nu_1, \nu_2, \nu_3, \nu_4, \dots, \nu_{2^N-2}, \nu_{2^N-1}, \nu_{2^N})$, a given sequence of label numbers, the reduction mappings are

$$\Lambda_0 : \quad (\nu_1, \nu_2, \nu_3, \nu_4, \dots, \nu_{2^N-2}, \nu_{2^N-1}, \nu_{2^N}) \;\rightarrow\; (\nu_2, \nu_4, \dots, \nu_{2^N-2}), \tag{4.2}$$

$$\Lambda_1 : \quad (\nu_1, \nu_2, \nu_3, \nu_4, \dots, \nu_{2^N-2}, \nu_{2^N-1}, \nu_{2^N}) \;\rightarrow\; (\nu_1, \nu_3, \dots, \nu_{2^N-1}). \tag{4.3}$$

The reduction mappings, Λ_0 and Λ_1, can be re-applied to the sequence repeatedly. Also, any intermixed application of Λ_0 and Λ_1 is a legitimate mapping on the sequence, e.g., $\Lambda_0\Lambda_0\Lambda_1\Lambda_0$, etc, as long as they are applied no more than N times. Notice that only the very first reduction leaves out the label number at the end of the sequence.

Using these two operations, 1-d λ-shuffling and h-reduction, we are now able to prove a recursive feature hidden in the hierarchical tree for the problem.

Theorem 4.1.

The sequence obtained by the 1-d λ-shuffling is *strictly oscillating* at every stage of h-reduction, excluding the last number of the original unreduced sequence. By strictly oscillating, we mean that every number in the sequence is either greater than its nearest neighbors or less than its nearest neighbors.

Proof.

Suppose $x-1$ is the coordinate of a site in an array of length 2^N, i.e., $1 \leq x < 2^N$. Let η_x be the image of x under λ_1, i.e., $\eta_x = \lambda_1(x)$. In other words, if

$$x = (a_N(x), a_{N-1}(x), a_{N-2}(x), \dots, a_1(x), a_0(x)),$$

then

$$\eta_x = (a_N(x), a_0(x), a_1(x), \ldots, a_{N-2}(x), a_{N-1}(x)).$$

Now, if the site is even, i.e, $a_0(x) = 0$, then its nearest neighbors, if they exist, must be odd, i.e., $a_0(x-1) = a_0(x+1) = 1$. Therefore, $\eta_{x-1} > \eta_x$ and $\eta_{x+1} > \eta_x$. Similarly, if the site is odd, then $\eta_{x-1} < \eta_x$ and $\eta_{x+1} < \eta_x$. Here it is assumed that $x + 1 < 2^N$. We have now proved that the sequence obtained by 1-d λ-shuffling is strictly oscillating.

Furthermore, it is easy to see that applying the h-reduction is entirely equivalent to extracting numbers from the sequence $\{\eta_x\}$, or from its subsequences, by selecting x's with the same last few binary digits. Thus, after the first reduction the surviving η_x's are those with the same $a_0(x)$, i.e., $a_0(x) = 0$ if Λ_0 reduction is applied and $a_0(x) = 1$ if Λ_1 reduction is applied. After the second reduction, the surviving numbers have the same values of $a_0(x)$ and $a_1(x)$. In general, after l reductions the survivors consist exclusively of the numbers with the same pattern of $\{a_0(x), a_1(x), \ldots, a_{l-1}(x)\}$. In addition, in this reduced sequence, it must be true that $a_l(x-1) \neq a_l(x)$ and $a_l(x+1) \neq a_l(x)$. Hence, this sequence is strictly oscillating. $\square$

The 1-d λ-shuffling described here is by no means uniquely defined by this theorem. *Definition 4.1* is used for mathematical simplicity. From Fig. 3.2. one can see that the order of numbers at the same level in the tree is irrelevant to the hierarchical structure. In other words, re-shuffling numbers at the same level will not have any effect to the recursive feature we have just discussed. Hence, for an array of length 2^N, the number of reshufflings that preserve the identical recursive relation is Γ

$$\Gamma = 2! \times 4! \times 8! \times 16! \times \ldots \times 2^{N-1}! = \prod_{p=1}^{N-1} 2^p! \qquad (4.4)$$

Γ increases very rapidly with N. Taking the leading term in Stirling's approximation, Γ is given approximately by

$$\Gamma \approx \frac{2^{(N-2)2^N}}{e^{(2^N)}} = (\frac{2^{N-2}}{e})^{(2^N)}. \qquad (4.5)$$

When the dimension of the problem is greater than one, this recursive relation is "*contaminated*" to a certain *degree*, depending on the geometrical structure of the cluster components. An example of two dimensions is illustrated in Fig. 4.2. In this example, the components consist of *paths* on the grid. In this cluster, the original shuffling may get rearranged at a certain level of the hierarchical tree, and extra steps are then added to the labeling procedure. The number of extra steps is related to the number of permutations of the *contaminated* recursive relation.

a)

25		24	30	26	28
9		8			12
41		40			44
1		0			4
49		48			52
17		16			20
33	37	32			36

b)

9		8	24	26	12
9		8			12
1		0			4
1		0			4
1		0			4
17		16			20
17	32	16			20

c)

1		0	8	12	4
1		0			4
1		0			4
1		0			4
1		0			4
1		0			4
1	16	0			4

d)

1		0	0	4	4
1		0			4
1		0			4
1		0			4
1		0			4
1		0			4
1	0	0			4

e)

0		0	0	0	0
0		0			0
0		0			0
0		0			0
0		0			0
0		0			0
0	0	0			0

Figure 4.2: The evolution of a snake-shape cluster.

4.2 Estimation Methods

The most difficult component labeling problems are encountered when facing clusters with geometrical structure of great complexity, e.g., those with multiple branches, windings, or zigzags. In an attempt to deal with clusters of arbitrary shape in an optimal way and yet without any *á priori* knowledge of the associated topology, the hierarchical cluster algorithm recursively refines the problem domain implicitly through the reshuffling methods. There is one *representative* root in each cell and, in an abstract sense, at each certain level of the refinement this root is always at the cell site with the minimum label number. The connected components of a cluster successively fuse into a bigger cluster in a hierarchical fashion. This occurs in the direction opposite to the previous refining process, and at each level of the hierarchy the cluster temporarily saturates at the label number of the root. From the standpoint of implementations, the root can be regarded as the *center* in the pseudo code presented in Chapter 3. The hierarchical trees displayed in Fig. 3.2 and Fig. 3.5 are skeletal visualizations of the above scenario.

Before going into the details of analyzing clusters in higher dimensions, we need a mathematical definition.

Definition 4.3.

In a two-dimensional grid, a polyomino P is *column-* (respectively *row-*) *convex* if the intersection of P with any vertical (respectively horizontal) line is connected. A polyomino is *convex* iff it is both column- and row-convex (see examples in Fig. 4.3). The generalization of this definition in n-dimensional space is merely straightforward. A polyomino P in the n-dimensional grid is convex iff the intersection of P with any line along a certain axial direction is connected.

Counting polyominoes is a famous classical problem in combinatorics [17], [18], [31]. It is a rich subject which has attracted continuous attention from mathematicians, physicists, and computer scientists. In physics, for example, polyominoes are considered as special cases of self-avoiding polygons that are used to model crystal growth and polymers [66].[1] On the other hand, they are also applied as pictorial representations of a special class of grammars in context-free languages, *attribute grammars* [31], which are extremely important in compiler design for computers and in syntactical pattern recognition [92]. Searching for the generating functions corresponding to appropriate classes of polyominoes, and exploring the profound algebraic structures of these generating functions is a fascinating

[1]Another example which is familiar to many physicists is the analysis of permutation groups with the aid of Young tableaux [107].

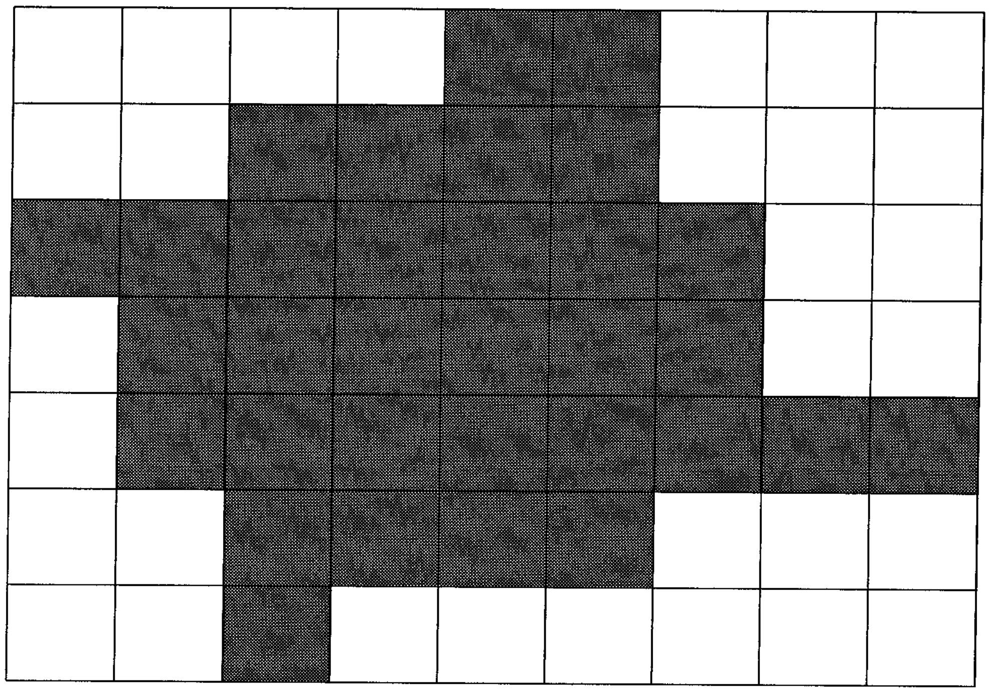


Figure 4.3: A two-dimensional convex polyomino.

area of research. Unfortunately, they are not directly related to the main problem addressed here and are thereby beyond the scope of this thesis.

The shuffling procedure in higher dimensions is defined as follows.

Definition 4.4.

The extension of 1-d λ-shuffling to an arbitrary n-dimensional grid, *n-d* λ-shuffling, is achieved using (3.3) (In $d = 2$, one can use either (3.2) or (3.3)), and λ_n denotes the corresponding mapping on each site, based on the binary expansion of its label.

One merit of the n-d λ-shuffling is that the smallest label number in a convex polyomino is always near its "*geometrical center.*" It is hard to define the geometrical center for an arbitrary convex polyomino and actually a rigorous definition will not be necessary. We can compute the upper bound for the number of steps to complete the labeling of a convex polyomino. By definition, the site with the smallest label number is the aforementioned root, whose associated refined cell corresponds to the highest level of the *minimum* hierarchical tree, or the *partial tree* (see below), in which the polyomino can be embedded. From the proof of *Theorem 3.3* and Fig. 3.5, it can be seen that an upper bound on the number of labeling steps is equal to the height of the *partial tree* in which the convex polyomino is embedded. Notice that the partial tree of a polyomino consists exclusively of nodes corresponding to the sites of the polyomino. The height of the partial tree is derived by counting the maximum length of all possible *orthogonal walks* on the polyomino. An *orthogonal walk* on a polyomino is a very useful technique for measuring the labeling time. It projects the dynamic evolution of the algorithm in the partial tree onto the corresponding polyomino.

Definition 4.5.

An *orthogonal walk* on a polyomino must start from the site, say S_0, with the smallest label number. Starting from S_0, the walk proceeds in any one of the $2d$ axial directions (d = number of dimensions), and temporarily ends at a certain site, say S_1. The *sectional length* of this path is defined to be the logarithm of the number of sites in this path, including the first and the last sites. Re-starting from site S_1, the walk continues in one of the $2d - 2$ orthogonal directions, and temporarily ends at another site. The sectional length of this path is computed in the same way. The walk progresses as far as possible until it reaches the boundary of the polyomino or is about to violate the following restriction rule. The restriction rule imposed on the orthogonal walks is that every chosen orthogonal walk is a *convex tour.* Quite similar to the definition of convex polyomino, a tour is convex iff its intersection with any line along a certain axial direction is connected. Thus for a convex tour, such an intersection is either a line or a single point. The total *length* of the walk is the sum of all sectional lengths.

It is not hard to see that an orthogonal walk explores the *extreme boundary* of the partial tree from its root, the site with minimum label number. Measuring these walks leads to an upper bound on the labeling time of a convex polyomino.

Proposition 4.1.

The number of steps required to label a convex polyomino, after performing the n-d λ-shuffling on the grid, is bounded by the maximum length from all orthogonal walks on the polyomino. This upper bound is less than or equal to the sum of the *logarithmic* widths of the polyomino in all axial directions.

Instead of adding up the logarithms of the lengths of each straight part of an orthogonal walk, it is often helpful to measure the regular length. Intuitively, we know that the logarithm of the maximum regular length of all possible convex tours is a lower bound for the labeling time. This lower bound is normally achieved only if the convex polyomino is very "*slim.*"

How do we deal with non-convex polyominoes? One possibility is to decompose the non-convex polyomino into several convex ones and proceed with the appropriate estimates. Such a decomposition into convex pieces is not unique. We choose here a decomposition algorithm which obeys the following rules:

1) Keep the volume (or the area, in a 2-dimensional grid) of each convex portion as large as possible, i.e., keep the number of pieces as small as possible.

