Efficient Parallelization of 2D Ising Spin Systems

Shuangtong Feng
(sfeng@vt.edu)

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Eunice E. Santos, Chair
Lenwood S. Heath
Calvin J. Ribbens

December 11, 2001
Blacksburg, Virginia

Keywords: Ising model, LogP parallel model, data layout optimization, performance prediction, spin selection schemes, computational science, parallel and distributed processing

Efficient Parallelization of 2D Ising Spin Systems

Shuangtong Feng

(Abstract)

The problem of efficient parallelization of 2D Ising spin systems requires realistic algorithmic design and implementation based on an understanding of issues from computer science and statistical physics. In this work, we not only consider fundamental parallel computing issues but also ensure that the major constraints and criteria of 2D Ising spin systems are incorporated into our study. This realism in both parallel computation and statistical physics has rarely been reflected in previous research for this problem.

In this thesis, we designed and implemented a variety of parallel algorithms for both sweep spin selection and random spin selection. We analyzed our parallel algorithms on a portable and general parallel machine model, namely the LogP model. We were able to obtain rigorous theoretical run-times on LogP for all the parallel algorithms. Moreover, a guiding equation was derived for choosing data layouts (blocked vs. stripped) for sweep spin selection. In regards to random spin selection, we were able to develop parallel algorithms with efficient communication schemes. We analyzed randomness of our schemes using statistical methods and provided comparisons between the different schemes. Furthermore, algorithms were implemented and performance data gathered and analyzed in order to determine further design issues and validate theoretical analysis.

Acknowledgements

I would like to thank my committee chair and advisor, Dr. Santos, for her endless help and support for my work. I would also like to thank my committee member Dr. Ribbens and Dr. Heath for their precious time and guidance.

I thank the Laboratory for Computation, Information and Distributed processing (LCID) of the department of Computer Science in Virginia Tech for providing me with the computational resource necessary to carry out the work presented in this thesis.

The results in this thesis are joint work with my advisor, Dr. Santos and Dr. Jeffery M. Rickman. I would also like to thank Dr. Rickman for his guidance.

Lastly, the research in this thesis was supported in part by the National Science Foundation and by Virginia Polytechnic Institute and State University.

Table of Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1 Introduction and Background

1.1 Introduction

Problems in computational science deal with complex phenomena with inherently large problem sizes. As such, parallelization becomes a viable approach in order to obtain results in an efficient manner. Computational scientists have designed a variety of algorithms and software for this purpose. Many of the codes designed by computational scientists have been "home grown codes" which run specifically on certain types of parallel machines.

In recent years, the field of parallel processing has moved towards clustered or distributed message passing platforms and environments. Many older codes are not well suited and at times not truly portable to this new machine paradigm. This leaves many computational scientists with issues of redesigning existing code. Therefore, it is apparent that designing efficient portable parallel code at the onset is of particular importance and spotlights the areas of parallel models and algorithm design.

This exploration of the intersection between computational and computer science brings forth many problems and issues demanding the need for a true understanding of both fields.[1] While the overall goal is particularly large and has fundamental impact, in order to obtain a true understanding of the issues involved, one approach is the in-depth exploration of specific important problems between these fields.

As such, in this thesis, we focus our attention on the parallelization of the 2D Ising model [Ising 1925][Domb et al. 1974] in the area of statistical physics. The Ising model is one of the most widely studied models in statistical physics. Researchers have utilized the Ising model to study the properties and dynamic behaviors of magnetic particles in a magnetic field or heat bath. The model has further been used to study alloys in components [Gonis et al. 1995] and even load-balancing in parallel computing [Bultan 1991].

To promote efficiency and portability to a variety of parallel platforms, a general parallel machine model is utilized. While there has been much research done in general models, cluster computing requires a model that utilizes network parameters for the basis of efficiency analysis.

Of all the well-known and widely accepted models available, the LogP model [Culler et al. 1996] seems particularly well suited to this task. The main emphasis of this thesis is the design, analysis and implementation of parallel algorithms for 2D Ising spin systems on the LogP model.

[1] The contents discussed above were originally provided in [Santos 2001]. As such, please refer to that paper for a more in depth discussion.

In particular, we focus our attention on the following aspects of 2D Ising spin systems:

(i) Our work will design and implement parallel algorithms for 2D Ising spin systems for both blocked and stripped data layouts on a cluster of workstations. Furthermore, we will analyze each algorithm on the LogP model, compare the results, and pinpoint the best data layout, in terms of run-time performance, to use under given network parameter settings. To the best of our knowledge, there has been little done on analyzing such parallel algorithms formally on a practical parallel model (LogP in particular), comparing performance differences of different data layouts, and choosing the best data layout.

(ii) Another important issue of Ising spin systems is the measurement of dynamic properties. To measure dynamic properties, we will need to use a random selection scheme [Binder 1997] that will be introduced in the background section. However, when the random selection scheme is used, it creates many degrees of complexity for parallel implementation on a distributed memory architecture. We will design and implement new parallel algorithms with efficient communication patterns that maintain randomness at high levels. We will introduce a particularly novel random scheme called the α-scheme that maintains much of the needed random characteristics for measuring dynamic properties. Again, we will rigorously analyze all our algorithms on the LogP model and compare their performance.

In addition to the above work on the parallelization of 2D Ising spin systems, we will design and implement algorithms to measure network parameters of the LogP model with the presence of system jitters. This research will further spotlight the validity and usefulness of the LogP model in analyzing and predicting the performance of parallel algorithms, as well as, determining important network characteristics in order to predict network performance and the design of efficient communication schedules. Our approaches to designing efficient parallel algorithms and analyzing them theoretically on the LogP model will be of interest to computer scientists in parallel processing or networking.

The contents of this thesis are as follows. Chapter 2 describes the design and analysis of parallel algorithms for the sweep selection scheme on different data layouts. Chapter 3 presents the design and analysis of parallel algorithms for the random selection scheme. Chapter 4 contains the concluding remarks, a summary of results and a discussion of future work.

The remainder of this chapter is spent covering background materials needed for this thesis.

1.2 Background

In this section, we discuss the 2D Ising model, the LogP parallel model, parallelization methods of the 2D Ising model, and the measurement of network parameters for LogP.

1.2.1 The 2D Ising Spin Model

We start the introduction of the 2D Ising spin model with some basic concepts and definitions.

1.2.1.1 Basics and Concepts

The Ising spin model [Ising 1925] is important in statistical physics and has been widely studied in various areas. Essentially, a 2D Ising spin model represents a set of magnetic spins that are assumed to be positioned on a two dimensional square lattice. These spins are put into a heat bath or an external magnetic field or both. The system then exhibits many interesting and complex properties and phenomena. Intuitively, a spin model consists of [Domb et al. 1974]:

> (a) A *lattice*, i.e. a non-empty set, whose elements are called *lattice sites*, and a binary, irreflexive, symmetric relation (of "bonds") among those sites.
>
> (b) A non-empty subset of lattice sites which are said to be *occupied*. Each such lattice site is *occupied* by a single *spin*.
>
> (c) A non-empty subset of the set of lattice bonds connecting occupied sites; such bonds are said to be *energy* bonds.
>
> (d) A real number associated with each spin whose value may change over time.
>
> (e) A definition of the interaction energy between any two spins in terms of their spin values.
>
> (f) A definition of the total energy of the spin system (the *Hamiltonian*) in terms of these interaction energies.

The 2D Ising spin model is a simplified spin model. It can be intuitively described as the following (Figure 1.1 shows an example): Each lattice site is occupied by a spin. Each spin takes only the values of +1 (up) or –1 (down). Spins $S_{(m,n)}$ and $S_{(x,y)}$ on adjacent lattice sites[2] interact with an energy of $-JS_{(m,n)}S_{(x,y)}$ (this definition of interacting energy is based on the assumption that there is no external magnetic field) and J is a positive real number. Clearly, identical value spins interact with an energy value of $-J$ while opposite value spins interact with an energy value of $+J$. Moreover, we are studying the *ferromagnetic Ising model* in which spins tend to align at the same direction and the system tends towards a lower level of energy. However, when the (heat bath) temperature

[2] Lattice sites *(x, y)* and *(m, n)* are said to be adjacent if and only if the following is true:

$$(x, y) \in \{(m-1, n), (m, n-1), (m+1, n), (m, n+1)\}$$

is above the critical temperature, there is no guarantee that the entire spin systems can stabilize at a certain configuration such as spins pointing either all up or all down.

The magnetization per spin for a system of N spins is defined as $\sum_{ij} \frac{S_{(i,j)}}{N}$, where the summation is over all the spins on the lattice. The *Hamiltonian* is defined as the sum of the interaction energies, i.e.,

$$Hamiltonian = \sum_{(m,n)} \sum_{(x,y)} -JS_{(m,n)}S_{(x,y)} = -J \sum_{(m,n)} \sum_{(x,y)} S_{(m,n)}S_{(x,y)},$$

where $(x, y) \in \{(m-1, n), (m, n-1), (m+1, n), (m, n+1)\}$ and the summation is over all pairs of nearest neighbor spins.

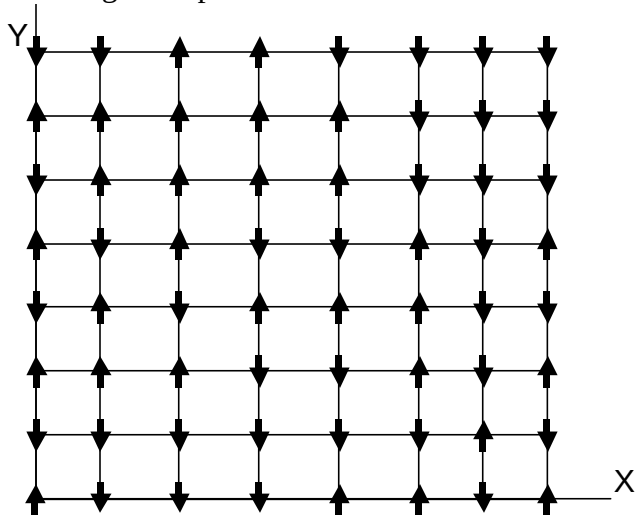


Figure 1.1 (An example of 2D Ising spin model. Spins are represented by up and down arrows)

If the spins are allowed to have $Q \geq 2$ states, this is referred to as the Q-state *Potts model* [Potts 1952]. If the lattice sites are not all occupied by a spin, we call it a *diluted* spin system. In the literature, researchers often refer to a 2D Ising spin system using slightly different terminologies. Examples of these include 2D Ising model, and 2D Ising spin model.

1.2.1.2 Algorithms and Properties

In this study, we focus only on the non-diluted, standard 2-value Ising spin system. The main objectives of studying Ising models are values of internal energy (i.e., the thermal average of the *Hamiltonian*), values of magnetism, critical temperature, correlation between different states and various other dynamic properties. All static values should be obtained when a system reaches equilibrium under a certain temperature and magnetism field setting. For temperatures near zero, all spins should either be aligning up or aligning down after a sufficiently long period of time. For higher temperatures, spins are

disordered and there is no particular common alignment of spins. However, the internal energy (Hamiltonian) or magnetization will still converge to certain values. For a certain temperature T_c, the material begins to lose magnetism as the temperature goes above that point. We refer to T_c as the critical temperature. Theoretical values of the above properties were obtained by Fisher via series expansion [Fisher 1967].

Physicists have studied this model for decades, and have obtained theoretical results for the 2D lattice, but not for the 3D. Researchers have utilized computers to simulate the Ising model and have developed various types of algorithms for such purposes. Furthermore, researchers have developed simulation methods on 2D lattices and then use the theoretical results to validate their simulation methods. Once validated, simulation methods applied on the 2D lattice are then extended to the 3D lattices and above. Thus, computational study on the 2D lattice is an important endeavor in order to understand larger dimensional systems.