2) The minimum label number in each portion is taken as the *representative* label number. The upper bound on the labeling time for this portion is calculated using the estimation methods stated in *Definition 4.5* and *Theorem 4.2*;

3) If there is any ambiguity in the decomposition, i.e., rule 1) always valid no matter which connected convex portion absorbs this part, yield this part to a connected convex portion with the smallest representative label number;

4) Single out the representative label number from each convex portion and put it in an array, in such a way that adjacent elements in the array represent adjacent portions, and the array should start with the smallest of all the representative label numbers from the connected portions; [2]

[2]When the geometry of the polyomino is not 'simple,' e.g., there exist several branching

5) Compute *merge_rep_p*, which is the number of steps for the array generated by rule 4) to merge into a single numbers, namely the smallest label number in the whole polyomino;

6) Add *merge_rep_p* to the maximum value of the upper bounds calculated in rule 2). This sum is the upper bound for the number of steps needed to label a non-convex polyomino after performing the n-d λ-shuffling on the grid.

This set of rules is quite heuristic and a little bit cumbersome, and they sometimes give an estimate of 1 or 2 steps more than the exact number of steps required. However, the sequence of rules is a simplified procedure to help one realize how the merging process can occur in an efficient way, and how the partial contamination of the hierarchical structure may occur due to the geometry of the cluster. The restriction from *rule 1* on deciding the size of decomposed convex polyominoes can be lifted by dividing the cluster into a greater number of polyominoes of smaller size. This change will not much affect the resulting estimates, since, as stated over and over, the algorithm has a recursive nature. But nonoptimal decomposition only adds unnecessary additional steps to the estimation. One can

components, there is more than one such array generated. Go through rules 5) and 6) for each array, then the maximum value obtained in rule 6) among all arrays is the final answer for this case.

a)

27	25		24	30	26	28	21
11	9		8			12	15
43	41		40			44	47
3	1		0			4	7
51	49		48			52	55
19	17		16			20	23
35	33		32			36	39
59	57	61	56			60	63

b)

11	9		8	24	26	12	15
9	9		8			12	12
3	1		0			4	7
1	1		0			4	4
3	1		0			4	7
17	17		16			20	20
19	17		16			20	23
35	33	56	32			36	39

c)

9	1		0	8	12	4	12
1	1		0			4	4
1	1		0			4	4
1	1		0			4	4
1	1		0			4	4
1	1		0			4	4
17	1		0			4	20
19	17	32	16			20	23

d)

1	1		0	0	4	4	4
1	1		0			4	4
1	1		0			4	4
1	1		0			4	4
1	1		0			4	4
1	1		0			4	4
1	1		0			4	4
17	1	16	0			4	20

e)

1	1		0	0	0	0	0
1	1		0			0	0
1	1		0			0	0
1	1		0			0	0
1	1		0			0	0
1	1		0			0	0
1	1		0			0	0
1	1	0	0			0	4

f)

0	0		0	0	0	0	0
0	0		0			0	0
0	0		0			0	0
0	0		0			0	0
0	0		0			0	0
0	0		0			0	0
0	0		0			0	0
0	0	0	0			0	0

Figure 4.4: The evolution of a thicker snake-shape cluster.

test this estimation method on the clusters shown in Fig. 4.2 and Fig. 4.4 and compare with the actual cluster evolutions. From more testings, one may find that the real number of iteration steps is often to be lower than the predicted upper bound. In general, the real number of iteration steps is equal to the sum of two terms, *max_convex_p* and *merge_rep_p*. Here *max_convex_p* denotes the maximum of the number of iteration steps needed to complete the labeling for the convex polyominoes after the decomposition, and *merge_rep_p* was defined in *rule 5*.

There is an alternative way of obtaining an upper bound which turns out to be much easier. Simply calculate the height of the partial tree which the polyomino is embedded on. For an n-dimensional polyomino of arbitrary geometrical shape, its height of the partial tree is the following sum:

$$Height = \sum_{i=1}^{n} \log W_i, \tag{4.6}$$

here W_i is the width of the polyomino in the i-th axial direction. It is possible that the minimum label number of an axial direction is far from the geometrical center of the polyomino. In this kind of case, W_i should be defined as two times of the geometrical width of the polyomino in this direction, since the extra number of polyomino labeling along this axis is at most 1. However, in the implementation on the Ising system, the data show that assigning W_i as the the geometrical width is closer to the reality. Now we acquire the upper bound as follows.

Theorem 4.2.

Once the n-d λ-shuffling has been performed on the grid, the upper bound on the labeling time for a polyomino of arbitrary geometry is less than or equal to the sum of the logarithmic width in each axial direction. Accordingly, the upper bound on the labeling time for the whole grid is the maximum value of these sums over all polyominoes.

This yields a much better estimate than does *Theorem 3.3.*

4.3 The General λ Shuffling

Before closing this chapter, we would like to re-state some interesting points and define a general method for shuffling. If the array obtained from *rule 5* still preserves the recursive feature of the n-d λ-shuffling, i.e., strictly oscillating, this merging process will terminate in the minimum number of steps, i.e., the lower bound is achieved. When the recursive relation is not perfectly preserved, i.e., is *contaminated*, some extra steps may be required for merging. For example, it takes only one step to merge the array {1,0,2}, but two steps to merge {0,1,2}. In other words, by appropriately re-shuffling the label numbers on parts of the array (or of the column) it is often possible to improve the recursive feature. This shuffling is performed on the grid only once at the very beginning of the program.

However, by re-shuffling the label numbers within some geometrically complex clusters based on some slight *á priori* knowledge of their geometries, one is very likely to achieve the lower bound. Take one trivial example from the 2-d Ising system: If one knows the largest cluster of the system, especially at criticality, tends to percolate along one certain axial direction, then one should take this direction as the x-axis in *Theorem 3.1* or the first axis in *Theorem 3.2*.

Besides re-shuffling the label numbers at the same level of the hierarchical tree introduced in Section 4.1, greater freedom for varying the shuffling is allowed, still maintaining the hierarchical property. We introduce, then, a shuffling of more general form.

We confine our discussion to the 1-d array, since the shuffling along each axis of the grid can be chosen independently, then combined using (3.2) or (3.3).[3] The shuffling rule is as follows. Given an array of length N, i.e., with N sites, a hierarchical tree of log N levels is to be constructed out of the label numbers, ranging from 0 through $N-1$. For the simplicity assume N to be a power of 2, $N = 2^H$. As before, the label number 0 is put at the highest level (level 0), the label numbers 1 and 2 are put at the second level (level 1), etc. The hierarchical

[3]Alternatively, one can, in higher dimensions, first calculate the level order (in the hierarchical tree) of each site by adding the level order from each axial direction (e.g., Fig. 4.5), and then assign the label numbers by this order.

4	3	4	2	4	3	4
3	2	3	1	3	2	3
4	3	4	2	4	3	4
2	1	2	0	2	1	2
4	3	4	2	4	3	4
3	2	3	1	3	2	3
4	3	4	2	4	3	4

Figure 4.5: Level order spreading.

tree is constructed by a single rule: the numbers at any given level are all less than the numbers at higher levels. The hierarchical tree with 4 levels is plotted in Fig. 4.6. The difference between this tree and the one in Fig. 3.2 is that the nodes are only labeled by their level number in the tree. Now we want to squeeze this tree into an array. The method obeys the following iterative rule: for $0 < k < H$

A) Between the $(2l-1)$-st and the $2l$-th label numbers from level k, insert the l-th label number from level $k-1$. Here $l = 1, \ldots, 2^{k-1}$;

B) Always put the label number $N-1$ at the end of the array.

Notice that this rule requires that, for example, the second label number, say μ, from level 2 must lie between the third and the fourth label numbers, say α

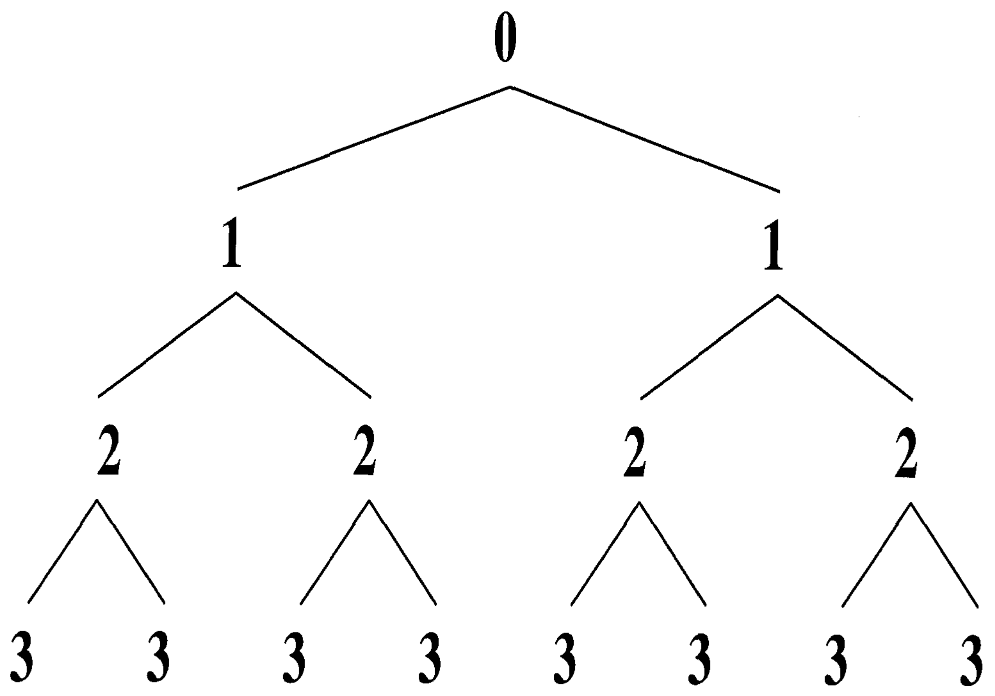


Figure 4.6: Another type of hierarchical structure.
The tree with only level numbers on the nodes, before being squeezed into a 1-d array.

a)	7	3	1	11	0	9	5	13	8	4	12	10	2	6	14
b)	3	1	1	0	0	0	5	5	4	4	4	2	2	2	6
c)	1	0	0	0	0	0	0	0	2	2	2	2	2	2	2
d)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.7: A 1-d cluster evolution after a more flexible shuffling.

and β from level 3, but does not specify how many label numbers from the lower levels, 4, 5, etc., should be inserted between μ and α and between μ and β. Thus there is plenty of freedom to re-arrange the label numbers within the restrictions enforced by the above rule. An array with label numbers arranged differently from 1-d λ-shuffling but obedient to this rule is shown in Fig. 4.7. One finds that the cluster is still able to merge in a hierarchical fashion. Basically, most of mathematical characteristics, except the strict oscillation property of the λ_1 mapping, are preserved by this general shuffling method. The strict oscillation of the original shuffling is replaced by a more flexible recursive rule, called *sub-rule A)*: *in the array every number from (hierarchical) level k is bounded on two sides by two numbers from level $k-1$*. We call the shuffling method *general λ_n shuffling* if this recursive rule is obeyed on each axis of an n-dimensional grid.

Re-shuffling the label numbers along some of the axial directions in the grid is equivalent to shifting the position of the *root* in the cell at different levels of the hierarchical tree. Our main purpose is to keep the root in the position closest to the center of the component enveloped by the cell and also to maintain the hierarchical structure, or the recursive regularity, as well as possible. By performing reshuffling (i.e., shifting some roots in some cells at some certain levels in the hierarchical tree) which incorporates any amount of *á priori* knowledge, if available, about the geometrical configurations of clusters, one is very likely to have the labeling complete within an optimal time complexity. Even when this knowledge is in a probabilistic form, a "probabilistic" or "randomized" shuffling scheme is still worth attempts and thus leads to an even more interesting research topic [61]. This is the major payoff for searching for such shufflings, and the technique is particularly necessary when the geometrical complexity of clusters is high and the dimension of the problem is large. The number, Ω, of different shufflings on an array of length $N = 2^H$, obeying the above rule, is given by

$$\Omega = \Gamma \times (3 + 4 \cdot \wp \cdot 3^{2\wp}), \tag{4.7}$$

where, $\wp = 2^{H-2} - 1$ and Γ has been defined in (4.4).

Chapter 5

Other Hierarchical Algorithms

5.1 Divide-and-Conquer and Shuffling

The divide-and-conquer technique is important for dealing with most complex problems. Scientific algorithms often employ this technique, as has been pointed out in earlier chapters. Motivated by the rapid progress in parallel computing, the divide-and-conquer strategy is the unifying principle common to a large number of parallel algorithms for various computational models. Using this strategy, a parallel algorithm can be viewed as a collection of independent task (or computation) modules that execute in parallel and communicate with each other in order to share intermediate results of the computation. Not confined solely to the design of algorithms, this methodology has become increasingly common in different facets of computing, for example, the design of hierarchical computer architectures [24], [51] and in the *Divacon* parallel programming language. In the construction of conventional programming languages, the emphasis on "*modularity*" in "*structured programming*" and "*object-oriented programming*" is yet another example of this basic idea.

There are a number of problems that exhibit parallelism explicitly, rather than implicitly, for instance the scalar product of vectors. For this sort of problem, the divide-and-conquer strategy is simple and straightforward to apply. However, in the most interesting problems, finding the "*right way*" to divide and conquer and to thus optimize the speedup can be very subtle. Very often, proper use of divide-and-conquer demands an appropriate shuffling of the objects involved in the computation. The hierarchical cluster labeling algorithm described in the previous two chapters is one example, where a special shuffling of the grid sites occurs at the very beginning of the algorithm. In this chapter, we discuss a famous algorithm of this kind, the *fast Fourier transform* (FFT). It is also a good example that embodys the spirit of "*split-and-hit*" in its application of the divide-and-conquer method.

Any divide-and-conquer hierarchical algorithm incorporates two primary actions: 1) a certain type of shuffling, trivial or non-trivial, of the elements in the problem space, and 2) a repetitive utilization of a recursive relation coordinating the course of the algorithm. The elements in the problem space of the (discrete) Fourier transform are the elements of the vector to be transformed or the entries in the matrix of transformation. The FFT algorithm is essentially another application of exchange shuffling (Equation (5.9)) and recursive re-scaling of two halves of a vector (Equation (5.10)). The exchange shuffling divides the vector into two sub-vectors, whose components are respectively the even-indexed and the odd-indexed entries of the original vector. The transformation is partially computed at each stage of recursion, in which the two sub-vectors are combined with an appropriate weight.

Compared to the FFT, the bitonic sort [62], an important sorting algorithm of hierarchy-type, appears to be simpler, though the proof of its validity is more subtle. The elements in the problem space are the numbers in the sequence to be sorted. The shuffling procedure divides the sequence into two subsequences of the same length and then recombines them into a new sequence by alternately interleaving the numbers from two subsequences, i.e., the reverse action of the shuffling occurring in the FFT. After this, the operation of *compare-exchange* seen in almost all sorting algorithms is performed. It can be proved that a sequence of length N is sorted after $O(\log^2 N)$ parallel steps.

Unlike in these two algorithms, the shuffling in the hierarchical cluster labeling algorithm occurs only once at the very beginning, although the re-shuffling may further optimize the performance when some *á priori* knowledge of the cluster geometry is available. The basic idea of the algorithm is to decompose (*divide*) the problem domain into different portions at each stage of recursion or at each level of the hierarchical tree, through the (*general*) λ *shuffling* procedure, and then merge (*conquer*) the connected components, following a path toward the root of the tree. The decomposition is only implicitly defined by the shuffling, and, as explained in the last chapter, the re-shuffling may change the decomposition. Another big difference is that the FFT and the bitonic sort are actually 1-dimensional algorithms, while the hierarchical cluster labeling algorithm is intended for problems in any dimension. Since the algorithm is designed for cluster labeling, the computation mainly involves comparison of label numbers. There should be no difficulty in applying similar ideas to parallelize most prefix algorithms in higher dimensions on an arbitrary domain geometry, thereby extending the method as a generalized *scan scheme* [16] for parallel computation.

5.2 The Fast Fourier Transform

No algorithm has had a greater influence in the recent past than the fast Fourier transform [78]. The speedup obtained using this algorithm is all the more important due to the many applications of the FFT. The nature of divide-and-conquer strategy contained in this algorithm makes it worthwhile to describe the FFT in detail.

The Fourier transform of a continuous function $x(t)$ is given by

$$y(f) = \int_{-\infty}^{\infty} x(t)\; e^{-2\pi i f t}\; dt, \tag{5.1}$$

while the inverse transform is

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} y(f)\; e^{2\pi i f t}\; df. \tag{5.2}$$

Most functions one deals with computationally have to be discretized for implementation. Corresponding to the continuous Fourier transform, then, is the *discrete* Fourier transform (DFT) which handles sample points of $x(t)$, namely $x_0, x_1, \ldots, x_{N-1}$. The discrete Fourier transform is defined by

$$y_j = \sum_{0 \le k \le N-1} x_k \, e^{-2\pi ijk/N} \qquad 0 \le j \le N-1, \tag{5.3}$$

and the inverse is

$$x_k = \frac{1}{N} \sum_{0 \le j \le N-1} y_j \, e^{2\pi ijk/N} \qquad 0 \le k \le N-1. \tag{5.4}$$

In short, the discrete Fourier transform of an N-vector $\vec{x}$ is a linear transformation defined by $\vec{y} = F_N \vec{x}$, where the i, j entry of F_N is ω_N^{ij}, $0 \le i,\ j < N$, and ω_N is a primitive N-th root of unity, namely, $\omega_N^N = 1$ and $\omega_N^j \ne 1$ for $0 < j < N$. For $N = 8$, the transformation matrix, using $\omega = \omega_8$, the primitive 8-th root of unity, is

$$F_8 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix}.$$

The key idea behind the FFT is to establish a connection between F_n and $F_{n/2}$, and this connection enables one to compute very quickly an n-point DFT from a pair of $(n/2)$-point DFTs, i.e., a realization of the divide-and-conquer method. The "*divide*" part of the FFT groups the points into two sets, one containing the even-indexed points and one for the odd-indexed points. For simplicity of discussion, the number of points N is assumed to be a power of 2. The problem of computing $\vec{y} = F_N\vec{x}$ can be reduced to the problem of computing

$$\vec{u} = F_{N/2} \begin{pmatrix} x_0 \\ x_2 \\ x_4 \\ \vdots \\ x_{N-2} \end{pmatrix} \quad \text{and} \quad \vec{v} = F_{N/2} \begin{pmatrix} x_1 \\ x_3 \\ x_5 \\ \vdots \\ x_{N-1} \end{pmatrix}, \tag{5.5}$$

where $\omega_{N/2} = \omega_N^2$ is the primitiv $(N/2)$-th root of unity used to define $F_{N/2}$. To see this, consider, for $0 \leq i < N/2$,

$$\begin{aligned} y_i &= \sum_{0 \leq j < N} \omega_N^{ij} \, x_j \\ &= \sum_{\substack{\text{even } j \\ 0 \leq j < N}} \omega_N^{ij} \, x_j + \sum_{\substack{\text{odd } j \\ 0 \leq j < N}} \omega_N^{ij} \, x_j \\ &= \sum_{0 \leq k < N/2} \omega_N^{i2k} \, x_{2k} + \sum_{0 \leq k < N/2} \omega_N^{i(2k+1)} \, x_{2k+1} \\ &= \sum_{0 \leq k < N/2} \omega_{N/2}^{ik} \, x_{2k} + \omega_N^i \sum_{0 \leq k < N/2} \omega_{N/2}^{ik} \, x_{2k+1} \\ &= u_i + \omega_N^i v_i. \end{aligned}$$

Similarly, for $N/2 \leq i < N$,

$$\begin{aligned} y_i &= \sum_{0 \leq k < N/2} \omega_{N/2}^{(i-N/2)k} \, x_{2k} + \omega_N^i \sum_{0 \leq k < N/2} \omega_{N/2}^{(i-N/2)k} \, x_{2k+1} \\ &= u_{i-N/2} + \omega_N^i v_{i-N/2}. \end{aligned}$$

Therefore, once $\vec{u}$ and $\vec{v}$ are computed, $\vec{y}$ can be retrieved via the connection

$$y_i = \begin{cases} u_i + \omega_N^i \; v_i & \text{if } 0 \le i < N/2 \\ \\ u_{i-N/2} + \omega_N^i \; v_{i-N/2} & \text{if } N/2 \le i < N. \end{cases} \tag{5.6}$$

The formulae are best illustrated in matrix form, in a manner revised from [113]. Consider the $N = 4$ case, we have then

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix},$$

where $\omega = \omega_4 = \exp(-2\pi i/4) = -i$ and $\omega^4 = 1$. Let Π_4 be the 4×4 permutation matrix which splits the points into two parts, i.e., the *shuffling operator*:

$$\Pi_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

such that

$$\Pi_4\,\vec{x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_2 \\ x_1 \\ x_3 \end{pmatrix}.$$

In general, the inverse operator of a shuffling operator is its transpose, i.e., $\Pi_n \Pi_n^T = \Pi_n^T \Pi_n = I_n$. For $n = 4$, the shuffling operator is its own transpose, i.e., $\Pi_4^T = \Pi_4$, and one has the relation $F_4\,\vec{x} = F_4\,\Pi_4^{-1}\,\Pi_4\,\vec{x} = (F_4\Pi_4)\,(\Pi_4\vec{x})$. If we further define a 2×2 block matrix as follows,

$$\Omega_2 = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix},$$

and recall that

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

then

$$F_4\,\Pi_4 = \begin{pmatrix} F_2 & \Omega_2\,F_2 \\ F_2 & -\Omega_2\,F_2 \end{pmatrix}.$$

Thus, each block of $F_4\,\Pi_4$ is either F_2 or a *diagonal scaling* of F_2.

The general form that connects F_n and $F_{n/2}$ is the heart of the FFT algorithm. Let us now work out this form for even n. The shuffling operator Π_n is a $n \times n$ matrix which, when applied to a vector, groups the even-indexed components first and the odd-indexed components second. The form of Π_n is simply a reshuffling of the columns of the identity matrix I_n. The inverse of the shuffling operator, $\tilde{\Pi}_n = \Pi_n^T$, is called the *co-shuffling operator.* The general form of the connection in matrix form is described in the following theorem.

Theorem 5.1. [1]

If $n = 2m$ and the *diagonal scaling operator* is defined by

$$\Omega_m = \text{diag}(1, \omega_n, \omega_n^2, \ldots, \omega_n^{m-1}), \tag{5.7}$$

then

$$F_n \, \tilde{\Pi}_n = \begin{pmatrix} F_m & \Omega_m F_m \\ F_m & -\Omega_m F_m \end{pmatrix} = \begin{pmatrix} I_m & \Omega_m \\ I_m & -\Omega_m \end{pmatrix} (I_2 \otimes F_m). \tag{5.8}$$

Proof.

[1]This theorem is revised from that of C. Van Loan [113].

If $0 \leq p, q < m$, then

$$\begin{array}{llllllll}
(F_n\tilde{\Pi}_n)_{p,q} & = & \omega_n^{p(2q)} & = & \omega_m^{pq} & & = & (F_m)_{p,q} \\
(F_n\tilde{\Pi}_n)_{p+m,q} & = & \omega_n^{(p+m)(2q)} & = & \omega_m^{(p+m)q} & & = & (F_m)_{p,q} \\
(F_n\tilde{\Pi}_n)_{p,q+m} & = & \omega_n^{p(2q+1)} & = & \omega_n^p\omega_m^{pq} & & = & (\Omega_m F_m)_{p,q} \\
(F_n\tilde{\Pi}_n)_{p+m,q+m} & = & \omega_n^{(p+m)(2q+1)} & = & \omega_n^p\omega_m^{pq}(\omega_n^{n/2} = -1) & & = & (-\Omega_m F_m)_{p,q}
\end{array}$$

Here $I_2 \otimes F_m \equiv \mathrm{diag}(F_m, F_m)$. □

Theorem 5.1 contains the recursive property of the FFT algorithm. If

$$\Pi_n \vec{x} \equiv \begin{pmatrix} \vec{x}_{\text{even}} \\ \\ \vec{x}_{\text{odd}} \end{pmatrix}, \tag{5.9}$$

where $\vec{x}_{\text{even}}$ is the vector containing only the even-indexed components of $\vec{x}$ and $\vec{x}_{\text{odd}}$ the odd-indexed ones, then the core recursive relation is

$$F_n\,\vec{x} = \begin{pmatrix} I_{n/2} & \Omega_{n/2} \\ \\ I_{n/2} & -\Omega_{n/2} \end{pmatrix} \begin{pmatrix} F_{n/2}\,\vec{x}_{\text{even}} \\ \\ F_{n/2}\,\vec{x}_{\text{odd}} \end{pmatrix}. \tag{5.10}$$

These two formulae explicitly exhibit the fact that the FFT algorithm is nothing more than an alternate application of *even-odd exchange shuffling* along with proper *re-scaling.*[2]

[2]Computational linear algebra involves mainly matrix factorization. Besides the FFT, the *fast wavelet transform* is a more complicated example which exposes the beauty of the decomposition and shuffling method [101]. Mathematically, the wavelet basis is similar to the Fourier basis in that it forms an orthonormal basis for square-integrable functions. However, unlike the Fourier

Since $F_n\ \vec{x}$ can be computed in a single parallel step given $F_{n/2}\ \vec{x}_{\text{even}}$ and $F_{n/2}\ \vec{x}_{\text{odd}}$, the total time needed to compute $F_n\ \vec{x}$ is given by the recursion relation

$$T(N) = T(N/2) + 1.$$

Therefore the FFT algorithm takes $O(\log N)$ parallel steps, or $O(N\log N)$ sequential steps.