One of the most well-known algorithms for simulating Ising models is called the Metropolis Monte Carlo algorithm, introduced in [Metropolis et al.1953]. The pseudo code of the Metropolis algorithm is provided below:

```
Algorithm METROPOLIS MONTE CARLO

Begin

Initialize() //Function definitions can be found at the end of this pseudo code

double Sum_Energy = Comp_Energy()
double Magnet = Comp_Magnet()
double δE = 0
double δM = 0
char S[SIZE][SIZE]        //S denotes the lattice

For iteration=1 to max_iteration DO

   Randomly select one spin from the lattice.

   δE = the value of potential energy change if that spin is flipped.

   IF (δE <= 0) DO
      S(i,j) = -S(i,j)              //accept the flip,
   ELSEIF (rand() <= exp(-δE/Kb*T)) //generate a random number
      S(i,j) = -S(i,j)              //accept the flip,
   ENDIF

   IF (S(i,j) has been changed) DO
      δM=2S(i,j)                    //the change in magnetism
   ENDIF
Sum_Energy = Sum_Energy + δE;

Magnet = Magnet + δM;

ENDFor
```

(Continued from previous page)
END
Function Definitions
rand()
Uniformly randomly generates a double precision number from [0,1].
Initialize()
Initializes the lattice to a random state.
Comp_Energy()
Compute the energy of the current state.
Comp_Magnet()
Compute the magnetism of the current state.

Algorithm 1.1: Sequential MC simulation of 2D Ising model [Metropolis et al. 1953]

Description of Algorithm 1.1 The algorithm works as follows:

1. Initialize the whole lattice to a random initial state.
2. Select a spin to examine, see whether that spin will flip based on the change of energy level.
3. If the energy level decreases, the spin will be flipped immediately. If the energy level increases, the spin will be flipped with probability $P(\delta E) = e^{(-\delta E/K_b T)}$, K_b is called the Bolzman factor (which is a constant). T is called the temperature (which can be expressed by positive real numbers).

1.2.1.3 The Principle of Detailed Balance

In the spin system, there is an important principle called the *principle of detailed balance* which asserts that at the thermodynamic equilibrium, the probability of a system being in any state A and changing to any other state B is equal to the probability of it being in state B and changing to state A. This principle may be stated as:

$$P(A)P(A \to B) = P(B)P(B \to A), \text{ for all states A and B.}$$

A s*tate* is defined as a particular configuration of spins on a lattice.

Moreover, there is another result from statistical physics that provides the probability of a spin system being in any state X at thermodynamic equilibrium in the canonical ensemble(i.e. , constant temperature, volume and particle number).

$$P(X) = \frac{e^{-\beta E_X}}{Z}, \text{ where } Z \text{ is the total energy of all states and } \beta = \frac{1}{K_b T}.$$

From the above two conditions, the following theorem was obtained by Metropolis in [Metropolis et al. 1953].

THEOREM 1. *For the Metropolis algorithm, let δE be the energy change caused be a spin flip. If $\delta E > 0$, the probability of that spin flip being accepted is:*

$$P(\delta E) = e^{-\beta \delta E}, \text{ where } \beta = \frac{1}{K_b T}.$$

PROOF [Metropolis et al. 1953]. Let the state before a certain spin flip be state A and the state after that spin flip be state B. Since $\delta E > 0$, E(A) < E(B). According to the metropolis algorithm, if a state change causes the energy level to decrease, that change will always be accepted. This implies that $P(B \to A) = 1$. So

$$P(B)P(B \to A) = P(B) = \frac{e^{(-\beta E_B)}}{Z},$$

Since $P(A) = \frac{e^{(-\beta E_A)}}{Z}$, according to the detailed balance principle we have

$$P(\delta E) = P(A \to B) = [P(B)P(B \to A)] / P(A) = \frac{e^{-\beta E_B} / Z}{e^{-\beta E_A} / Z} = e^{-\beta(E_B - E_A)} = e^{-\beta \delta E}. \quad \square$$

1.2.1.4 The Periodic Boundary Condition

Another important condition that must be maintained while implementing the Metropolis algorithm is the *periodic boundary condition.* This condition is defined as follows:

The Periodic boundary condition: For an $L \times L$ two-dimensional lattice, grid points (0, j) and (L-1, j) are north-south neighbors and (i, 0) and (i, L-1) are west-east neighbors. Essentially, this defines a wrapped around lattice (See Figure 1.2).

By utilizing a "wrap around configuration", each 2D lattice is essentially mapped to a toriodal lattice, and boundary spin points become no different than interior points. This property is used by researchers to simulate the lattice system without surface effects.

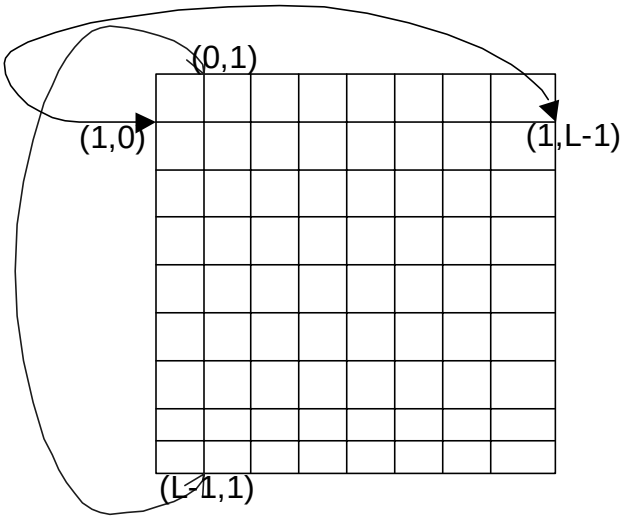


*Figure 1.2 Periodic boundary condition for an L*L two dimensional lattice. According to the periodic boundary condition, (0,1) and (L-1,1) are defined to be neighbors.*

1.2.2 The LogP Parallel Machine Model

The LogP model [Culler et al. 1996] is a general parallel machine model for the design and analysis of portable parallel and distributed algorithms. It is a widely used model for many aspects of parallel processing.

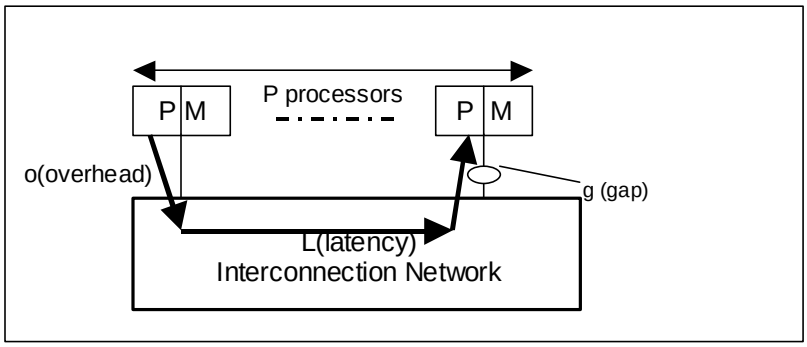
LogP was designed with the goal of being not only general and practical, but also realistic. Earlier models, such as PRAM, specific network topologies, and BSP, unfortunately, did not address all three criteria [Culler et al. 1996]. The PRAM model is too simple and not realistic. It imposes the usage of a central, shared memory and does not consider network topology at all. On the other hand, specific network topologies typically provide realistic representations of particular interconnection networks. However, such representations are not general since they cannot account for portability making them potentially impractical for overall efficient algorithm design. The BSP model does not have sufficient parameters to characterize important network properties. As such, it fails to predict algorithm's performance under a wide range of circumstances. Thus, we decided to base our algorithm design and analysis on the LogP model.

We now briefly describe the LogP model.

In LogP, the letters *L, o, g, and P* represent 4 parameters. These parameters are:

> **L**: the bound on the latency (or delay) incurred in communicating a message
> containing a numerical value from its source module to its target module.
>
> **o**: the overhead, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time, the processor can not perform arithmetic operations.
>
> **g**: the gap, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions to a processor.
>
> **P**: the number of processor/memory modules

Figure 1.3 provides a pictorial overview of the LogP model:



([Culler et al.1996] Figure 1.3 The LogP model describes an abstract machine configuration in terms of four performance parameters: L,o,g and P)

The LogP model assumes a finite number P of processors with local memory connected by a network. It abstracts important criteria of the network topology via the L,o and g parameters. Each processor has its own clock, so synchronization and communication is performed via message passing amongst processors. All send and receive operations are initiated by each processor.

For programmers utilizing clusters of computers, the network has no direct connection to the local memory. There is no global shared memory at all. All communication is done via sending /receiving messages between processors.

L is the delay for sending a unit length message. Here, we consider a byte as our length unit; *o* is the overhead for pushing the message out to the network / pulling the message in from the network. So, the cost for sending one message of 1-byte length is :

$$C(1) = L + o_{send} + o_{recv} = L+2o \ (if\ o_{send} = o_{recv})$$

Let us now consider the cost C(K) of sending a K-byte message on the LogP model. We assume that the underlying network will send messages in a pipelined or overlapped scheme that is widely used in modern communication networks. A pipelined communication scheme sends the (k+1)th byte shortly after it sends out the kth byte, not necessarily needing to wait until the kth byte has reached its destination. The interval between sending the (k+1)th byte and kth byte is defined as g in the LogP model, which essentially can be measured as the per processor bisection bandwidth of the network. The parameter g is a property of the network and it represents the reciprocal of the bandwidth. The faster the network is, the smaller the g is.

Thus, the cost for sending a K-byte message can be summarized as:

$$C(K) = L + 2o + (k-1)g\ ,$$

which can be shown by the following figure:

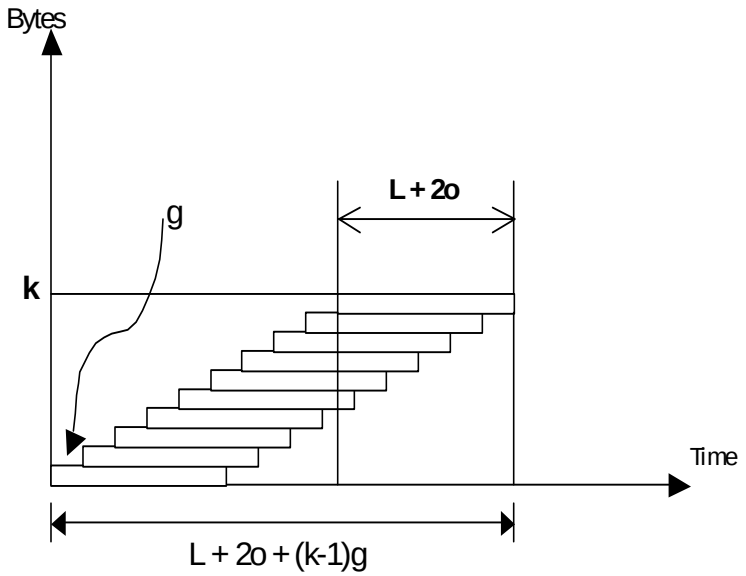


Figure 1.4 Cost of sending a K-byte message on the LogP model

Much research has been published in regards to designing efficient and optimal methods for broadcasting on LogP [Karp et al. 1993][Santos 1999]. These results will provide much needed insights in our design of communication patterns in later chapters.

From the definitions of g and L, we see that there are at most L/g messages in transit from any or to any processors at any time, otherwise the communication will stall[Culler et al. 1996]. This is a particularly important feature of the LogP model and spotlights the practical restrictions of modern communication networks while conveying this restriction to algorithm designers. By doing this, the parallel algorithm will take consideration of the actual network capacity such that communication patterns will not flood the underlying network. In our case, we consider a valid algorithm on LogP to satisfy this capacity constraint and we base our analysis on this. In fact, our algorithms for parallelization of the 2D Ising model have included efforts to control (and fix) the message length such that the underlying network will not be flooded as the lattice size grows. This will be introduced in the next chapter.