The inverse DFT preserves a recursive form nearly identical to that of the FFT, since the i, j entry of the inverse matrix F_n^{-1}, denoted by $\tilde{F}_n$, is simply

transform, the wavelet transform expands an arbitrary function into a hierarchy of contributions labeled by a scale and a position parameter and hence better captures the multi-length-scale features of the function. The rate of progress on this subject has increased significantly in recent years. It has been realized that many ideas and techniques related to this transform can give rise to straightforward calculational methods applicable to mathematical analysis, theoretical physics, and engineering [25], [30]. To give the reader a sense of this transform of increasing importance, here, we would like to take a simple example –the Haar wavelet. Without going into any detail, one can consider the Haar wavelet as a transform which constitutes a local (orthogonal) basis for piecewise-constant functions and whose *translations* and *dilations* are mutually orthogonal (a brief definition). The Haar wavelet of the lowest rank is a 2 by 2 matrix W_2,

$$W_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = F_2.$$

In contrast to the recursive relation of the FFT

$$F_{2n} = \begin{pmatrix} I_n & \Omega_n \\ I_n & -\Omega_n \end{pmatrix} \begin{pmatrix} F_n & \\ & F_n \end{pmatrix} \Pi_{2n},$$

the fast Haar wavelet transform has the following recursive form

$$W_{2n} = \begin{pmatrix} W_2 & & \\ & \ddots & \\ & & W_2 \end{pmatrix} \Pi_{2n} \begin{pmatrix} W_n & \\ & I_n \end{pmatrix};$$

the matrix W_2 appears n times in the left factor. This is a beautiful "pyramid scheme," which drops the complexity of the operation from $O(n\log n)$ to $O(n)$.

ω_n^{-ij}/n. Following nearly the same algebraic manipulations contained in the proof of *Theorem 5.1*, one has the following corollary.

Corollary 5.1.

If $n = 2m$ and the *co-diagonal scaling operator* is defined by

$$\tilde{\Omega}_m = \mathrm{diag}(1, \omega_n^{2m-1}, \omega_n^{2m-2}, \ldots, \omega_n^{m+1}), \tag{5.11}$$

then

$$\tilde{F}_n \, \Pi_n = 2 \begin{pmatrix} \tilde{F}_m & \tilde{\Omega}_m \tilde{F}_m \\ \tilde{F}_m & -\tilde{\Omega}_m \tilde{F}_m \end{pmatrix} = 2 \begin{pmatrix} I_m & \tilde{\Omega}_m \\ I_m & -\tilde{\Omega}_m \end{pmatrix} (I_2 \otimes \tilde{F}_m). \tag{5.12}$$

Here $\tilde{\Omega}_m$ is the "inverse" of Ω_m, since $\omega_n^{-k} = \omega_n^{2m-k}$, for arbitrary integer k. Also, the factor 2 comes from the fact that $(\tilde{F}_n)_{i,j} = \omega_n^{-ij}/n$, $(\tilde{F}_m)_{i,j} = \omega_m^{-ij}/m$, and $n = 2m$. If

$$\Pi_n \vec{y} \equiv \begin{pmatrix} \vec{y}_{\text{even}} \\ \vec{y}_{\text{odd}} \end{pmatrix}, \tag{5.13}$$

then the recursion relation for the inverse DFT is

$$\tilde{F}_n \, \vec{y} = 2 \begin{pmatrix} I_{n/2} & \tilde{\Omega}_{n/2} \\ I_{n/2} & -\tilde{\Omega}_{n/2} \end{pmatrix} \begin{pmatrix} \tilde{F}_{n/2} \, \vec{y}_{\text{even}} \\ \tilde{F}_{n/2} \, \vec{y}_{\text{odd}} \end{pmatrix}. \tag{5.14}$$

Except for the factor of 2 and $\tilde{\Omega}_{n/2}$ in place of $\Omega_{n/2}$, this recursion relation is a replica of that for the FFT.

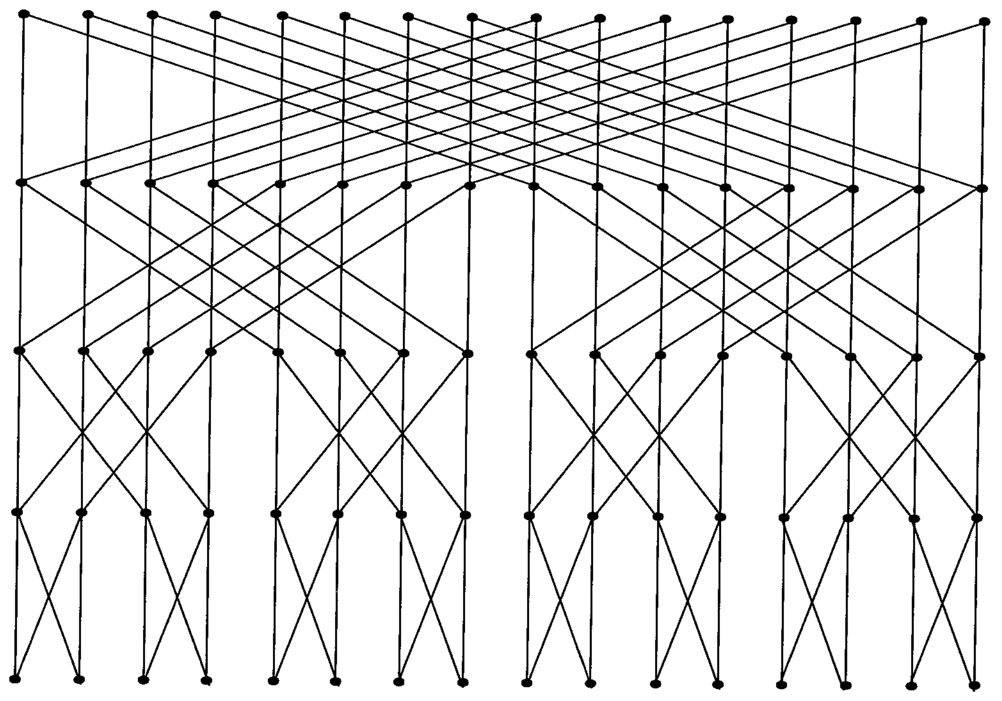


Figure 5.1: The 16-point FFT.
The input is “pumped in” from the top; the paths in the diagram depict the shuffling process required in the transform.

The FFT algorithm is one of the very few algorithms which is easily parallelizable and is fast in both the sequential and parallel domains. It has been extensively studied since its invention in the 1920's. Special hardwares have even been designed to implement this algorithm. A famous example is shown in Fig. 5.1. This figure illustrates the so-called *butterfly* network, for 16 points. The diagram gives a very clear picture of when and where the divide-and-conquer and (exchange) shuffling take place in the course of FFT computation. After an appropriate shuffling on the input points, the whole procedure of an 8-point FFT is shown in Fig. 5.2. One can easily see the similarities between these two diagrams.

One may have noticed that in the proof of *Theorem 5.1* only the two facts $\omega_n^2 = \omega_{n/2}$ and $\omega_n^{n/2} = -1$ are used. However, the essential property that ω_n is a *primitive* root of unity is never mentioned in the proof. The reason for selecting ω_n to be a primitive root is to ensure the existence of F_n^{-1}, the inverse transform.

It is interesting to note that the FFT algorithm can speed up the multiplication of polynomials [64], [78], Here the inverse transform is necessary. Let $f(x) = c_{N-1}x^{N-1} + c_{N-2}x^{N-2} + \ldots + c_0$ be any $(N-1)$-degree polynomial and ω_N be a

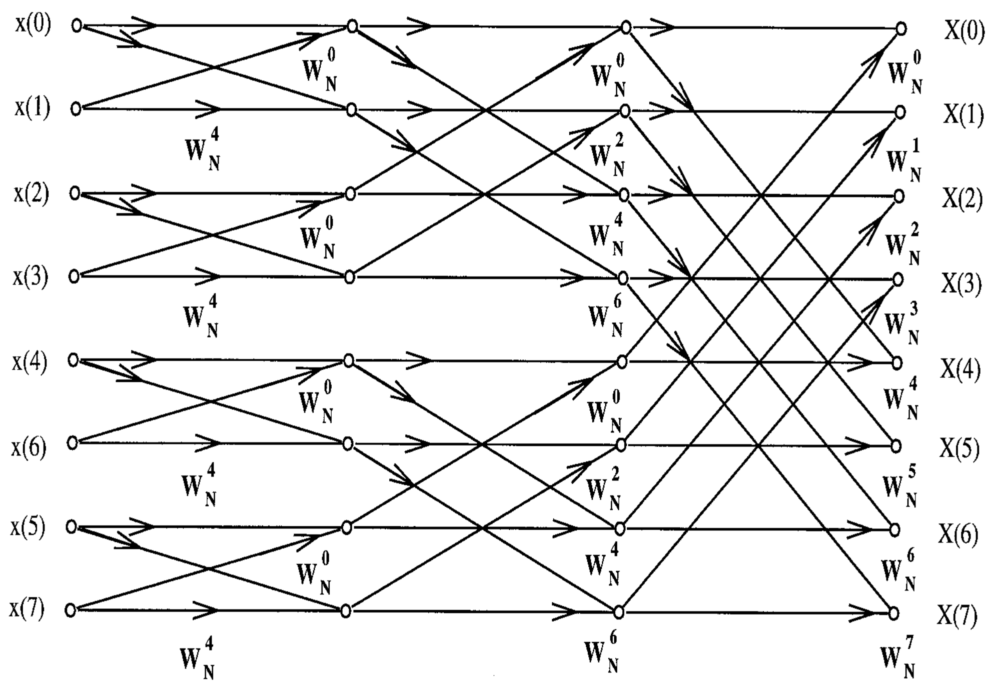


Figure 5.2: The 8-point FFT.
An 8-point FFT after shuffling the input points, where W_N^i represent the scaling weights along the paths. The shuffling paths are in the opposite direction, right to left, if the input points are not shuffled, i.e., the same pattern as that of 8 points in Fig. 5.1.

primitive N-th root of unity. It follows that

$$\begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ f(\omega_N^2) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix} = F_N \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{pmatrix},$$

since $f(\omega_N^i) = c_0 + c_1\omega_N^i + c_2\omega_N^{2i} + \ldots + c_{N-1}\omega_N^{(N-1)i}$ for $0 \leq i < N$. Thus, *performing a discrete Fourier transform is equivalent to evaluating a polynomial at the N N-th roots of the unity.* Conversely, the polynomial can be *interpolated* by taking the inverse DFT,

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{pmatrix} = F_N^{-1} \begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ f(\omega_N^2) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix}.$$

Clearly, the powers of ω_N must be distinct, i.e., ω_N must be a primitive N-th root of unity. Since, $(F_N)_{i,j} = \omega_N^{ij}$ for $0 \leq i,j < N$, if $\omega_N^k = 1$ for a certain $0 < k < N$, the entries in column k will all be 1 and F_N would be a singular matrix. Considering now two polynomials $f(x) = a_0 + a_1x + \ldots + a_{M-1}x^{M-1}$ and $g(x) = b_0 + b_1x + \ldots + b_{M'-1}x^{M'-1}$, let $h(x) = c_0 + c_1x + \ldots + c_{N-1}x^{N-1}$ denote the product $f(x)g(x)$, where $N = M + M' - 1$. The strategy of computing $h(x)$ is [64]

1) evaluate $f(x)$ and $g(x)$ respectively at the N N-th roots of unity,
2) evaluate $h(x)$ at the N N-th roots of unity, i.e., for $0 \leq i, j < N$ computing $h(\omega_N^i) = f(\omega_N^i)g(\omega_N^i)$, and
3) interpolate $h(x)$ from its values on the roots of unity.

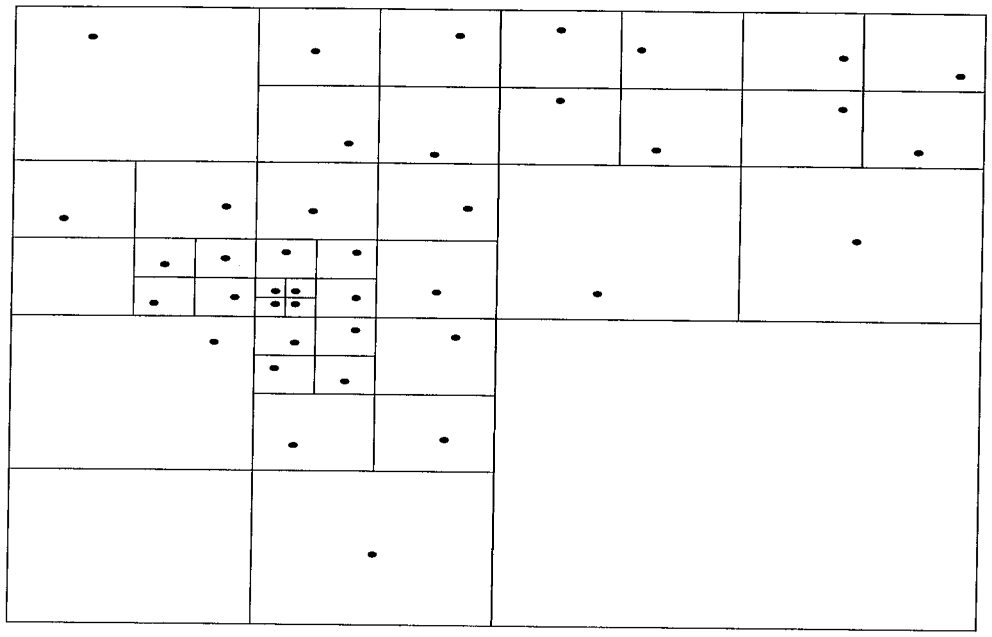
Both steps 1) and 3) can be accomplished using the FFT algorithm and step 2) takes N sequential steps, or only 1 parallel step. Hence, the entire multiplication takes $O(N \log N)$ sequential steps or $O(\log N)$ parallel steps, in contrast to the usual $O(N^2)$ sequential steps and $O(N)$ parallel steps. Since polynomial multiplication is the same as convolution, the FFT algorithm can be used as well in computing convolutions. In particular, from a wider mathematical point of view, the FFT algorithm is applicable on any *commutative ring*, [3] though it is often used for complex numbers or finite fields. More comprehensive discussions of the FFT algorithm in different fields of mathematics are contained in [64], [78].

[3]A set R is a ring with respect to the two operations $\oplus$ and $\otimes$ if the following conditions are fulfilled:
1. $(R, \oplus)$ is an *abelian* group.
2. *Closure* If $c = a \otimes b$, for $a, b \in R$, then $c \in R$.
3. *Associative Law* $a \otimes (b \otimes c) = (a \otimes b) \otimes c$.
4. *Distributive Law* $a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$ and $(b \oplus c) \otimes a = b \otimes a \oplus c \otimes a$.
The *ring* is *commutative* if the law $\otimes$ is commutative.

5.3 Hierarchical N-Body Methods

This last section of Chapter 5 is devoted to a brief discussion of the increasingly popular methods for scientific computation of N-body problems. The N-body problems here are basically concerned with simulations of galaxy evolutions under the influence of gravitational forces. John Salmon's Ph.D. thesis [91] probably contains the best review of these methods, and we refer the reader to this reference for details.

An essential property of these kind of problems is the large range of scales in spatial and temporal information requirements. For example, a point in the problem domain requires less information, usually less frequently, from distant regions of the domain, than from closer regions. A physical system that involves a large number of particles evolving under their mutual gravitational interaction is an example of such a problem. It has been known for about a decade that exploiting this property and thus approaching the problem from a hierarchical point of view can substantially reduce computational complexity. Improvement of these hierarchical methods still continues, and research into possible parallel implementations is ongoing. The Barnes-Hut algorithm is a very successful method for tackling



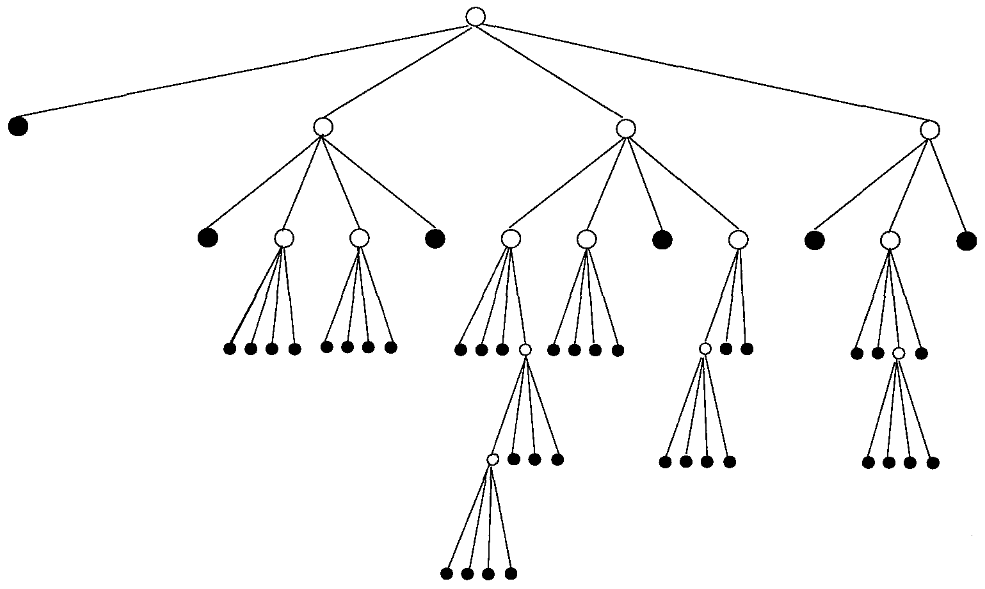
a)

Figure 5.3: A 2-d BH domain subdivision.

N-body problems.

A hierarchical octree, i.e., a tree with at most eight children for each node, is at the heart of the Barnes-Hut algorithm in three dimensions. The root of the tree is a space cell encompassing all particles in the system. A recursive subdivision procedure is performed on the root until each leaf cell contains a single particle. [4] A two-dimensional example and its corresponding quadtree

[4]Some modified versions of the algorithm allow more than one particle in a leaf cell, but this



b)

Figure 5.4: The corresponding quadtree from subdivision.

is shown in Figs. 5.3 and 5.4. The algorithm is designed to simulate evolving systems, so it proceeds in an iterative manner. Within each iterative timestep, the algorithm, in its sequential version, has four phases [81]: building the hierarchical tree, computing the cell centers of mass via an upward pass through the tree, computing the necessary interaction forces, and updating the dynamical properties of the particles.

As mentioned earlier, the hierarchical approach is intended to reduce computational complexity, i.e., to decrease the execution time spent on the force computation phase. The tree is traversed once for each particle to compute the net force acting on that particle. This phase starts from the root and performs the following recursive test for every cell it visits: If the cell center of the mass is far enough away from the body, the entire subtree under that cell is approximated by a single particle at that center of mass, and the force this center of mass exerts on the body is then computed. If, however, the center of mass is not far enough away, the cell must be "opened" and each of its subcells must be visited. The criterion to decide if the cell is far enough away is varied by a user-defined accuracy parameter θ. A cell is said to be far enough away if $l/d < \theta$, where l is the length of the cell sides and d is the distance of the body from the cell's center of mass. In this way,

change does not affect much the basic ideas in the implementation of the algorithm.

bodies in the system are grouped on a hierarchy of length scales, and a particle interacts directly with more levels of those parts of the tree which are spatially closer. Through this approximation scheme, the computational complexity of the sequential Barnes-Hut algorithm is $O(N \log N)$.

For the parallel version of this algorithm, a fifth phase [81], work partitioning, must be added. A good partitioning technique should yield a balanced workload among the processors, and provide data locality. The data locality on a multiprocessor refers the requirement that those particles assigned to a given processor should be close together in space, while particles should be assigned to different processors in such a way that the spatial distance corresponds roughly to the length of communication route. One of the main contributions of Salmon's thesis is a detailed investigation on this partitioning phase. At this point, it may be interesting to compare two hierarchical methods, namely the hierarchical cluster labeling algorithm and the Barnes-Hut algorithm, in terms of this partition. Unlike the FFT and the bitonic sort, these two hierarchical methods should be applicable to domains in arbitrary dimension. Fig. 5.5 illustrates the "slicing" process of the hierarchical cluster algorithm occurring in the direction opposite to that of cluster merging. This picture of slicing demonstrates that cluster labeling

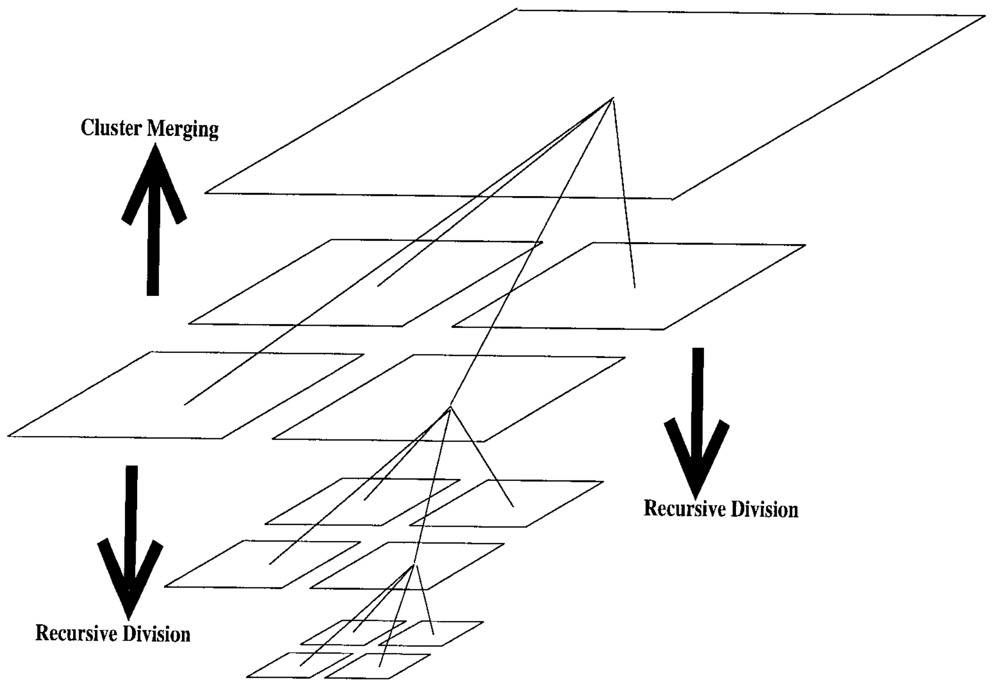


Figure 5.5: The slicing procedure on the problem domain.

can proceed *in parallel on different length scales*, in a simple recursive manner. By contrast, the natural cluster-growth process is inherently sequential and does not reveal any evidence of length-scale dependence of this kind.

From this recursive slicing process, one can see a tree structure created in a way similar to that shown in Fig. 5.4. However, the hierarchical structure obtained in the hierarchical cluster labeling algorithm is a complete n-ary tree, while the one in the parallel Barnes-Hut algorithm an incomplete n-ary tree. In this cluster labeling algorithm, as already explained in the preceding chapters, the hierarchical structure is imposed on the grid through a shuffling of label numbers at the very beginning. That is, the grid (the problem domain), a 2-d mesh for example, is decomposed into 4 subcells first and then *each subcell* undergoes a recursive subdivision procedure. Therefore a *complete* quadtree is created in this procedure, in contrast to the *incomplete* quadtree spawned by the 2-d Barnes-Hut algorithm. This distinction makes a great difference in the partitioning phase. Generally speaking, the partitioning phase of an algorithm associated with an incomplete n-ary tree incurs greater difficulties and must be more sophisticated than an algorithm with a complete n-ary tree.

Before going further into the discussion of the partitioning phase, we would like to bring up an important point. Cluster labeling or connected component identification is a problem of primary importance occurring in many statistical growth models. However, most of the sequential or parallel algorithms that tackle this problem do not necessarily account for any physical characteristics of the system. Cluster-merging in Potts spin systems and percolation models generate patterns via a *step-by-step* process. In other words, this process is strongly history dependent and inherently sequential. We have now proven that the cluster labeling problem can be solved with a logarithmic number of iterations on a parallel machine using a divide-and-conquer scheme. Whether pattern formation problems in other growth models, e.g., diffusion-limited aggregation (DLA) and fluid invasion in porous media, admit efficient parallelization is still an open problem. Efficient parallelization here means that the existence of a parallel algorithm to generate patterns for the problem that runs in polylog time, $O(\log^k N)$ for some k, using only a polynomial number of processors. Interestingly analogous to the theory of NP-completeness, if a parallel algorithm running in polylog time can be discovered for any problem in the class of "parallel P-completeness" (e.g., DLA is in this class and it has been proven to be solvable in polynomial time on a parallel machine), then all problems in this class can be efficiently parallelized [67].

Partitioning a problem to satisfy the criteria of load balancing and data locality is one of the most difficult parts in parallel implementation. Quite often, these criteria are too stringent to deal with, and approximate partitions must be accepted. In his thesis, Salmon proposes a decomposition method, "orthogonal recursive bisection" (ORB), to deal with the partitioning phase in the N-body problem. In simulating galaxy evolution, the workload associated with a particle is roughly proportional to the logarithm of the local density of particles in its neighborhood. Using this workload estimation, ORB iteratively decomposes the spatial domain into several subdomains of different areas (or volumes in three dimensions) with similar workload, and assigns each subdomain to a processor. Notice that this decomposition may be subject to a slow change from one iterative timestep to the next, due to the evolving estimation and the fluctuating particle density. It is important to realize that the nature of the hierarchical algorithm allows each processor to neglect most (but not all) of the tree data. In this N-body problem, a "locally essential" tree is built in each processor, and this tree contains data only up to the level prior to the last ORB which creates the subdomain assigned to this processor. An example of a 2-d domain decomposition after ORB is shown in Fig. 5.6. In this example, the whole domain is divided into 16 subdomains and each of them is assigned a 4-digit binary number that is known as the Gray Code number. On a hypercube multiprocessor, to each processor is addressed by a Gray Code number, and two processors are adjacent to one another if and only if their Gray Codes differ in a single digit.