As with any parallel models, there are comments and evaluations on the usage of that model in practice. The authors of the LogP paper made several points regarding the utility of LogP. In particular, as specifically stated in [Culler et al. 1996]:

1. Algorithms may adapt their computation and communication structures in response to each of the parameters of the model.

2. In specific situations some parameters become insignificant and one can work with a simplified model (e.g. g=0 or o=0).

3. Adjusting data placement and scheduling communication are important techniques for improving algorithms. (We take careful consideration of this point during the design and implementation of our parallel algorithms for the Ising spin system)

4. LogP can be used not only for algorithm design and analysis but also as a model to determine lower bounds on parallel running time.

5. The LogP model is extremely valuable in guiding algorithm design. Discrepancies between predicted and measured execution time often highlight deficiencies in implementation that violate constraints specified by the model. Correcting these deficiencies resulted in a large performance improvement and led to a close match between predicted and measured execution time.

1.2.3 The Parallelization of 2D Ising Spin Systems

Clearly, there are two types of simple spin flipping schemes: (a) single spin flip or (b) multi-spin flip. Because single spin flipping is not particularly efficient and has difficulty to achieve quick convergence [Wang et al. 1990], researchers have developed various ways to flip multiple spins at the same time. There are mainly two approaches to this. One approach is to modify the Metropolis algorithm to determine a legitimate way to flip spins in groups (i.e. "clusters") instead of flipping spins one by one. This is called *clustering of spins,* and there are two well known methods for this: the Swendsen-Wang algorithm [Wang et al. 1990] and the Wolff algorithm [Wolf 1989]. Another approach is to develop a parallel implementation of the Metropolis algorithm using multiple processors where the lattice is partitioned among processors [Heermann et al. 1991]. This is essentially applying the Metropolis algorithm simultaneously on different parts of the lattice. Our study is focused on the second approach.

Thus, the main goal of this study is to design portable methods for the parallelization of the Metropolis algorithm for the 2D Ising spin system. We base our work on the basic 2D lattice Ising spin model. We believe that our work on this model provides needed insight to more general spin models, be it the 3D Ising model or the Q-state Potts model or other kinds of lattice models.

1.2.3.1 Static and Dynamic Properties

Physicists are particularly interested in two categories of properties of an Ising spin system: (a) Static properties and (b) Dynamic properties. Static properties are typically measured when the system is at the thermodynamic equilibrium state; examples include internal energy per spin, magnetism per spin, critical exponent and critical temperature. Dynamic properties are typically measured throughout the process of state changes. Dynamic properties characterize the behavior of the system over time, e.g., the manner in which it decays from an initial state of magnetization 0. To accurately measure dynamic properties, the system should evolve in such a way that spins are selected randomly, and there should be little to no correlation between spin selections of different time steps in

the process. [Binder et al. 1997] [Binder 1997]. In other words, the more random the spin selection process is, the more accurate the measurement for dynamic properties. Sequential simulations were performed to obtain some of the dynamic properties of the 2D lattice spin systems [Okano et al. 1997]. We note that designing efficient random selection schemes on parallel platforms that can accurately measure dynamic properties is nontrivial.

In this thesis, we design and implement different parallel algorithms to measure both types of properties.

1.2.3.2 Sweep Selection and Random Selection Schemes

At the first step of the Metropolis algorithm, we need to select a spin from the lattice. From there, we make a decision on whether to flip that spin or not based on the potential energy change of such a flip.

There are various types of schemes to select a spin out of a 2D lattice of size $h \times h$. Based on the randomness of the selection scheme, there are 2 extremes: Sweep selection and Random selection.

A) Sweep Selection Scheme

Intuitively, the sweep selection scheme can be described as selecting spins one by one on the lattice from left to right and then from top to bottom. More formally:

Definition1.1. Given a 2D lattice of size $h \times h$, the *sweep selection scheme* selects spin $S(t \bmod h, \lfloor t/h \rfloor)$ at time step t for $0 \le t < h^2$.

Pseudo code for the sweep selection scheme:

```
Algorithm SWEEP SELECTION SCHEME

Begin

I=0, J=0
For I=0 to h-1 DO
    For J=0 to h-1 DO
        select spin at lattice site (I,J)
    ENDFor
ENDFor

END
```

Algorithm 1.2 Sweep Selection Scheme

Sweep selection is both easy to implement and parallelize. We will introduce two different sweep implementations in the next chapter and perform analysis of those implementations on the LogP model. However, we note that sweep selection can**not** be used to measure dynamic properties of the Ising spin system [Binder et al. 1997], thus making random selection necessary.

In the simulation of Ising spin systems, researchers typically define one Monte Carlo Step (MC step for short) as a step in which every spin on the lattice is selected (or given the chance to be selected) once. Obviously, in the case of sweep selection, one MC step consists of one sweep over the whole lattice.

B) Random Selection Scheme

Intuitively, the random selection scheme can be described as selecting spins uniformly randomly on the lattice. One MC step of random selection consists of selecting h^2 spins uniformly randomly over the whole lattice (assuming we have a lattice of size $h \times h$). More formally:

Definition1.2. Given a 2D lattice of size $h \times h$, the *random selection scheme* selects spins uniformly at random on the lattice. One MC step of the random selection scheme consists of h^2 such selections.

Pseudo code of the random selection scheme for one MC step:

```
Algorithm RANDOM SELECTION SCHEME

Begin
Initialize random number generator and give it a seed
While (k < h · h) DO
    Generate a random number R from [0, h · h -1]
    Let I = R / h, J = R % h
    Select spin at lattice site (I,J)
    k = k+1;
ENDWhile
End
```

Algorithm 1.3 Random Selection Scheme

To implement the random selection scheme, we need to utilize a "good" random number generator with a sufficiently long period for large lattices. The random number generator used in our implementation is called the *Mersenne Twister* random number generator [Matsumoto et al. 1998]. It has a very long period of $2^{19937} - 1$, and it has been widely used by researchers and users of simulation programs.

Turning our attention to the parallel implementation of the random selection scheme on a 2D lattice, we see that it is conceivable that this scheme is more difficult to implement than the sweep selection scheme. This is due to the fact that this selection scheme is random and thus has a variable number of spins being selected on borders of the lattice. This can lead to messages of variable lengths being sent between processors, thereby affecting the performance and stability of the parallel algorithm.

We design and implement a method utilizing fixed message lengths in the random selection scheme. This will improve the performance and stability of the parallel algorithm. In Chapter 3, we show by statistical methods that our approach is very close in

randomness to the truly random selection scheme. Clearly, no "non-random" scheme can obtain truly random selection. Full discussion of these topics will be described in Chapter 3.

1.2.3.3 Blocked Domain and Stripped Domain Decompositions

There are two well-known ways to decompose the 2D lattice for parallel processing, blocked decomposition and striped decomposition. They can also be referred to as blocked data layout and stripped data layout, respectively.

Definition1.3. Given a 2D lattice of size $S \times S$ and P processors denoted by P_0, P_1, …, P_{p-1}, a *blocked domain decomposition method*(i.e., *blocked data layout*) is defined as partitioning the lattice into P contiguous sublattices each of size $\frac{S}{\sqrt{P}} \times \frac{S}{\sqrt{P}}$, where processor P_i is assigned the spins at site (j,k) for $(i \bmod \sqrt{P}) \cdot \frac{S}{\sqrt{P}} \le j < (i \bmod \sqrt{P} + 1) \cdot \frac{S}{\sqrt{P}}$, and $(\lfloor i/\sqrt{P} \rfloor) \cdot \frac{S}{\sqrt{P}} \le k < (\lfloor i/\sqrt{P} \rfloor + 1) \cdot \frac{S}{\sqrt{P}}$.
The following figure provides an example of a blocked data layout for 9 processors:

0	1	2
3	4	5
6	7	8

Figure 1.5 Blocked data layout for 9 processors

Definition1.4. Given a 2D lattice of size $S \times S$ and P processors denoted by P_0, P_1, …, P_{p-1}, a *stripped domain decomposition method* (i.e., *stripped data layout*) is defined as partitioning the lattice into P contiguous sublattices each of size $\frac{S}{P} \times S$ (or $S \times \frac{S}{P}$), where processor P_i is assigned the spins at site (j,k) for $0 \le j < S$ (or $i \cdot \frac{S}{P} \le j < (i+1) \cdot \frac{S}{P}$), and $i \cdot \frac{S}{P} \le k < (i+1) \cdot \frac{S}{P}$ (or $0 \le k < S$).
The following figure provides an example of a stripped data layout for 9 processors:

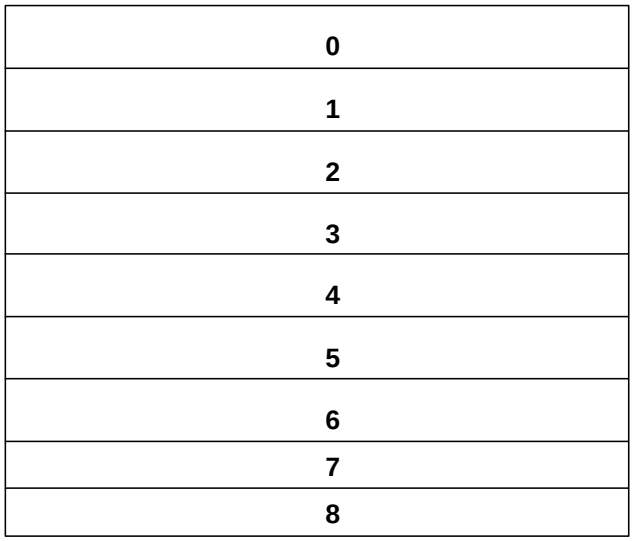


Figure 1.6 Stripped data layout for 9 processors
(Note: It could also be partitioned into vertical strips)

Obviously, the choice of data layouts plays an important role in the design and analysis of parallel algorithms. In previous parallel simulations of Ising spin systems, researchers chose either blocked data layout, or stripped data layout and simply designed and implemented algorithms for that data layout [Heermann et al. 1991].

To the best of our knowledge, there have been no previous efforts in this field to analyze the parallel algorithm formally on a general parallel model (LogP in particular), and then compare performance differences of different data layouts, and finally choose the best data layout for parallel implementation. We will design and implement parallel algorithms for both blocked and stripped data layout on a cluster of workstations, and then analyze both algorithms on the LogP model, compare the results, and pinpoint the best data layout to use under given network parameter settings.

1.2.4 Measuring LogP Model Parameters For a Given Network Cluster

To compare the experimental data with our LogP theoretical analysis, we need to measure the network parameters of our workstation cluster. The parameters of interest are clearly the three network parameters of the LogP model (the *P* is self-explanatory).

L: latency of sending 1 byte from machine A to machine B
o: the overhead of sending/receiving a byte
g: the gap between sending/receiving consecutive bytes.

Moreover, in order to obtain correct measurements of L, o and g, we need to consider the presence of system clock jitters, since different systems of the network cluster usually have different system clocks unless there exists some hardware synchronization mechanisms to ensure consistency among system clocks. Therefore we must also consider:

j: the jitter, which is defined as the system time difference between machine A and machine B.

We will design and implement algorithms to eliminate the system clock jitter and measure L, o and g parameters for our cluster. This will be described in Chapter 2.

In the next chapter, we will present the design and analysis of parallel algorithms for the sweep selection scheme on different data layouts. We will then make comparisons between blocked and stripped data layout and present our results.

Chapter 2 Parallel Algorithms for Sweep Selection Schemes: Design, Implementation and Analysis

In Chapter 1, we provided pertinent background materials regarding the 2D Ising spin system. Spin selection schemes are categorized into two different types: sweep selection and random selection. Each scheme has its importance and, as such, we address both schemes in this thesis. In this chapter, we present algorithm design and analysis for the sweep selection scheme.

2.1 **Algorithm Design For Sweep Schemes**

As introduced in Chapter 1 (Definition 1.1 and Algorithm 1.2), a sweep selection scheme can be described in words as selections of spins one by one, from left to right, and then from top to bottom. Sweep selection schemes can be applied on any kind of data layout, either blocked or stripped. This chapter presents the design of parallel algorithms for sweep schemes on both data layouts.

In [Heerman 1991], brief descriptions of parallel algorithms are provided for both data layouts. However, these algorithms are not presented in detail. Furthermore, no theoretical performance analysis of these algorithms is provided, and only experimental results on Transputers are given.

In this chapter, we provide clear descriptions and full pseudo codes for both algorithms, along with implementation details. More importantly, we provide rigorous theoretical analysis of both algorithms on a portable, practical parallel model, i.e. LogP, and establish a threshold value on lattice size in order to determine when blocked data layout outperforms stripped data layout.

An important condition we must satisfy throughout the whole algorithm is the *Detailed Balance Condition.*

As introduced previously, the detailed balance condition can be described as following: $P(A)P(A \to B) = P(B)P(B \to A)$ for all states A and B under thermodynamic equilibrium, i.e. the probability of a system being in any state A and changing to any other state B is equal to the probability of it being in state B and changing to state A.

In terms of multiple simultaneous spin flips using Metropolis Monte Carlo dynamics, the detailed balance condition implies that neighboring spins should never be allowed to be flipped at the same time. This must be kept in mind for designing any parallel schemes for the Metropolis Monte Carlo algorithm.

Clearly, processors need to avoid flipping neighboring boundary points simultaneously. One widely used solution is to split one iteration step into 2 stages. At the end of each stage, communication and synchronization will be performed among neighboring

processors. At the beginning of each stage, the data necessary for executing that stage needs to be available for that processor.

In each data layout, each processor receives additional data from neighboring processors. Researchers call them “cached boundary data” which essentially holds the boundary data from neighboring processors. Such data are necessary for local processors to compute the potential energy change caused by flipping a spin on the boundary. Because of the detailed balance condition, once a processor P_i sends out its boundary data to its neighbors, P_i cannot further update any boundary spins, unless its neighbors are no longer using the boundary data from it. This leads to the necessity for synchronization and communication.

The following sections describe in detail the algorithm design and analysis for both blocked data layout and stripped data layout.

2.2 Blocked Data Layout

In this section, we introduce the design, pseudo code and analysis of parallel algorithms on the blocked data layout.

2.2.1 Description of Blocked Data Layout

We begin by providing a figure containing an example of blocked data layout

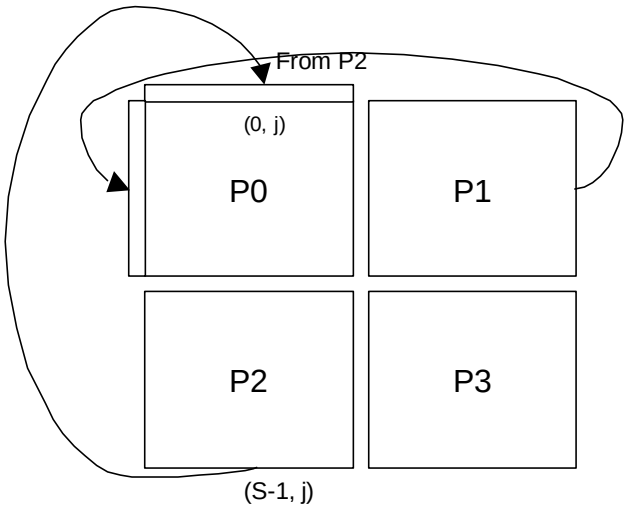


Figure 2.1 Blocked Data layout for 4 processors

Assuming we have 4 processors and a lattice size of $S \times S$, we now describe our parallel algorithm and then present a generalized pseudo-code.

In our design, processor P_0 is assigned a sub block containing rows 0 to h-1 and columns 0 to h-1, where h = S/2. In order to flip a spin at site (0, j), it needs to know the value of 4 neighbor spins, which are (0,j-1), (1,j), (0,j+1) and (S-1,j). The first three lattice sites are

in processor P_0, but the last one, (S-1,j), is in processor P_2, clearly communication between P_0 and P_2 is needed.

One method of handling such communication in a general and structured fashion is to allow each processor to send its 4 boundaries to its neighbors at the start of each iteration and meanwhile, receive boundary data from its neighbors. The received data will be stored locally at each processor in 4 arrays and used throughout the iteration. We call these 4 arrays *boundary arrays* and they are named the *top boundary array*, the *left boundary array*, the *bottom boundary array* and the *right boundary array*.

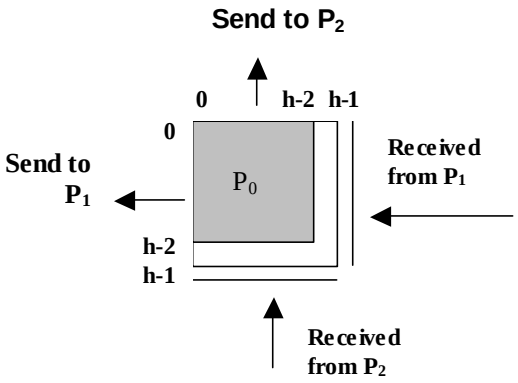
However, what if P_0 and P_2 want to update spins (0, j) and (S-1, j) at the same time? This is not allowed due to the detailed balance condition (neighboring spins cannot be flipped at the same time). Clearly, we need to synchronize these updates!

The solution is to split a Monte Carlo step into 3 stages:

> At stage 1, processor P_0 locally updates rows 0 to h-2 and columns 0 to h-2. Furthermore, it sends spins of row 0 and column 0 to its upper and left neighbors respectively while receiving spins of row h-1 and column h-1 from its lower neighbor (P_2) and right neighbor (P_1) respectively.
>
> At stage 2, processor P_0 locally updates two borders: the bottom border and the right border (excluding the lower-right corner), which is row {(h-1, 0), (h-1, h-2)} and column {(0,h-1), (h-2, h-1)}. It will send spins of these two borders to its lower and right neighbors respectively. Furthermore, it needs to send spin (h-1, 0) to its left neighbor and spin (0,h-1) to its upper neighbor. At the same time, it also receives spins from its neighbors.
>
> At stage 3, processor P_0 locally updates the spin at lattice site (h-1,h-1) which is the lower right corner, and then send that spin to P_0's right neighbor and lower neighbor.

We now provide figures representing these three stages. We note that the process is symmetric in regards to processors. Thus, our figures focus on processor P_0.

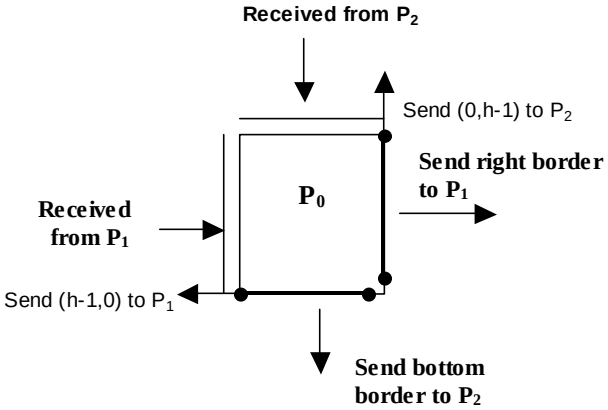
Stage 1



The shaded area represents the region in which P_0 executes its local spin update operations. {(0,0), (h-2,h-2)}

Figure 2.2 Stage 1 of sweep selection using blocked data layout

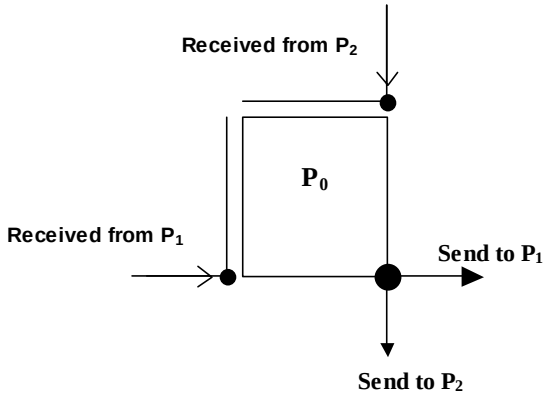
Stage 2



P_0 executes its local spin update operations on two borders: the bottom border and the right border

Figure 2.3 Stage 2 of sweep selection using blocked data layout.

Stage 3



P_0 executes its local spin update operations on the lower right corner (h-1,h-1).

Figure 2.4 Stage 3 of sweep selection using blocked data layout.

An efficient yet fairly straightforward parallel algorithm based on a blocked data layout sweep scheme is presented below:[3]

2.2.2 Algorithm Pseudo Code For Blocked Sweep Scheme

Algorithm BLOCKED SWEEP SELECTION SCHEME

INPUT: A lattice of size $S \times S$ **,** and *P* processors. *S* must be divisible by $\sqrt{P}$.
OUTPUT: Static properties of the Ising spin system, e.g. internal energy, magnetism.

Begin

/*
Initialization: Each processor initializes its block of the sub-lattice and send/receive boundaries from its neighbors. Each processor has a sub-lattice of size h*h where h equals to (S / $\sqrt{P}$), S is the width (height) of the lattice, and P is the number of processors available.
*/

For each processor P_i **DO**

Initialize (); //function definitions are given at the end of this pseudo code

ENDFor

// Now we begin the Monte Carlo process

[3] As stated previously, a brief discussion of this method was provided in [Heermann et al. 1991]. However, no design specifications were provided. This left the reader to determine how to best design and implement this method. In this thesis, we have developed a full design specification that provides overall efficiency (i.e. both computational and communication efficiency) spanning wide classes of parallel machines.

(Continued from previous page)

For each processor P_i **DO**

K = 0;

While K < (Total MC steps) **DO**

Begin

//Stage 1
/*
Pi sweeps over sub lattice (0,0) – (h-2,h-2), select spins one by one from left to right and up to down. Compute potential energy change for each selection and decide whether to flip it or not according to the Metropolis algorithm.
*/

Sweep((0,0) , (h-2,h-2));

/*
P_i sends row 0 and column 0 to its upper neighbor and left neighbor, respectively.
*/

Send (row_0, upper);
Send (column_0, left);

/*
P_i receives spins from its right neighbor and lower neighbor, and put them into right boundary array and bottom boundary array respectively.
*/

Receive(right, right_array);
Receive(lower, bottom_array);

//Stage 2
/*
Pi sweeps over the bottom row from (h-1,0) to (h-1,h-2), and sweeps over the right column from (0,h-1) to (h-2,h-1). For each spin it selects, compute potential energy change and decide whether to flip it or not according to the Metropolis algorithm.
*/

Sweep((h-1,0), (h-1,h-2));

/*
P_i sends bottom row and right column to its lower neighbor and right neighbor
*/

Send (row_h-1, lower);
Send (column_h-1, right);

/*
P_i sends spin (h-1,0) to its left neighbor, and spin(0,h-1) to its upper neighbor
*/

Send ((h-1,0), left);
Send ((0,h-1), upper);

(Continued from previous page)

```
/*
Pi receives spins from its left neighbor and upper neighbor, and put them into the left boundary
array and top boundary array respectively.

Pi receives one spin from its right neighbor and put it into the (h-1)th position of its right boundary
array, and one spin from its lower neighbor and put it into the (h-1)th position of its bottom
boundary array
*/

Receive( left, left_array );
Receive( upper, top_array );

Receive( right, right_array[h-1] );
Receive( lower, bottom_array[h-1] );

//Stage 3
/*
Pi selects spin at site (h-1,h-1), compute potential energy change and decide whether to flip it or
not according to the Metropolis algorithm.
*/

Sweep( (h-1,h-1), (h-1,h-1) );

/*
Pi sends spin of (h-1,h-1) to its right neighbor and lower neighbor
*/

Send ( (h-1,h-1), right );
Send ( (h-1,h-1), lower );

/*
Pi receives one spin from its upper neighbor and put it in the (h-1)th position of the top boundary
array, and receives one spin from its left neighbor and put it in the (h-1)th position of the left
boundary array.
*/

Receive( upper, top_array[h-1] );
Receive( left, left_array[h-1] );

K=K+1;

IF (K>= WARM_UP && K % SAMPLE_INTERVAL == 0) DO
        // Run functions to measure all desired physics properties.
        Compute_Energy();
        Compute_Magnetism();
ENDIF

ENDWhile

ENDFor

ENDAlgorithm
```

(Continued from previous page)

Function Definitions:

Initialize()
Initialize the local lattice. Each spin will be randomly assigned a value of +1 or –1.