One may have observed that in this decomposition the criterion of data locality is not completely attained. Thus, some domains share boundaries and yet are

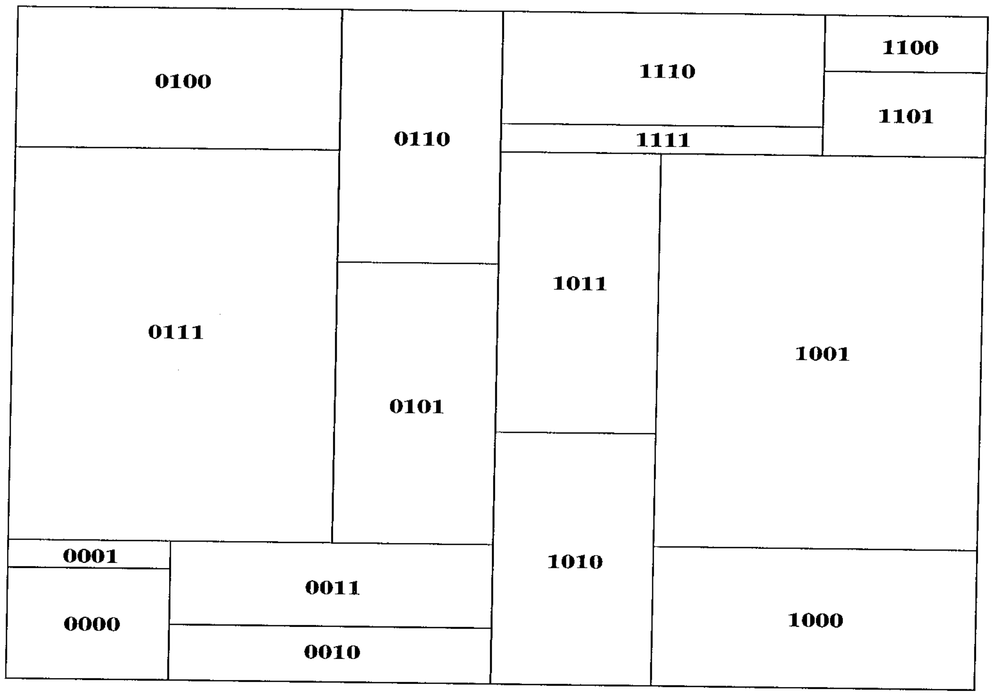


Figure 5.6: Salmon's subdivision.

assigned nonadjacent Gray Codes, e.g., the domains 0000 and 0011 in Fig. 5.6. This occurrence of *dilation* is basically a consequence of the arbitrary (incomplete) tree structure in the Barnes-Hut algorithm and the orthogonal recursive bisection. Because of these types of tree structures, the decomposition of the domain becomes irregular and there is no way in general to keep the Gray Code assignment dilation free. [5] A similar situation occurs when one tries to embed a tree structure in a hypercube. It has been proven [64] that in general a complete binary tree can be embedded in a hypercube with at most dilation 1. But there is no simple, systematic method for constructing a good, i.e., small dilation, mapping from an arbitrary (incomplete) tree onto a hypercube [108]. In contrast, the partitioning phase of the hierarchical cluster labeling algorithm is much easier to deal with. First, the workload on each processor can be roughly estimated as proportional to the logarithm of the area (or volume) occupied by the subdomain after partitioning. Thus, to achieve load balancing, one simply needs to subdivide the whole domain into portions with the same volume. The problem of the Gray Code assignment appropriate to locality now turns out to be trivial. This problem is identical to that of embedding a regular n-d mesh in a hypercube, and the method for doing this is already known. Some examples of the Gray Code assignment are

[5]Fortunately, the appearance of dilation is relatively rare, and does not cause any serious problem according in Salmon's implementation.

000	**001**	**011**	**010**	**110**	**111**	**101**	**100**

a)

Concatenation

	00	**01**	**11**	**10**
00	**0000**	**0001**	**0011**	**0010**
01	**0100**	**0101**	**0111**	**0110**
11	**1100**	**1101**	**1111**	**1110**
10	**1000**	**1001**	**1011**	**1010**

b)

Figure 5.7: Gray Code assignment on mesh.

displayed in Fig. 5.7 to demonstrate the mesh-hypercube embedding.

Chapter 6

Future Directions

The best way to conclude this thesis may be to consider future research directions. Rather than seeking the ultimate answers for the questions raised in the very beginning of the thesis, we shall restrict our attention to further exploration of algorithm improvement for solving increasingly complex problems on high performance computing systems. However, instead of simply concentrating on algorithm design, we believe a broader view should be taken: Algorithm design must be considered in the context of the computer architecture on which it is to be implemented. Otherwise, the best performance can never be achieved. Computer architectures here refer to the abstractions of the hardware and operating system, and mathematically they are computational models in themselves [63], [112]. The underlying theme in the field of high performance computing is: how should the architecture interact with the algorithm and the physical problem in order to obtain "optimal" computation? Put another way, given a specific computing environment, what is the "best" communication design (e.g., through an appropriate arrangement or scheduling) to account for all global communications which must capture long-range features of the problem, and for all local communications which capture the short-range features. This optimization must balance a number of major factors such as speed, accuracy, and a few more practical costs.

A greater and greater number of scientists (e.g., [24] and [51]) have argued that a combined use of hierarchical algorithms and hierarchical architectures should be pursued. Their arguments are bolstered by at least two observations. First, many problems in various scientific disciplines permit treatment by one or more hierarchical algorithms analogous to the ones described in this thesis. This is obviously true for the hierarchical models mentioned in Chapter 1. Second, it is always easier to adapt one hierarchical architecture to another , than it is to adapt a nonhierarchical architecture. In other words, the synthesis of hierarchical algorithms and hierarchical architectures provides a paradigm that motivates a systematic, mathematical analysis of the interactions between algorithms and architectures. For instance, in an intuitive sense, increasing the locality (or range of interactions) of an algorithm often decreases the communication diameter within an architecture, which in turn increases the contention of data during communications. In a hierarchical setting, the variation of locality, communication diameter, and other parameters in the computation, whether deterministic or nondeterministic, are scaled by the level in the hierarchy. This allows a proper mathematical formulation of the whole computation procedure to be derived.

The study of hierarchical architectures is, on the other hand, motivated by the practical concern of designing efficient and cost-effective hardware. It is well known that the hypercube supports a wide class of hierarchical algorithms and is among the most versatile and efficient networks ever developed for parallel computations. One thing that accounts for its popularity is that the hypercube can effectively embed and simulate any other network of the same size, such as an array, binary tree, or mesh of tree with only a small constant multiple of the original cost [64]. There is, however, a fatal disadvantage against the direct enlargement of a hypercube network. Since the number of links needed with a hypercube is a logarithmically increasing function of the total number of processors, the system becomes prohibitively large and it becomes extremely difficult, if not impossible, to fabricate each processor. Considerable effort has been expended looking for other network structures that cope with this problem and yet are still able to preserve all the advantages of the hypercube. In this regard, hierarchy-configured networks arise as a potential choice. Several hierarchical networks have been proposed and even built, such as the fat-tree and clustered hypercube. Advanced design is a matter of trade-off between system efficiency (and complexity) and flexibility. It is still too early to foresee what the best design will be. However, for a large range of problems, exploiting the hierarchical characteristics hidden in the algorithms and thus identifying or classifying the various patterns observed in them may provide important clues to solving these problems.

In spite of the continuing success of hierarchical algorithms, there are many problems that still lack hierarchical approach. Spin glass-like systems are apparently a good example of such problems. As described in Chapter 1, locating the global minimum of the free energy in a spin glass-like system is often NP-complete and no efficient algorithm exists. The main reason for the failure of hierarchical treatments of this type of problem is that *local information gives almost no information about the global picture.* Nevertheless there is a bright side to this failure: Should a hierarchical algorithm for these problems eventually be found, it certainly will tell us whether or not NP = P. Even if no such algorithm exists for these problems, we can still try to understand or to alleviate the degree of difficulty on a hierarchical basis. Thus, in terms of an appropriate hierarchical framework, one can ask how much of the global picture of the problem can be reflected by patching together local information. Such a program may help clarify the distinction between "easy" and "hard" problems, like P and NP problems. With these future directions in mind, I would like to close this thesis as a first step of research in what promises to be an exciting journey.

Appendix A

Huffman Code

Information theory [15] is a discipline focused on mathematical approach to the collection and manipulation of *information.* Several statisticians have made theoretical inquiries into problems of *information* in communication since the 1920's. However, the subject did not take its present form until Claude Shannon laid the central foundation in the 1940's [94]. Strongly influenced by his experience in cryptography during World War II, this work developed from a realization that communication at its most fundamental level is a probabilistic process. Essentially, information theory is intended to answer a number of very basic questions, classified by R. Blahut [15]:

1. What is information? That is, how do we measure it?
2. What are the fundamental limits on the transmission of information?
3. What are the fundamental limits on the extraction of information from the environment?
4. What are the fundamental limits on the compression and refinement of information?
5. How should devices be designed to approach these limits?
6. How closely do existing devices approach these limits?

Owing to the fundamental nature of the questions raised above, information theory has found applications in a variety of disciplines as diverse as physics, artificial neural networks, biology, etc. In classical and quantum physics, many versions of *(physical) entropy* are defined as measurements of *randomness* or *complexity* in a physical or computational system (see Sec. 1.1). These variants of entropy mostly trace their roots back to the original Shannon entropy, and sometimes are simply straightforward applications of the latter in more physical settings. In addition, since quantum mechanics, statistics, and information theory all deal with probabilistic phenomena, i.e., phenomena with incomplete information, connections between these subjects are often based on the Shannon entropy minimum principle. [1] In the highly active research area of artificial neural networks, *learning* theories and algorithms [50] have attracted much attention. The concept and techniques of minimizing the Shannon entropy are often employed in describing the learning capabilities of networks. The question of what is the fundamental mechanism governing the basic units of organic bodies, e.g., DNA, RNA, and cells, and is responsible for the emergence and evolution of life in general has been puzzling scientists for a long time. In attempting to answer this question, more and more evidence has been found to support the idea that life can be considered as a dynamic state of matter organized by information [36].

Given a finite random output sequence $\{a_0, a_1, \ldots, a_{K-1}\}$ from a discrete information source, we would like to write down a mathematical definition of the information contained in this output. If an output, say a_k, occurs with probability $p(a_k)$, then on average the output a_k will be observed once out of every $\frac{1}{p(a_k)}$ outputs. To "encode" this fact, $\log \frac{1}{p(a_k)} = -\log p(a_k)$ *bits* of information are

[1]An interesting and promising development is that such connections can be obtained in the natural meeting ground of differential geometry [4], [5], [20], [21], [60], [75]. As a result, a variety of stochastic dynamics discussed in different disciplines seemingly without any significant overlap can be placed in one common mathematical framework.

required. Here the logarithm is to be the base 2, though this is not necessary. If the number of total selections, n, is very large, a_k will occur approximately $np(a_k)$ times. Therefore the *average information* contained in n outputs is equal to

$$p(a_0)\log\frac{1}{p(a_0)} + p(a_1)\log\frac{1}{p(a_1)} + \ldots + p(a_{K-1})\log\frac{1}{p(a_{K-1})}$$

bits. The *uncertainty* or *entropy* $H(\mathbf{p})$ is defined as follows.

$$H(\mathbf{p}) = -\sum_{k=0}^{K-1} p(a_k)\, \log\, p(a_k) \equiv -\sum_{k=0}^{K-1} p_k\, \log\, p_k, \qquad (A.1)$$

where $p(a_k)$ is abbreviated as p_k.

The entropy thus defined is a very "well-behaved" function with a number of good properties. For instance, the entropy is a convex function of $\mathbf{p}$,

$$\forall\ 0 \le \lambda \le 1 \quad H(\lambda\mathbf{p} + (1-\lambda)\mathbf{p}') - \lambda H(\mathbf{p}) - (1-\lambda)H(\mathbf{p}') \ge 0, \qquad (A.2)$$

i.e., a straight line joining any two points $\mathbf{p}$ and $\mathbf{p}$' on the graph of H($\mathbf{p}$) never lie below the graph. This property enables the entropy to be easily minimized since any local minimum is also the global minimum of a convex function. Obviously, the entropy function is continuous in $\mathbf{p}$. This property guarantees that small changes in the probability distribution only give rise to small changes in the uncertainty. The entropy is an additive function for independent random events, i.e., the entropy is equal to the sum of the entropies of the individual events. It is trivial to see that the entropy is a non-negative function and $H(\mathbf{p}) = 0$ if and only if all p_k but one vanish. Thus there is no uncertainty when the output is sure. Moreover, intuitively we know that the greater number of the unbiased selections available, the larger the uncertainty is. Indeed, the uncertainty is maximum when all selections are equally probable. This fact is equivalent to the property $H(\mathbf{p}) \leq \log K$.

The transmission of a large volume of data demands a very clever use of coding techniques so that the information can be conveyed in a reliable and optimal way. A tremendous amount of effort has been devoted to this problem since the appearance of Shannon's groundbreaking work. A code is a special representation of a string of data that satisfies a given need. Various types of codes are employed to serve different functions subject to many practical conditions on the communication channels [15]. For example, at a certain stage of source data encoding, one might or might not introduce some intentional distortion into the data, depending on the degree of data compression required. Introducing some degree of distortion into the data may yield methods that more concisely encode the data, at the cost of error-free source data recovery. In practice, the communicating channels are usually noisy to some extent and may only allow the transmission of data within certain constraints. Special designs of error-correcting codes meeting these constraints must therefore be developed.