Sweep(From, To)
Apply sweep selection scheme of Algorithm 1.2 on the rectangle specified by upper-left point "From" and lower-right point "To", apply Metropolis Monte Carlo Algorithm (algorithm 1.1) on each selected spin.

Send (DATA , DESTINATION)
Send the data buffer "DATA" to the processor specified by "DESTINATION"

Receive(SOURCE, BUFFER)
Receive data from the processor specified by "SOURCE" and store the data in "BUFFER"

Compute_Energy(), Compute_Magnetism()
Computes the internal energy and magnetism of the lattice.

Algorithm 2.1:Blocked Sweep Selection Scheme

2.2.3 Analysis of Algorithm 2.1

In this section, we analyze the cost of Algorithm 2.1.We provide both computational and communication costs of the algorithm.

LEMMA 2.1. *Given P processors and assuming a lattice size of* $S \times S$ *, the computation cost of one Monte Carlo step of the sweep selection scheme using blocked data layout is*

$$T_{comp} = \frac{CS^2}{P},\ C > 0 \text{ is a constant.}$$

PROOF. For one Monte Carlo step, there is a total of h^2 selections and updates of spins on each processor, where *h* is the size of the local block. Let the total cost of selecting a spin, computing the potential energy change, and generating a random number be C. Clearly, C is a positive constant for any given processor. Thus, the total computation cost of one Monte Carlo step is:

$$T_{comp} = C \cdot h^2 = C \cdot \left(\frac{S}{\sqrt{P}} \right)^2 = \frac{CS^2}{P}.$$

LEMMA 2.2. *Assume the size of a sublattice assigned to each processor is* $h \times h$ *, the communication cost of one Monte Carlo step of the sweep selection scheme using blocked data layout is*

$$T_{communication} = 8(L + 2o) + 4g(h - 2),$$

where L, o, and g are the parameters of the LogP model.

PROOF. As introduced before, sending one k-byte message on LogP has a cost of $L+2o+(k-1)g$. In our case, we observe the following facts:

Stage 1: Sending 2 messages each of size h-1 costs:

$$T_1 = 2(L + 2o + (h-2)g).$$

Stage 2: Sending 2 messages each of size h-1 followed by 2 messages each of size 1 costs:

$$\begin{aligned} T_2 &= 2(L + 2o + (h-2)g) + 2(L + 2o) \\ &= 4(L + 2o) + 2(h-2)g. \end{aligned}$$

Stage 3: Sending 2 messages each of size 1 costs:

$$T_3 = 2(L + 2o).$$

Thus, the total cost of communication is:

$$\begin{aligned} T_{communication} &= T_1 + T_2 + T_3 \\ &= 4(L + 2o + (h-2)g) + 4(L + 2o) \\ &= 8(L + 2o) + 4(h-2)g. \end{aligned}$$

THEOREM 2.3. *Given P processors and assuming a lattice size of $S \times S$, let h be equal to $\frac{S}{\sqrt{P}}$, the total run-time cost of one Monte Carlo step of the sweep selection scheme using blocked data layout is*

$$T(B) = \frac{CS^2}{P} + 8(L + 2o) + 4g(h-2),$$

where L, o, and g are the parameters of the LogP model, and C > 0 is a constant.

PROOF. The total cost of the sweep selection scheme using blocked data layout is simply the summation of the computation cost and the communication cost. Let *T(B)* denote the total cost. From Lemma 2.1 and Lemma 2.2, we obtain the following result

$$T(B) = T_{comp} + T_{communication}$$

$$= \frac{CS^2}{P} + 8(L + 2o) + 4g(h-2).$$

In the following section we will discuss algorithms utilizing stripped data layout. Once run-time has been analyzed utilizing stripped data layout, we will compare them with the run-time for blocked data layout.

2.3 Stripped Data Layout

In this section, we provide the detailed design, pseudo code, and analysis of parallel algorithms for the stripped data layout.

2.3.1 Description of Stripped Data Layout

The design of the sweep scheme on stripped data layout is similar to that of blocked data layout, except that each processor has only 2 neighbors. This slightly simplifies the problem. The following figure provides an example of a stripped data layout along with how processor P_i stores boundary information from its neighbors.

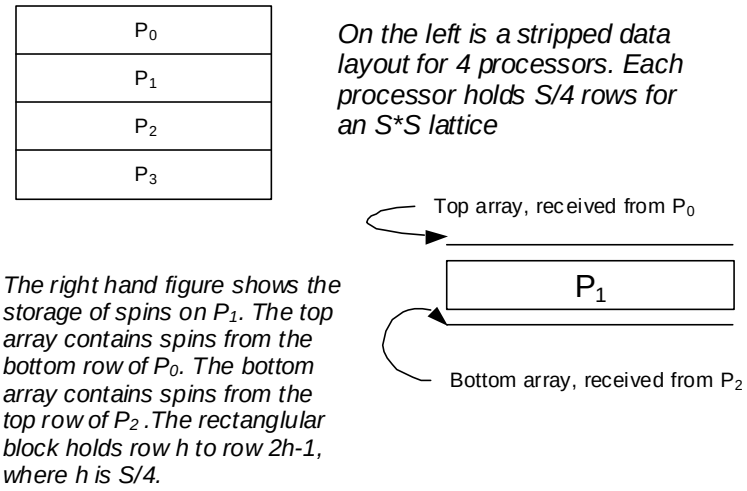


Figure 2.5 Stripped Data Layout

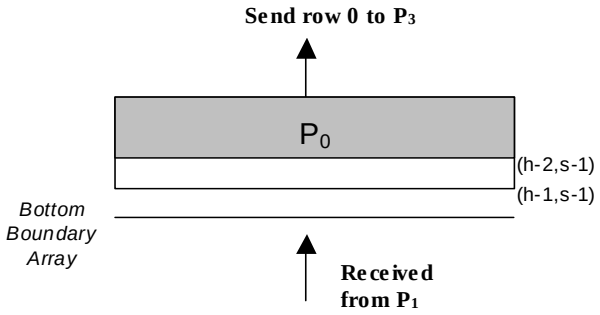
We note that we also need to satisfy the detailed balance condition as before. We divide one MC step into stages. For the stripped data layout, we have 2 stages (WLOG, we explain these 2 stages on P_0. Clearly, each processor performs the same procedure as P_0):

> At stage 1, processor P_0 locally updates rows 0 to (h-2) in its own block. It then sweeps from left to right and then up to down. After local updates, it will send spins of row 0 to its upper neighbor (P_3), while receiving spins from its lower neighbor (P_1), and stores the received spin into the bottom boundary array.

At stage 2, processor P_0 locally updates the bottom row (row h-1) and sweeps from left to right. After local updates, it sends the bottom row to its lower neighbor, and at the same time, receives spins from its upper neighbor (P_3), and stores the received spin in the top boundary array.

The following figures pictorially represent these two stages:

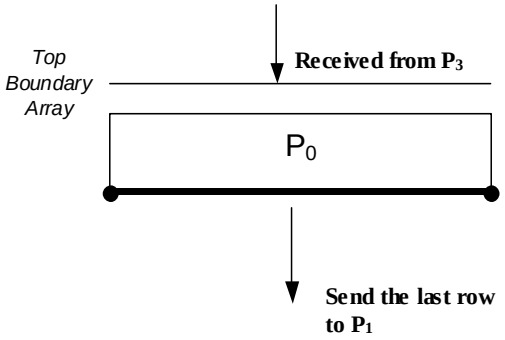
Stage 1



The shaded area represents the region in which P_0 executes its local spin update operations. {(0,0), (h-2 , S-1)}

Figure 2.6 Stage1 of sweep selection using stripped data layout.

Stage 2



P_0 executes its local spin update operations on the last row (the bolded line). {(h-1,0), (h-1,S-1)}

Figure 2.7 Stage 2 of sweep seleciton using stripped data layout

We now present the pseudo-code for stripped data layout

2.3.2 Algorithm Pseudo Code For Stripped Sweep Scheme

Algorithm STRIPPED SWEEP SELECTION SCHEME

INPUT: A lattice of size $S \times S$ **,** and *P* processors. *S* must be divisible by *P*.
OUTPUT: Static properties of the Ising spin system, e.g. internal energy, magnetism.

Begin

/*
Initialization: Each processor initializes its strip of the sublattice, sends and receives boundaries from its neighbors. Each processor has a sublattice of size $S \times h$ where *h* equals to S/P , *S* is the width (height) of the lattice and P is the number of processors.
*/

For each processor P_i **DO**

Initialize (); //Function definitions can be found at the end of this pseudo code

ENDFor

// Now we begin the Monte Carlo process

For each processor P_i **DO**

K = 0;

While K < (Total MC steps) **DO**

Begin

//Stage 1
/*
Pi sweeps over sublattice *(0,0) – (h-2,S)*, select spins one by one from left to right and up to down. Compute potential energy change for each selection and decide whether to flip it or not according to the Metropolis algorithm.
*/

Sweep((0,0) , (h-2,S));

/*
P_i sends row 0 to its upper neighbor.
P_i receives spins from its lower neighbor, and put them into bottom boundary array.
*/

Send (row_0, upper);
Receive (lower, bottom_array);

//Stage 2
/*
Pi sweeps over the bottom row from (h-1,0) to (h-1,S-1).For each spin it selects, compute potential energy change and decide whether to flip it or not according to the Metropolis algorithm.
*/

(Continued from previous page)

Sweep((h-1,0) , (h-1,S-1));

/*
P_i sends bottom row to its lower neighbor
P_i receives spins from its upper neighbor, and put them into top boundary array.
*/

Send (row_h, lower);
Receive (upper, top_array);

K=K+1

If (K>= WARM_UP && K % SAMPLE_INTERVAL == 0)
// Run functions to measure all desired physics properties.
Compute_Energy();
Compute_Magnetism();

ENDWhile

ENDFor

ENDAlgorithm

Function Definitions:

Initialize()
Initialize the local lattice. Each spin will be randomly assigned a value of +1 or –1.

Sweep(From, To)
Apply sweep selection scheme of Algorithm 1.2 on the rectangle specified by upper-left point "From" and lower-right point "To", apply Metropolis Monte Carlo Algorithm (Algorithm 1.1) on each selected spin.

Send (DATA , DESTINATION)
Send the data buffer "DATA" to the processor specified by "DESTINATION"

Receive(SOURCE, BUFFER)
Receive data from the processor specified by "SOURCE" and store the data in "BUFFER"

Compute_Energy(), Compute_Magnetism()
Computes the internal energy and magnetism of the lattice.

Algorithm 2.2 Stripped Sweep Selection Scheme

2.3.3 Analysis of Algorithm 2.2

LEMMA 2.4. *Given P processors and assuming a lattice size of $S \times S$, the computation cost of one Monte Carlo step of the sweep selection scheme using stripped data layout is*

$$T_{comp} = \frac{CS^2}{P},\ C > 0 \text{ is a constant.}$$

PROOF. For one Monte Carlo step, there is a total of $S \cdot h$ selections and updates of spins on each processor, where *h* is the height of the local strip. Let the total cost of selecting a spin, computing the potential energy change, and generating a random number be C. C is a positive constant for any given processor. Thus, the total computation cost of one Monte Carlo step is:

$$T_{comp} = C \cdot S \cdot h = C \cdot S \cdot \frac{S}{P} = \frac{CS^2}{P}.$$

LEMMA 2.5. *Assume the size of a sublattice assigned to each processor is $S \times h$, the communication cost of one Monte Carlo step of the sweep selection scheme using stripped data layout is*

$$T_{communication} = 2(L + 2o) + 2g(S - 1),$$

where L, o and g are the parameters of the LogP model.