Huffman codes are in the category of *data compaction codes.* This type of code represents the source data more efficiently, but also in such a way that the source data can be recovered essentially error-free. In most practical situations, data streams are normally so long as to appear infinite to the encoder and the decoder. Building long codes from small codewords and breaking the data streams into sequences of these codewords is a feasible way of dealing with large amounts of data. Classified by codeword structure, there are several different kinds of codes, e.g., block codes, tree codes, and prefix codes. Among them, prefix codes admit *self-punctuation*, and are the kind utilized in Huffman codes. By self-punctuation, we mean the code stream which is obtained by concatenating a string of codewords can be uniquely decomposed (punctuated) into the blocks of codewords unambiguously, i.e., an indefinitely long string of code is uniquely decodable. To accomplish this, the so-called *prefix condition* is imposed on the codewords. The condition requires that no codeword is the beginning of any other codewords. More precisely, suppose $\{n_0, n_1, \ldots, n_{K-1}\}$ is the set of codewords. If the codeword n_k is not the beginning of $n_{k'}$ for any $k' \neq k$ and $0 \leq k, k' < K$, then the code is called a *prefix code.* The following is an example of a set of binary prefix codewords.

$$\{00,\ 01,\ 110,\ 11100,\ 11110,\ 11111\}$$

The string

$$0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1 \ldots$$

can only be punctuated as

$$0\ 1,\ 1\ 1\ 0,\ 1\ 1\ 1\ 0\ 0,\ 0\ 1 \ldots$$

Now we know all prefix codes are uniquely decodable, but not vice versa. There is a loss of generality if prefix codes are considered exclusively. Thus a crucial question should be asked: Does there exist a set of numbers $\{n_0, n_1, \ldots, n_{K-1}\}$ which are the codeword lengths of some uniquely decodable code, but not of any prefix code? Fortunately, the answer to this question is *no.* The following *Kraft Inequality* plays a key role in establishing this fact.

Theorem A.1. *(Kraft Inequality)*

In an alphabet of size S, there exists a prefix code with K codewords of length n_k for $k = 0, 1, \ldots, K-1$ if and only if

$$\sum_{k=0}^{K-1} S^{-n_k} \leq 1. \qquad (A.3)$$

For binary codes, the alphabet is $\{0,1\}$ and the size S is 2.

Therefore, assuming the Kraft Inequality holds, without loss of generality, we can address the representation of information while dealing exclusively with prefix codes.

We are now in a position to talk about the celebrated Huffman code. The question is to find the most efficient method of representing numbers, letters or other symbols using a binary code. Let $X = \{a_0, a_1, \ldots, a_{K-1}\}$ be an ensemble with the corresponding positive probabilities $p_0, p_1, \ldots, p_{K-1}$. What is the minimum average number of bits required to represent the outcome, X? Equivalently, can we find a set of codewords with lengths $n_0, n_1, \ldots, n_{K-1}$ and minimize the *average* information $\bar{n} = \sum_{k=0}^{K-1} p_k n_k$, under the constraint $\sum_{k=0}^{K-1} 2^{-n_k} \leq 1$? [2]

The way to construct such a code is as follows. Suppose $K > 2$, let $X = \{a_0, a_1, \ldots, a_{K-1}\}$ be arranged in non-increasing order by the probabilities, i.e., $p_0 \geq p_1 \geq \ldots \geq p_{K-2} \geq p_{K-1} > 0$. In this ordering, a_{K-1} refers to be the

[2]This is the same question [100] was set by Robert Fano as a term paper in a graduate electrical engineering course (in lieu of taking the final exam.) at MIT in 1951. David A. Huffman was one of the students in this class. Several good people, including Claude E. Shannon, the creator of the field of information theory, had struggled with this problem for a long time and were unable to obtain an optimal solution. Without being told this fact, Huffman worked on the problem for months and almost despaired of ever finding a solution. Just before he decided to start studying for the final, an "absolute lightning of sudden realization" hit his mind, and Huffman wrote down a simple and elegant binary code which answered the question.

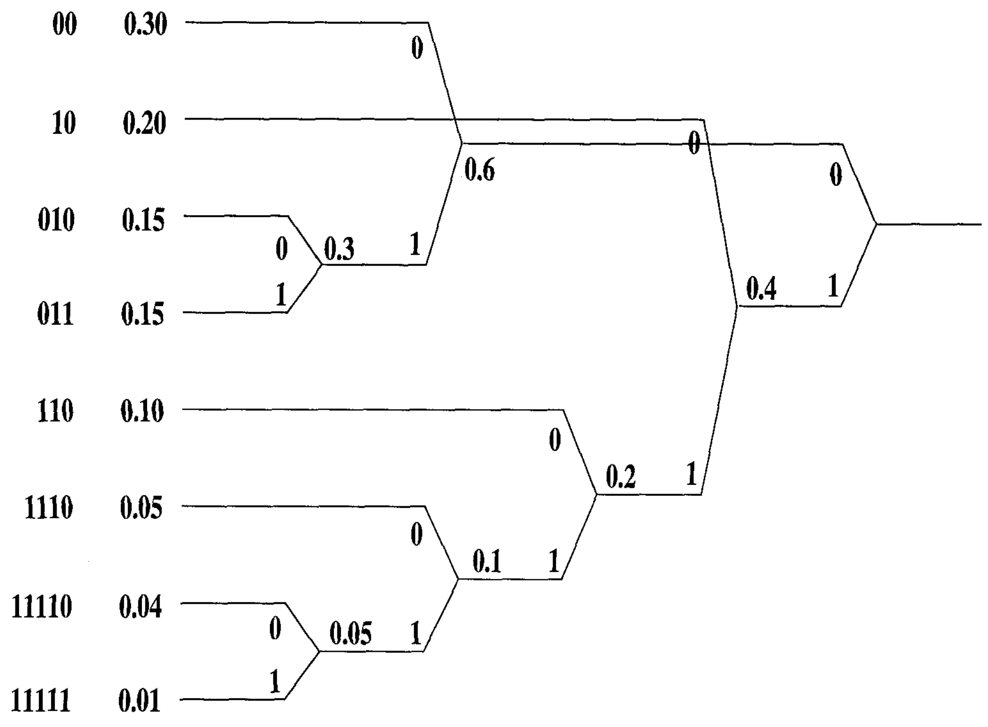


Figure A.1: An example of Huffman Code.

least probable element, and a_{K-2} the next least probable element. Let $X' = \{a'_0, a'_1, \ldots, a'_{K-2}\}$ with probabilities $p'_0, p'_1, \ldots, p'_{K-2}$ given by $p'_k = p_k$ for $0 \leq k \leq K-3$, and $p'_{K-2} = p_{K-2} + p_{K-1}$. Here X' is called the *reduced* ensemble. The Huffman code assigns 1 and 0, respectively, as the *last* digit for the codewords of the two least probable elements a_{K-2} and a_{K-1}. The two least probable elements are then "merged" into a'_{K-2} and this step is repeated recursively on the reduced ensemble. Fig. A.1 shows an example of this procedure.

The optimality of this algorithm is based on two facts contained in the following propositions.

Proposition A.1.

Given any ensemble X, there exists a prefix code that achieves the minimum $\bar{n}$ in which the two least probable elements have the same codewords except for the last digit.

Proposition A.2.

The average codeword length of the encoding of X is minimum iff the average codeword length of the encoding of the reduced ensemble X' is minimum.

These two propositions guarantee an optimal code for X by obtaining an optimal code for the reduced ensemble, after committing to the *initial last-digit* assignment. It recursively proves that Huffman's algorithm indeed yields an optimal prefix code. The rigorous proof can be found in most standard books on the information or coding theory. One more pleasant fact about the Huffman code is that the average codeword length obtained by this algorithm differs from the information, or entropy, $H(X)$ of the ensemble X by no more than 1 bit, i.e., $\bar{n} \leq 1 + H(X)$!

Huffman code has been appraised by Donald E. Knuth as "one of the fundamental ideas that people in computer science and data communications are using all the time." We have re-introduced it here because of its elegant use of *divide-and-conquer*. There is a very amusing application of Huffman's algorithm which we would like to pose as a question to close this chapter. Given m_1 balls of color c_1, m_2 balls of color c_2, ..., and m_K balls of color c_K, Mr. H picks one ball at random with uniform probability from this group of balls. How do we design a minimum set of questions for Mr. H, who is only allowed to answer "yes" or "no," such that we are able to determine the color of the ball?

Bibliography

[1] A. Aho, J. Hopcroft, and J. Ullman, *Design and Analysis of Computer Algorithms*, Addison-Wesley (1979).

[2] H. M. Alnuweiri and V. K. Prasanna, "Parallel Architectures and Algorithms for Image Component Labeling," *IEEE Trans. Patt. Anal. Machine Intell.*, **14**, 1014 (1992).

[3] N. Alon, J. H. Spencer, and P. Erdős, *The Probabilistic Method*, New York: Wiley (1992).

[4] S. Amari, "Differential-Geometrical Methods in Statistics," (Springer Lecture Notes in Statistics vol. 28), New York: Springer (1985).

[5] S. Amari, K. Kurata, and H. Nagaoka, "Information Geometry of Boltzmann Machines," *IEEE Trans. Neural Networks*, **3**, 260 (1992).

[6] J. Apostolakis, P. Coddington, and E. Marinari, "New SIMD Algorithms for Cluster Labeling on Parallel Computers," *Int. J. Mod. Phys. C*, **4**, 749 (1993).

[7] C. P. Bachas, "Computer-Intractability of the Frustration Model of a Spin Glass," *J. Phys. A: Math. Gen.*, **17**, L709 (1984).

[8] C. F. Baillie, "Lattice Spin Models and New Algorithms -A Review of Monte Carlo Computer Simulations," *Caltech Concurrent Computation Project Tech. Rep.*, C^3P-854, submitted to *Int. J. Mod. Phys. C.*

[9] C. F. Baillie and P. D. Coddington, "Cluster Identification Algorithms for Spin Models," *Concurrency: Practice and Experience*, **3**, 129 (1991).

[10] F. Barahona, R. Maynard, R. Rammal, and J. P. Uhry, "Morphology of Ground States of Two-Dimensional Frustration Model," *J. Phys. A: Math. Gen.*, **15**, 673 (1982).

[11] F. Barahona, "On the Computational Complexity of Ising Spin Glass Models," *J. Phys. A: Math. Gen.*, **15**, 3241 (1982).

[12] C. H. Bennett, P. Gács, M. Li, P. M. B. Vitányi, and W. H. Zurek, "Thermodynamics of Computation and Information Distance," *Proc. 25th Annual ACM Symp. on Theory of Computing, San Diego, May 15-18, 1993.*

[13] A. Berthiaume and G. Brassard, "The Quantum Challenge to Structural Complexity Theory," *Proc. 7th IEEE Conf. on Structure in Complexity Theory*, (Jun. 1992).

[14] I. Bieche, R. Maynard, R. Rammal, and J. P. Uhry, "On the Ground States of the Frustrated Model of a Spin Glass by a Matching Method of Graph Theory," *J. Phys. A: Math. Gen.*, **13**, 2553 (1980).

[15] R. E. Blahut, *Principles and Practice of Information Theory,* Addison-Wesley (1987).

[16] G. E. Blelloch, *Vector Models for Data-Parallel Computing*, MIT Press (1990).

[17] M. Bousquet-Melou, "Convex Polyominoes and Heaps of Segments," *J. Phys. A: Math. Gen.*, **25**, 1925 (1992).

[18] M. Bousquet-Melou, "Convex Polyominoes and Algebraic Languages," *J. Phys. A: Math. Gen.*, **25**, 1935 (1992).

[19] R. C. Brower, P. Tamayo and B. York, "A Parallel Multigrid Algorithm for Percolation Clusters," *J. Stat. Phys.*, **63**, 73 (1991).

[20] E. R. Caianiello, "Geometrical Identification of Quantum and Information Theories," *Lett. Nuovo Cimento*, **38**, 539 (1983).

[21] E. R. Caianiello, "Quantum and Other Physics as Systems Theory," *Riv. Nuovo Cimento*, **15**, 1 (1992).

[22] G. Castagnoli, M. Rasetti, and A. Vincenzi, "Steady, Simultaneous Quantum Computation," *Int. J. Mod. Phys. C*, **3**, 661 (1992).

[23] G. J. Chaitin, *Information, Randomness and Incompleteness -Papers on Algorithmic Information Theory*, World Scientific (1987).

[24] T. F. Chan, "Hierarchical Algorithms and Architectures for Parallel Scientific Computing," *Proc. ACM Int. Conf. on Supercomputing*, 318, ACM (1990).

[25] J. M. Combes, A. Grossmann, and Ph. Tchamitchian (eds.), *Wavelets: Time-Frequency Methods and Phase Space*, Springer-Verlag (1989).

[26] A. Coniglio, F. di Liberto, G. Monroy, and F. Peruggi, "Cluster Approach to Spin Glasses and the Frustrated-Percolation Problem," *Phys. Rev. B*, **44**, 12605 (1991).

[27] N. C. A. da Costa and F. A. Doria, "Undecidability and Incompleteness in Classical Mechanics," *Int. J. Theor. Phys.*, **30**, 1041 (1991).

[28] R. Cypher, J. L. C. Sanz, L. Snyder, "Algorithms for Image Component Labeling on SIMD Mesh Connected Computers," *IEEE Trans. Comput.*, **39**, 276, (1990).

[29] S. P. Dandamudi, *Hierarchical Hypercube Multicomputer Interconnection Networks*, Ellis Horwood (1991).

[30] I. Daubechies, *Ten Lectures on Wavelets*, SIAM (1992).

[31] M. P. Delest and J. M. Fedou, "Attribute Grammars are Useful for Combinatorics," *Theor. Comput. Sci.*, **98**, 65 (1992).

[32] D. Deutsch, "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer," *Proc. R. Soc. Lond.*, **A 400**, 97 (1985).