PROOF. Similar to the proof of Lemma 2.2, we analyze the stripped sweep selection scheme by the following two stages,

Stage 1: Sending 1 message of size *S* costs:

$$T_1 = L + 2o + (S - 1)g.$$

Stage 2: Sending 1 message of size S (same as Stage 1) costs:

$$T_2 = L + 2o + (S - 1)g.$$

Thus, the total cost of communication is:

$$\begin{aligned} T_{communication} &= T_1 + T_2 \\ &= 2(L + 2o + (S - 1)g) \\ &= 2(L + 2o) + 2g(S - 1). \end{aligned}$$

THEOREM 2.6. *Given P processors and assuming a lattice size of $S \times S$, the total run-time cost of one Monte Carlo step of the sweep selection scheme using stripped data layout is*

$$T(S) = \frac{CS^2}{P} + 2(L + 2o) + 2g(S - 1),$$

where L, o, and g are the parameters of the LogP model and C > 0 is a constant.

PROOF. The total cost of the sweep selection scheme using stripped data layout is simply the summation of the computation cost and the communication cost. Let *T(S)* denote the total cost. From Lemma 2.4 and Lemma 2.5, we obtain the following result

$$\begin{aligned} T(S) &= T_{comp} + T_{communication} \\ &= \frac{CS^2}{P} + 2(L+2o) + 2g(S-1). \end{aligned}$$

2.4 Comparison of Blocked Data Layout and Stripped Data Layout

LEMMA 2.7. *Given P processors and assuming a lattice size of* $S \times S$, $T(B) \geq T(S)$ *if and only if*

$$\frac{L+2o}{g} \geq \left(1 - \frac{2}{\sqrt{P}}\right)\frac{S}{3} + 1,$$

where L, o, and g are the parameters of the LogP model.

PROOF. By Theorem 2.3 and Theorem 2.6, we obtain the following equation:

$$\begin{aligned} T(B) - T(S) &= \frac{CS^2}{P} + 8(L+2o) + 4g\left(\frac{S}{\sqrt{P}} - 2\right) - \frac{CS^2}{P} - 2(L+2o) - 2g(S-1) \\ &= 6(L+2o) + g\left(\frac{4S}{\sqrt{P}} - 6 - 2S\right). \end{aligned}$$

Clearly, $T(B) \geq T(S)$ if and only if $T(B) - T(S) \geq 0$. Thus,

$$6(L+2o) + g\left(\frac{4S}{\sqrt{P}} - 6 - 2S\right) \geq 0$$

$$\Leftrightarrow \qquad \frac{L+2o}{g} \geq \left(1 - \frac{2}{\sqrt{P}}\right)\frac{S}{3} + 1$$

The implications of Lemma 2.7 are as follows:

- If $\frac{L+2o}{g} \geq \left(1 - \frac{2}{\sqrt{P}}\right)\frac{S}{3} + 1$, stripped data layout should be utilized over blocked data layout.

- If $\frac{L+2o}{g} < \left(1-\frac{2}{\sqrt{P}}\right)\frac{S}{3}+1$, blocked data layout should be utilized over stripped data layout.

THEOREM 2.8. *Given P processors and assuming a lattice size of $S \times S$. Let L, o, and g be the parameters of the LogP model.*

$$\textit{If } P = 4,\ T(B) > T(S),$$

$$\textit{If } P > 4,\ T(B) \geq T(S) \textit{ if and only if}$$

$$S \leq \frac{3\sqrt{P}(L+2o-g)}{(\sqrt{P}-2)g}.$$

PROOF. By Lemma 2.7, $T(B) \geq T(S)$ if and only if

$$\frac{L+2o}{g} \geq \left(1-\frac{2}{\sqrt{P}}\right)\frac{S}{3}+1.$$

For a given network cluster, the parameter set of {L, o, g} is assumed to be fixed (stable). So, the ratio at the left hand side of the inequality is fixed and greater than 0.

Consider the right hand side of the inequality (we use RHS to denote the right hand side and LHS to denote the left hand side):

If $P = 4$, RHS = $\left(1-\frac{2}{\sqrt{4}}\right)\frac{S}{3}+1 = 0 \Rightarrow$ LHS > RHS. Thus from Lemma 2.7, *T(B)>T(S).*

If $P > 4$, RHS = $\left(1-\frac{2}{\sqrt{P}}\right)\frac{S}{3}+1$, i.e., $T(B) \geq T(S)$ if and only if

$$\frac{L+2o}{g} \geq \left(1-\frac{2}{\sqrt{P}}\right)\frac{S}{3}+1$$

$$\Leftrightarrow \quad S \leq \frac{3\sqrt{P}(L+2o-g)}{(\sqrt{P}-2)g}.$$

To summarize Theorem 2.8, we point out the following important results:

- For $P = 4$, blocked data layout is slower than stripped data layout. Clearly, choose stripped data layout.

- For $P > 4$, when the lattice size S is less than or equal to $\frac{3\sqrt{P}(L+2o-g)}{(\sqrt{P}-2)g}$, blocked data layout is slower than stripped data layout, otherwise the reverse is true.

The following table illustrates some sample threshold values of S for P = 9, 16, 25. When S is less than the threshold value, blocked is slower than stripped data layout, otherwise blocked is faster than stripped data layout.

Processor Number	Threshold Value of S
9	$\frac{9(L+2o-g)}{g}$
16	$\frac{6(L+2o-g)}{g}$
25	$\frac{5(L+2o-g)}{g}$

Table 2.1 Sample Threshold Values of S for P = 9, 16, 25

Once we are given a network cluster, we can measure L, o and g. A simple method to measure these parameters will be introduced in the following section. After we have obtained values for all these parameters, we can decide which data layout to use for a certain pair of S and P.

2.5 Measuring the L, o, and g Parameters of the LogP Model

We note that there have been some previous research studies performed on the measurement of parameters for the LogP model [Kielmann et al. 2000].

Here, we present a simple method to measure latency (L), overhead (o) and gap (g) for a given cluster of workstations. Our method assumes a homogeneous and single level network. We allow workstations to have different system clocks, and we will eliminate the difference in our method.

To measure these parameters, we need to send and receive messages in between workstations. Let us consider two nodes: A and B. Assuming that A and B have asynchronous system clocks, i.e. at the same physical time t, node A has system time of T_A but B has a different system time of T_B. We define the jitter j as $T_B - T_A$.

Let T_1 be the time to send one double number (8-byte) from A to B, and T_1 is measured by node B, then

$$T_1 = L + 2o + 7g + j \qquad (1)$$

Let T_2 be the time to send K double numbers from A to B (K can be chosen arbitrarily as long as there is a clear difference between T_2 and T_1), then

$$T_2 = L + 2o + (8K-1)g + j \qquad (2)$$

Let T_3 be the time to send one double numbers from B to A, and T_3 is measured by node A, then

$$T_3 = L + 2o + 7g - j \qquad (3)$$

Each T_i should be measured by repeating the same experiment multiple times, discard the first 2 measurements (thus eliminating the warm-up time for a network), and finally average over the remaining measurements.

By solving equations (1), (2), and (3) we obtain:

$$j = (T_1 - T_3)/2,$$

$$g = (T_2 - T_1)/8(K-1),$$

$$L + 2o = T_1 - \left(\frac{7}{8(K-1)}\right)(T_2 - T_1) - \frac{(T_1 - T_3)}{2}.$$

Once K is chosen, we are able to obtain the numerical values for j, g and $L+2o$.

The overhead, o, can be measured by taking 2 timestamps before and after MPI_Send() and computing the difference between those time stamps. In order to achieve a more accurate measurement of o, we should measure o on both machines A and B and then take the average as the final o.

After o is measured, we can determine L.

If using MPI, all time values can be accurately measured by calling the MPI_Wtime() routine on the local machine. To measure T_1, we let machine A get a local time stamp T_A, (a double precision number) and then send that number to machine B. On B's side, upon receiving a number from A, it takes a time stamp T_B. T_1 is the result of subtracting T_A from T_B.

The following pseudo code shows how to measure T_1:

Algorithm MEASURING T1

INPUT: 2 Processors and the network in between them.
OUTPUT: T_1 the cost of sending a double precision number.

Begin

rank = the rank of local processor.

SOURCE = a constant indicating the rank of the source processor.

DESTINATION = a constant indicating the rank of the destination processor.

IF (rank == SOURCE){

numbers = MPI_Wtime();
Send (numbers, DESTINATION);

}
ELSEIF (rank == DESTINATION)
{

Receive(SOURCE, numbers)
end=MPI_Wtime();
T_1 = end - numbers;
Output(T_1);

}

ENDIF

END

Function Definitions:

MPI_Wtime()
Get the system time from the local processor.

Send (DATA , DESTINATION)
Send the data buffer "DATA" to the processor specified by "DESTINATION"

Receive(SOURCE, BUFFER)
Receive data from the processor specified by "SOURCE" and store the data in "BUFFER"

Remarks
This algorithm can be changed to measure T_2 and T_3 also -- only need to modify the message length or change SOURCE and DESTINATION value.

Algorithm 2.3 Measuring T_1 -- The cost of sending a double precision number

2.6 Experimental Data and Plots

In this section, we present the experimental data for the parallel algorithms described in previous sections along with corresponding plots. We ran all our simulations on a 16-node Solaris Ultra 5 workstation cluster with a 100Base-T Ethernet backbone. We also used the MPICH implementation [Gropp et al. 1996][Gropp et al. 1996a] of the MPI standard as the message passing library.

2.6.1 Measuring L, o, and g Values of the Experimental Environment

We measured the latency, overhead and gap of our cluster using the algorithm introduced in the previous section. We ran the measuring program 10 times and took the average of the outcomes. The measured values are listed in the following table.

Experiment	L+2o	L	o	g
1	234	186	24	1
2	242	194	24	1
3	205	159	23	1
4	246	198	24	1
5	241	191	25	1
6	251	203	24	1
7	245	197	24	1
8	234	186	24	1
9	248	200	24	1
10	249	201	24	1
High	**251.0**	**203.0**	**25.0**	**1.0**
Low	**205.0**	**159.0**	**23.0**	**1.0**
Std Dev	**12.75**	**12.21**	**0.45**	**0.00**
Average	**239.5**	**191.5**	**24.0**	**1.0**

Table 2.2 Measured L, o, g values and the statistics (in μs)

We summarize the above table by highlighting the following average values of our network properties (in μs):

Latency:	191.5
Overhead:	24.0
Gap:	1.0
Latency + 2*Overhead:	239.5

The jitter is also measured and it is: -0.186718 seconds.

2.6.2 Performance Data of Blocked and Stripped Data Layouts

For the purpose of obtaining algorithm performance data, we run each simulation for 20 MC steps. We note that researchers (e.g. statistical physicists) who are interested in obtaining precise Ising spin system data can change the number of MC steps of each simulation through command line input.[4]

According to the theoretical analysis given above, we predicted that when $S \le \frac{3\sqrt{P}(L+2o-g)}{(\sqrt{P}-2)g}$ and P > 4, blocked data layout will be slower than stripped data layout. When $S > \frac{3\sqrt{P}(L+2o-g)}{(\sqrt{P}-2)g}$ and P > 4, blocked data layout will be faster than stripped data layout. Finally, for P = 4, regardless of the size of S, blocked data layout is always slower than stripped data layout.

Now, when we plug in the values we measured for *L*, *o* and *g*, we are able to obtain the following real value prediction about the critical lattice size S_c.