[33] J. J. Dongarra and D. C. Sorensen, "A Fully Parallel Algorithm for the Symmetric Eigenvalue Problem," *2nd SIAM Conf. on Vector and Parallel Processing in Scientific Computing, VA, Nov. 20, 1985.*

[34] J. J. Dongarra, "PVM and HeNce: Tools for Heterogeneous Network Computing," a tutorial on *6th SIAM Conf. on Parallel Processing for Scientific Comput, Norfolk, VA, Mar. 21-24, 1993.*

[35] R. G. Edwards and A. D. Sokal, "Generalization of the Fortuin-Kasteleyn-Swendsen-Wang Representation and Monte Carlo Algorithm," *Phys. Rev. D*, **38**, 2009 (1988).

[36] M. Eigen, *Steps Towards Life: A Perspective on Evolution*, Oxford Univ. Press, (1992).

[37] H. G. Evertz, G. Lana, and M. Marcu, "Cluster Algorithm for Vertex Models," *Phys. Rev. Lett.*, **70**, 875 (1993).

[38] K. H. Fischer and J. A. Hertz, *Spin Glasses*, Cambridge Univ. Press (1991).

[39] M. Flanigan and P. Tamayo, "A Parallel Cluster Labeling Method for Monte Carlo Dynamics," *Int. J. Mod. Phys. C*, **3**, 1235 (1992).

[40] S. Forrest, ed., *Emergent Computation: Self-Organization, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks*, MIT Press (1991) or a special issue of *Physica D*, **42** (1990).

[41] C. M. Fortuin and P. W. Kasteleyn, "On the Random-Cluster Model," *Physica*, **57**, 536 (1972).

[42] G. C. Fox, "Physical Computation," *Concurrency: Practice and Experience*, **3**, 627 (1991), or *Syracuse Center for Computational Science Tech. Rep. SCCS-2*, (1990).

[43] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors*, vol. I, Prentice Hall (1988).

[44] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice Gas Automata for the Navier-Stokes Equation," *Phys. Rev. Lett.*, **56**, 1505 (1986).

[45] Y. Fu, "The Use and Abuse of Statistical Mechanics in Computational Complexity," *Santa Fe Inst. Studies in the Sciences of Complexity Lectures*, vol. 1, ed. D. L. Stein, 815 (1989).

[46] M. R. Garey and D. S. Johnson, *Computer and Intractability*, New York: W. H. Freeman (1979).

[47] H. Gazit, "An Optimal Randomized Parallel Algorithm for Finding Connected Components in a Graph," *SIAM J. Comput.*, **20**, 1046 (1991).

[48] A. M. Gibbons and W. I. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press (1988).

[49] J. Gill, "Computational Complexity of Probabilistic Turing Machines," *SIAM J. Comput.*, **6**, 675 (1977).

[50] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, (Santa Fe Institute Studies in the Sciences of Complexity Lecture Notes vol. 1), Addison-Wesley (1991).

[51] T. Heywood and S. Ranka, "A Practical Hierarchical Model of Parallel Computation: I. The Model & II. Binary Tree and FFT Algorithms," *J. Parallel Distributed Comput.*, **16**, 212 (1992).

[52] D. S. Hirschberg, A. K. Chandra, and D. V. Sarwate, "Computing Connected Components on Parallel Computers," *Commun. ACM*, **22**, 461 (1979).

[53] K. H. Hoffmann, S. Grossmann, and F. Wegner, "Random Walk on a Fractal: Eigenvalue Analysis," *Z. Phys. B*, **60**, 401 (1985).

[54] J. J. Hopfield and D. W. Tank, "'Neural' Computation of Decisions in Optimization Problems," *Biological Cybernetics*, **52**, 141 (1985).

[55] B. A. Huberman and M. Kerszberg, "Ultradiffusion: The Relaxation of Hierarchical Systems," *J. Phys. A: Math. Gen.*, **18**, L331 (1985).

[56] D. Kandel, R. Ben-Av, and E. Domany, "Cluster Dynamics for Fully Frustrated Systems," *Phys. Rev. Lett.*, **65**, 941 (1990).

[57] D. Kandel and E. Domany, "General Cluster Monte Carlo Dynamics," *Phys. Rev. B*, **43**, 8539 (1991).

[58] K. Kaneko, "Clustering, Coding, Switching, and Hierarchical Ordering in Network of Chaotic Elements," preprint LA-UR-89-698 (1989).

[59] I. Kanter, "Undecidability Principle and the Uncertainty Principle Even for Classical Systems," *Phys. Rev. Lett.*, **64**, 332 (1990).

[60] N. Karmarkar, "Riemannian Geometry Underlying Interior-point Methods for Linear Programming," *Contemporary Mathematics*, **114**, 51 (1990).

[61] R. M. Karp, "An Introduction to Randomized Algorithms," *Discrete Appl. Math.*, **34**, 165 (1991).

[62] D. E. Knuth, *The Art of Computer Programming*, vol. 3, 2nd ed., Addison-Wesley (1981).

[63] C. P. Kruskal, L. Rudolph, and M. Snir, "A Complexity Theory of Efficient Parallel Algorithms," *Theor. Comput. Sci.*, **71**, 95 (1990).

[64] F. T. Leighton, *Introduction to Parallel Algorithms and Architecture: Arrays, Trees, Hypercubes*, Morgan Kaufmann, (1992).

[65] S. Levialdi, "On Shrinking Binary Picture Patterns," *Commun. ACM*, **15**, 7 (1972).

[66] K. Y. Lin and W. J. Tzeng, "Perimeter and Area Generating Functions of the Staircase and Row-Convex Polygons on the Rectangular Lattice," *Int. J. Mod. Phys. B*, **5**, 1913 (1991).

[67] J. Machta, "The Computational Complexity of Pattern Formation," *J. Stat. Phys.*, **70**, 949 (1993).

[68] N. Madras and A. D. Sokal, "The Pivot Algorithm: A Highly Efficient Monte Carlo Method for the Self-Avoiding Walk," *J. Stat. Phys.*, **50**, 109 (1988).

[69] M. Mezard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond*, World Scientific (1987).

[70] H. Mino, "A Vectorized Algorithm for Cluster Formation in the Swendsen-Wang Dynamics," *Computer Phys. Commun.*, **66**, 25 (1991).

[71] M. L. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall (1967).

[72] C. Moore, "Unpredictability and Undecidability in Dynamical Systems," *Phys. Rev. Lett.*, **64**, 2354 (1990).

[73] C. Moore, " Generalized Shifts: Unpredictability and Undecidability in Dynamical Systems," *Nonlinearity*, **4**, 199 (1991).

[74] Z. G. Mou, "Divacon: A Parallel Language for Scientific Computing Based on Divide-and-Conquer," *Proc. 3rd Symp. on the Frontier of Massively Parallel Computation, Oct. 1990*, 451, IEEE (1990).

[75] T. Obata, H. Hara, and K. Endo, "Differential Geometry of Nonequilibrium Processes," *Phys. Rev. A*, **45**, 6997 (1992).

[76] A. T. Ogielski and D. L. Stein, "Dynamics on Ultrametric Spaces," *Phys. Rev. Lett.*, **55**, 1634 (1985).

[77] D. Nassimi and S. Sahni, "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer," *SIAM J. Comput.*, **9**, 744 (1980).

[78] H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, Springer-Verlag (1982).

[79] M. D'Onorio De Meo, D. W. Heermann, and K. Binder, "Monte Carlo Study of the Ising Model Phase Transition in Terms of the Percolation Transition of 'Physical Clusters'," *J. Stat. Phys.*, **60**, 585 (1990).

[80] R. G. Palmer, D. L. Stein, E. Abrahams, and P. W. Anderson, "Models of Hierarchically Constrained Dynamics for Glassy Relaxation," *Phys. Rev. Lett.*, **53**, 958 (1984).

[81] J. Pal Singh, J. L. Hennessy, and A. Gupta, "Scaling Parallel Programs for Multiprocessors: Methodology and Examples," *Computer*, **26**, no. 7, 42 (1993).

[82] C. H. Papadimitriou and M. Yannakakis, "Towards an Architecture-Independent Analysis of Parallel Algorithms," *SIAM J. Comput.*, **19**, 323 (1990).

[83] R. Penrose, *The Emperor's New Mind: Concerning Computers, Minds and the Law of Physics*, Oxford Univ. Press (1989).

[84] M. O. Rabin, "Probabilistic Algorithms," in *Algorithms and Complexity*, ed. J. Traub, Academic Press (1976).

[85] R. Rammal, G. Toulouse, and M. A. Virasoro, "Ultrametricity for Physicists," *Rev. Mod. Phys.*, **58**, 765 (1986).

[86] S. Ranka and S. Sahni, "Clustering on a Hypercube Multicomputer," *IEEE Trans. Parallel Distributed Syst.*, **2**, 129 (1991).

[87] T. S. Ray, P. Tamayo, and W. Klein, "Mean-Field Study of the Swendsen-Wang Dynamics," *Phys. Rev. A*, **39**, 5949 (1989).

[88] T. S. Ray and P. Tamayo, "Properties of Metastable Ising Models Evolving under the Swendsen-Wang Dynamics," *J. Stat. Phys.*, **60**, 851 (1990).

[89] K. Rose, E. Gurewitz, and G. C. Fox, "Statistical Mechanics and Phase Transitions in Clustering," *Phys. Rev. Lett.*, **65**, 945 (1990).

[90] P. Rossi and G. P. Tecchiolli, "Finding Clusters in a Parallel Environment," Thinking Machines Corporation preprint, Oct. (1991).

[91] J. K. Salmon, "Parallel Hierarchical N-Body Methods," Ph.D. Dissertation, California Institute of Technology (1991).

[92] R. Schalkoff, *Pattern Recognition: Statistical, Structural, and Neural Approaches*, Wiley (1992).

[93] M. Schreckenberg, "Long Range Diffusion in Ultrametric Spaces," *Z. Phys. B*, **60**, 483 (1985).

[94] C. E. Shannon and W. W. Weaver, *The Mathematical Theory of Communication*, Univ. Ill. Press, Urbana (1949).

[95] Y. Shiloach and U. Vishkin, "An $O(\log n)$ Parallel Connectivity Algorithm," *J. Algorithms*, **3**, 57, 1982.

[96] P. Simic, "Statistical Mechanics as the Underlying Theory of 'Elastic' and 'Neural' Optimizations," *Network*, **1**, 89 (1990).

[97] A. Sloman, "The Emperor's Real Mind," *Artificial Intelligence*, **56**, 355 (1992).

[98] A. D. Sokal, "Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms," Lecture in "Troisième Cycle de la Physique en Suisse Romande" (1989).

[99] D. L. Stein, "Statics and Dynamics of Complex Systems," in *Chance and Matter*, eds. J. Souletie et al. Amsterdam: North-Holland (1987).

[100] G. Stix, "Profile: David A. Huffman, Encoding the 'Neatness' of Ones and Zeros," *Scientific American*, **265**, Sept., 54 (1991).

[101] G. Strang, "Wavelet Transforms versus Fourier Transform," *Bull. Amer. Math. Soc.*, **28**, 288 (1993).

[102] V. Strassen, "Gaussian Elimination is not Optimal," *Numerical Mathematik*, **1**, 354 (1969).

[103] Z. Y. Su, W. Chen, and G. C. Fox, "Implementations of the Hierarchical Cluster Labeling Algorithm on Parallel Computation Environments," to appear in *Proc. 2nd Int. Conf. on Computational Phys. Beijing '93.*

[104] R. Swendsen and J. S. Wang, "Nonuniversal Critical Dynamics in Monte Carlo Simulations," *Phys. Rev. Lett.*, **58**, 86 (1987).

[105] P. Tamayo, R. C. Brower, and W. Klein, "Single-Cluster Monte Carlo Dynamics for the Ising Model," *J. Stat. Phys.*, **58**, 1083 (1990).

[106] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley (1974).

[107] W.-K. Tung, *Group Theory in Physics*, World Scientific (1985).

[108] A. Wagner and D. G. Corneil, "Embedding Trees in a Hypercube is NP-Complete," *SIAM J. Comput.*, **19**, 570 (1990).

[109] S. Wolfram, *Theory and Applications of Cellular Automata*, World Scientific (1986).

[110] U. Wolff, "Comparison between Cluster Monte Carlo Algorithms in the Ising Model," *Phys. Lett.*, **B228**, 379 (1989).

[111] Y. F. Wong, "Clustering Data by Melting," *Neural Computation*, **5**, 89 (1993).

[112] L. G. Valiant, "A Bridging Model for Parallel Computation," *Commun. ACM*, 103 (Aug. 1990).

[113] C. Van Loan, *The Radix-2 Algorithms*, the class release of CS721, Dept. Computer Science, Cornell Univ.

[114] V. S. Vladimirov and I. V. Volovich, "*p*-Adic Quantum Mechanics," *Commun. Math. Phys.*, **123**, 659 (1989).

[115] D. J. A. Welsh, "The Computational Complexity of Some Classical Problems from Statistical Physics," in *Disorder in Physical Systems*, ed. G. R. Grimmett and D. J. A. Welsh, Oxford Univ. Press (1990).

[116] W. H. Zurek, "Algorithmic Randomness and Physical Entropy," *Phys. Rev. A*, **40**, 4731 (1989).