For P=9, $S_c = \frac{3\sqrt{P}(L+2o-g)}{(\sqrt{P}-2)g} = \frac{3\times 3\times(239.5-1)}{3-2} = 9\times 239.5 = 2155.5$

For P=16, $S_c = \frac{3\sqrt{P}(L+2o-g)}{(\sqrt{P}-2)g} = \frac{3\times 4\times(239.5-1)}{4-2} = 6\times 239.5 = 1437$

So, for $S < S_c$, blocked data layout is slower than stripped data layout ($T(B) - T(S) > 0$); for $S > S_c$, blocked data layout is faster than stripped data layout.($T(B) - T(S) < 0$).

The following tables and graphs show the running time results for 4, 9 and 16 processors. We note that although there are some random variations of the algorithm run-time due to occasional network fluctuations, the theoretical results still clearly predicted the outcome of the experimental results.

Lattice Size	Blocked	Stripped	T(B)-T(S)
720	5.50	5.49	0.01
1440	23.70	22.04	1.65
2160	53.07	49.35	3.72
2880	86.22	85.57	0.65
3600	137.67	135.67	2.00
5400	309.46	305.14	4.31

Table 2.3 Results for 4 processors (in seconds)

[4] Source code is available by emailing the author of this thesis.

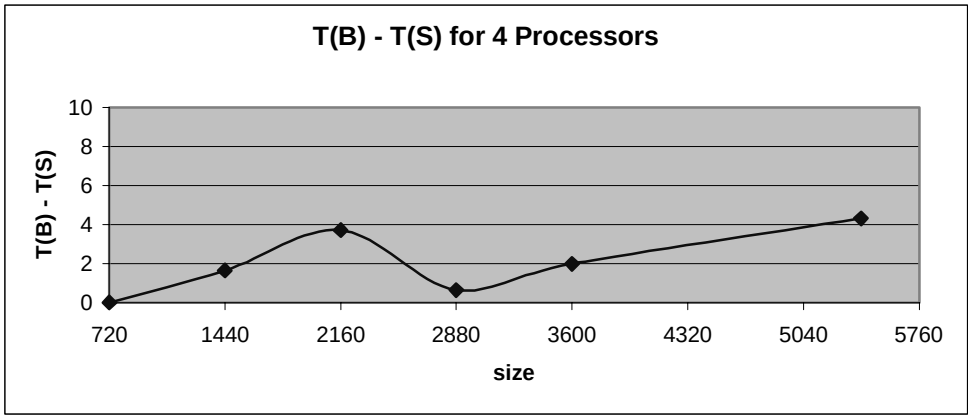


Figure 2.8 T(B)-T(S) for 4 processors
Remark: For four processors, the theoretical analysis predicted that blocked data layout is slower than stripped data layout. The actual data verifies this.

Lattice Size	Blocked	Stripped	T(B)-T(S)
720	2.87	2.50	0.38
1440	10.36	9.78	0.58
2160	21.61	22.12	-0.51
2880	37.95	39.22	-1.28
3600	59.95	62.03	-2.08
5400	134.86	138.09	-3.22

Table 2.4 Results for 9 processors (in seconds)

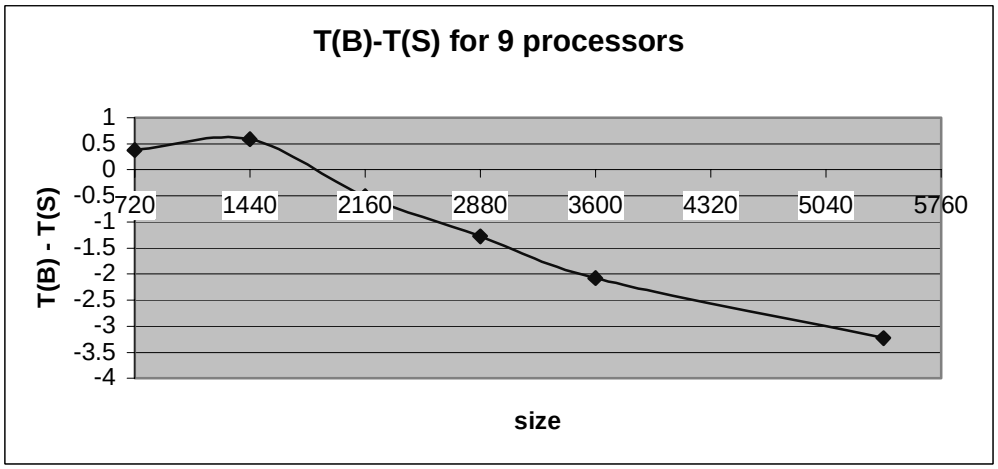


Figure 2.9 T(B)-T(S) for 9 processors
Remark: For nine processors, the theoretical analysis predicted that the critical lattice size is 2155.From our experiment, we found out that at size 1440, T(B) > T(S). However at size 2160, T(B) < T(S)which verifies the prediction.

Lattice Size	Blocked	Stripped	T(B)-T(S)
720	1.67	1.44	0.23
1440	5.53	5.74	-0.22
2160	12.21	13.45	-1.24
2880	21.58	22.97	-1.39
3600	33.00	35.35	-2.35
5400	76.05	81.49	-5.44

Table 2.5 Results for 16 processors (in seconds)

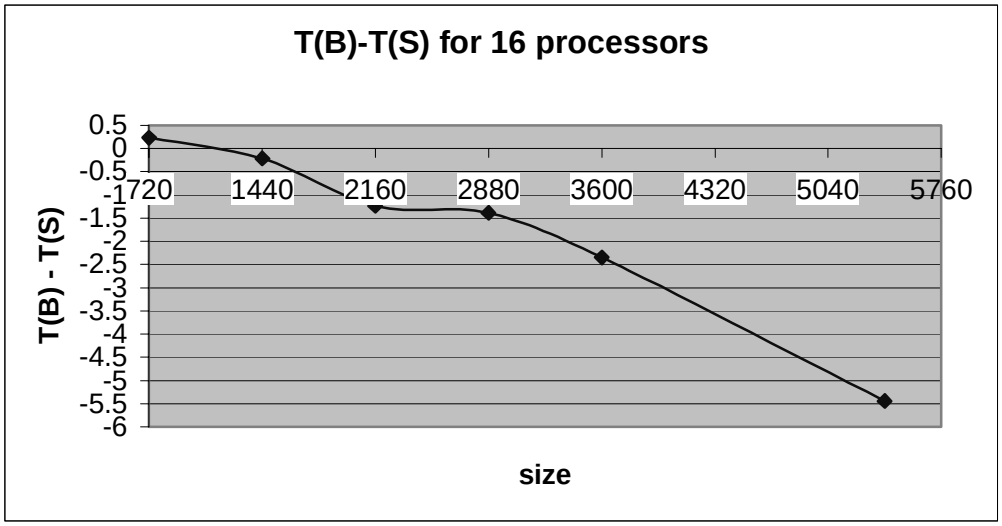


Figure 2.10 T(B)-T(S) for 16 processors
Remark: For 16 processors, the theoretical analysis predicted that the critical lattice size is 1437.From our experiment, we found out that at size 720, T(B) > T(S). However at size 1440, T(B) < T(S)which verifies the prediction.

Chapter 3 Parallel Algorithms For Random Selection Schemes: Design, Implementation and Analysis

In Chapter 2, we investigated the design and analysis of parallel algorithms for sweep selection on the LogP model. Moreover, we provided comparison results based on two different data layouts and derived a threshold value for lattice size for determining when blocked data layout outperforms stripped data layout.

In this chapter, our focus is on the random selection scheme. In the study of random selection schemes, we focus our attention on the design and implementation of parallel algorithms with efficient communication patterns, while simultaneously ascertaining the amount of random characteristics in each particular design.

3.1 Overview of Algorithm Design For Random Selection Schemes

The random selection scheme can be described in words as the selections of spins one by one uniformly randomly on the lattice (See Definition 1.2 and Algorithm 1.3). As described in Chapter 1, researchers must rely on random selection schemes to measure dynamic properties of the Ising spin system. Unfortunately, the sweep selection scheme is simply not usable to measure dynamic properties.

Our work on the random selection scheme is focused on algorithmic performance issues. Our goal is to improve the efficiency and reliability of random selection parallel algorithms. In order to achieve such a goal, we introduce an efficient and stable communication pattern into the parallel algorithm design.

We design and implement three random selection schemes. They are: (a) Truly random scheme, (b) Fixed Message Length scheme (called the FML scheme), and (c) a modified FML scheme which takes into account interleaving and changing interior to exterior ratio of spin selections. This scheme also utilizes a bit to decide the order of interior and exterior selection. We refer to this modified scheme as the α-scheme (or ALPHA scheme).

We base our work on random selection on blocked data layout. The ideas and principles of our work can be easily applied to stripped data layout or any other data layout.

3.2 The Truly Random Scheme

For the truly random scheme, the parallelization is straightforward. For each block of the lattice assigned to each processor, it is divided into Upper-Left and Lower-Right parts.
One Monte Carlo step is composed of several iterations. In each iteration, every processor is working on the same part of the lattice block. This ensures that no neighboring spins will be flipped at the same time, thus the detailed balance condition is

not violated. At the end of each iteration, processors communicate boundary spin update information with their neighbors.

The following figure shows the division of Upper-Left part and Lower-Right part, plus their relationship with each iteration.

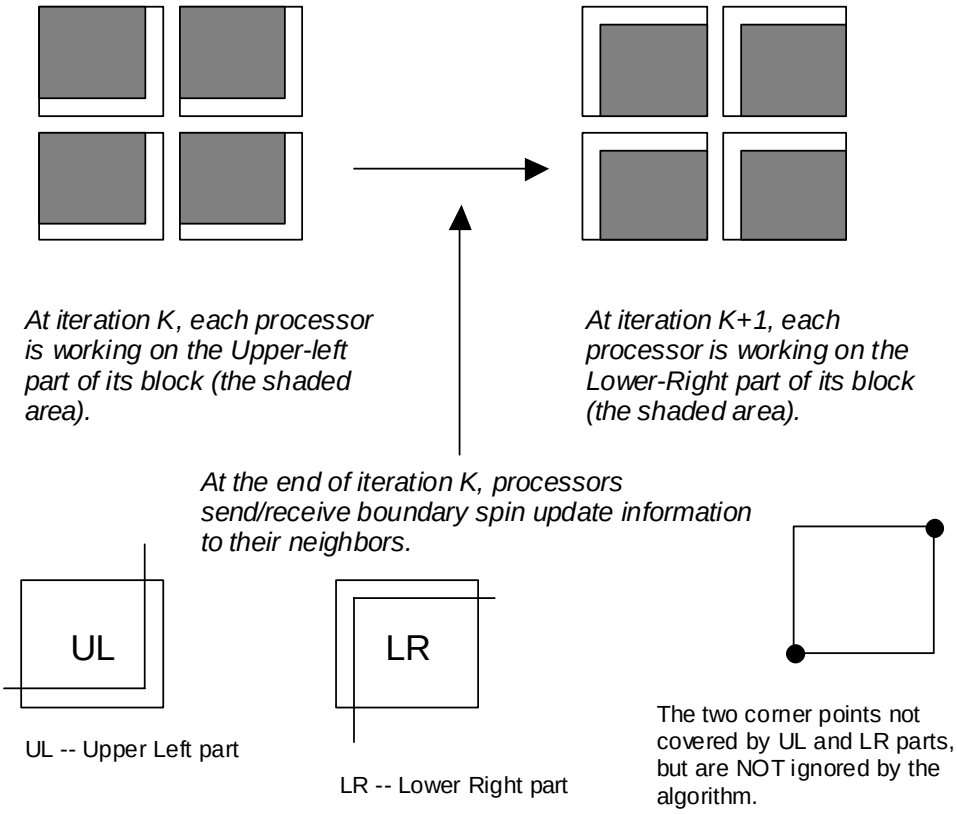


Figure 3.1 Block subdivision and Iterations

Due to the fact that neighboring spins cannot be flipped at the same time, any random selection scheme must impose this constraint. As such, the scheme we have just presented derives its name from the fact that it is truly random inside each iteration rather than being "truly random" throughout the whole algorithm.

3.2.1 Algorithm Pseudo Code

This section provides the pseudo code of the truly random scheme.

3.2.1.1 Assumptions and Definitions

Each processor holds a $h \times h$ block of the $S \times S$ lattice where $h = S/\sqrt{P}$.

Upper-left part: {(0,0), (h-2, h-2)}.
Lower-right part: {(1,1), (h-1, h-1)}.

3.2.1.2 The Pseudo Code

Algorithm TRULY RANDOM SCHEME

INPUT: A lattice of size $S \times S$ **,** and *P* processors. *S* must be divisible by *P*.
OUTPUT: Dynamic properties of the Ising spin system.

Begin

/*
Initialization: Each processor initializes its block of the sub-lattice and send/receive boundaries from its neighbors.
*/

For each processor P_i **DO**

Initialize ()

ENDFor

MC = 1

For MC <= Total Monte Carlo Steps **DO**

/*
Begin One Monte Carlo Step
*/

/*

Processor 0 generates a sequence of numbers $X_1, X_2, \ldots X_k$ $(1 \le X_i \le 4(h-1))$, until $\sum_{i=1}^{K} X_i \ge h^2$.

*/

For P_0 **DO**

Begin

K = 1
SUM = 0

While SUM <= 4(h-1) **DO**

Begin

X[k] = *rand()**4(h-1) + 1 //function definitions are listed at the end of the pseudo code
SUM = SUM + X[k]
K = K+1

ENDWhile

/*
After knowing K, processor 0 generates a number C from [0,k-1] such that each processor will select 2 corner points (upper right corner and lower left corner) at iteration C of the MC step.

(Continued from previous page)

Now, we have an array like this on processor 0:

$X_1,X_2,...X_k,C$
Processor 0 broadcasts this array to all processors on network
{Reason for the range of X_i to be [1,4(h-1)] is that $E(X_i)$ will be equal to 2(h-1),which is equal to the total number of rows and columns in the region (there are h-1 rows and h-1 columns in the region)}
*/

```
C = rand()*(k-1)
Array[1..k] = X[1..k]
Array[K+1] = C
Broadcast(Array, K+1)
```

ENDFor // For P_0 DO

For each processor P_i **DO**

Begin

```
j = 1
```

While j <= K **DO**

Begin

/*
//Stage 1
Pi selects Xj number of spins uniformly randomly from the Upper Left Part of its local block, and apply sequential Metropolis algorithm on those Xj number of selections

Pi sends spin update information of its top border and left border to its upper neighbor and left neighbor respectively.
Pi receives spin update information from its lower neighbor and right neighbor.
*/

```
RandomSelectAndApplyMetropolis (Xj,(0,0),(h-2,h-2))

Array1= ScanUpdated (top_border)
Array2= ScanUpdated (left_border)
Send (Array1, upper)
Send (Array2, left)
Receive (lower, bottom_array)
Receive (right, right_array)
```

/*
//Stage 2
Pi selects X_{j+1} number of spins uniformly randomly from the Lower Right Part of its local block, and apply sequential Metropolis algorithm on those X_{j+1} number of selections

Pi sends spin update information of its bottom border and right border to its lower neighbor and right neighbor respectively.

(Continued from previous page)

Pi receives spin update information from its upper neighbor and left neighbor

*****/

RandomSelectAndApplyMetropolis (X_{j+1},(1,1),(h-1,h-1))

Array1= *ScanUpdated (bottom_border)*
Array2= *ScanUpdated (right_border)*
*Send (*Array1, lower)
*Send (*Array2, right)
Receive (upper, top_array)
Receive (left, left_array)

/*
Processing the "Corner" iteration
*/

if (j == C **OR** j+1 == C) **DO** //C is the corner iteration

/*
Pi selects those two corner points of (0,h-1) and (h-1,0) and apply Metropolis algorithm on them.
Pi sends to upper and right neighbor corner (0,h-1)
Pi sends to lower and left neighbor corner (h-1,0)
P_i receives from its neighbors corner spins
*/

Metropolis ((0,h-1))
Metropolis ((h-1,0))

Send ((0,h-1), upper)
Send ((0,h-1), right)
Send ((h-1,0), lower)
Send ((h-1,0), left)

Receive (upper, top_array[0])
Receive (left, left_array[0])
Receive (lower, bottom_array[h-1])
Receive (right, right_array[h-1])

ENDIF

j = j + 2

ENDWhile // One Monte Carlo Step

MC = MC+1

IF (MC>= WARM_UP && MC % SAMPLE_INTERVAL == 0) **DO**
    // Run functions to measure all desired dynamic properties.
    Compute_Correlation()

ENDIF

(Continued from previous page)

ENDFor // MC<=TOTAL Monte Carlo Step

ENDAlgorithm

Function Definitions:

Initialize ()
Initialize the local lattice. Each spin will be randomly assigned a value of +1 or –1.

rand ()
Generates a random number uniformly from [0,1]

Broadcast (array, number_of_items)
Broadcast an array whose length is "number_of_items" to all processors.

Metropolis ((x, y))
Apply Metropolis Monte Carlo Algorithm (Algorithm 1.1) on the spin S (i,j)

RandomSelectAndApplyMetropolis (N, From, To)
Uniformly randomly selection N spins on the rectangle specified by upper-left point "From" and lower-right point "To", apply Metropolis Monte Carlo Algorithm (Algorithm 1.1) on each selected spin.

ScanUpdated (Border)
Scans the border specified by "Border" and put the updated spin of that border into an output array, return the output array to caller.

Send (DATA , DESTINATION)
Send the data buffer "DATA" to the processor specified by "DESTINATION"

Receive(SOURCE, BUFFER)
Receive data from the processor specified by "SOURCE" and store the data in "BUFFER"

Compute_Correlation ()
Compute the correlation of the spin system, which is a dynamic property. This function can be replaced by any other customized functions to measure different dynamic properties.

Algorithm 3.1 TRULY RANDOM SCHEME

3.2.2 Communication Pattern Design

In this section, we introduce the design of the communication pattern for the truly random scheme.

Because we are examining the truly random scheme, the number of border spins being updated in each iteration will be varying. We do not want to send the whole border out. Instead, we want to send out **only** those spins that have been updated. So, the lengths of the messages are varying. We used the following mechanism to handle communication in the truly random selection scheme:

We allocated a buffer for each border. We called that buffer the work_array. The length of work_array is the same as the length of the border. It is indexed from 0 to h-1. Work_array is initialized to be 0, which implies that the corresponding spin on the border has not been changed.

Whenever a spin i is changed on the border, work_array[i] will be set to the latest value of that spin (-1 or 1).After all the updates are performed for this iteration, we will send the spin update information to the neighbors. We scan work_array to find out all those items that are not 0 and put them into "send" arrays.

For example: assuming the border length = 8:
work_array is {0,-1,1,-1,-1,1,0,1}. So, spin[0] on the border has not been changed, spin[1] has been changed with the latest value of -1, spin[2] has been changed with the latest value of 1, etc. According to the above work_array, we have the following send array:

send_array is {1,-1;2,1;3,-1;4,-1;5,1;7,1}. We can compute the length of the send array by multiplying 2 with the number of non-zero items in the send array.

After the send array is available, we send two messages to neighbors. The first message contains the length of the second message. Thus, the receiver can correctly allocate its receiving buffer for the second message. The second message contains the actual send array which has the actual spin update information.

3.2.3 Algorithm Analysis of the Truly Random Scheme

This section analyzes the runtime cost of the truly random scheme.

LEMMA 3.1. *Given P processors and assuming a lattice size of $S \times S$, let h be equal to $\frac{S}{\sqrt{P}}$, the expected communication cost of one iteration of the truly random scheme is*

$$T_{comm} = 2hb + 4(L + 2o + 9g),$$

where b > 0 is a constant and L, o, and g are parameters of the LogP model.

PROOF. We set the time unit to be the time of sending a one-byte message divided by $L + 2o$. By this definition, the cost of sending a k-byte message is simply $L + 2o + (k-1)g$. However, we should scale the computation cost and other costs to this unit by multiplication of constants.

During each iteration, processor P_i works on either the UL or the LR part of their sub block and has two neighbors to communicate with.

The cost of setting up one communication buffer is $h \cdot b$, where h is length of the buffer and b is the cost of processing one item of the buffer. For each iteration, we have two

work_arrays to scan. Thus, the cost of setting up communication buffers for one iteration is:

$$T_{setup} = 2hb .$$

We always send a header message to neighbors to indicate the length of the actual message. If there is nothing to update, we still need to send the head message containing the value 0. The header message is simply an integer, which is 4 bytes. For each iteration, we send two header messages. So, the cost of sending header messages for one iteration is:

$$T_{header} = 2(L + 2o + 3g) .$$

After sending the header message, we need to send the actual message containing the spin update information on the borders. As described in the section of communication pattern design, we send out two send arrays each containing 2m integers (8m bytes) where m is the number of spins that have been updated in this iteration. The cost of sending the actual message is:

$$T_{actual} = 2(L + 2o + (8m - 1)g) ,$$

where m is the number of border spins being updated in one iteration. We use the expected value of m, i.e. E(m), to perform our expected case analysis.

At the i^{th} iteration, there are X_i total number of spins being selected uniformly randomly on the UL part or LR part. Each such part is of size $(h-1)\times(h-1)$.

Thus, $E(m) = E(X_i)\dfrac{h-1}{(h-1)^2} = \dfrac{E(X_i)}{h-1}$

Because X_i is a uniform random number from [0,4(h-1)], $E(X_i) = 2(h-1)$. So, we have

$$E(m) = \frac{E(X_i)}{(h-1)} = \frac{2(h-1)}{(h-1)} = 2 .$$

Using this value in T_{actual} , we have

$$T_{actual} = 2(L + 2o + 15g) .$$

The expected communication cost for one iteration is simply the summation of $T_{setup,}$ $T_{header,}$ and T_{actual}:

$$\begin{aligned} T_{comm} &= T_{setup} + T_{header} + T_{actual} \\ &= 2hb + 2(L + 2o + 3g) + 2(L + 2o + 15g) \\ &= 2hb + 4(L + 2o + 9g) . \end{aligned}$$

LEMMA 3.2. *Given P processors and assuming a lattice size of $S \times S$, let h be equal to $\frac{S}{\sqrt{P}}$, the expected number of iterations in one Monte Carlo step of the truly random scheme is*

$$E(K) = \frac{h^2}{2(h-1)} \approx h/2 .$$

PROOF. One MC step consists of K iterations. K is not a fixed number; it's a random variable related to X_i. From the algorithm we see that processor 0 generates a sequence of numbers $X_1, X_2, \ldots X_k$ $(1 \le X_i \le 4(h-1))$, until $\sum_{i=1}^{K} X_i \ge h^2$. X_i is the number of spin selections for the i^{th} iteration, and K is the total number of iterations in that Monte Carlo step. The expected value of K is

$$E(K) = \frac{h^2}{E(X_i)} = \frac{h^2}{2(h-1)} \approx h/2 .$$

LEMMA 3.3. *Given P processors and assuming a lattice size of $S \times S$, let h be equal to $\frac{S}{\sqrt{P}}$, the expected communication cost of one Monte Carlo step of the truly random scheme is*

$$T_{MC\_comm} = h^2 b + 2h(L + 2o + 9g),$$

where b > 0 is a constant and L, o, and g are parameters of the LogP model.

PROOF. The expected communication cost for one Monte Carlo step is the product of the communication cost per iteration and the expected number of iterations per Monte Carlo step, i.e.,

$$T_{MC\_comm} = E(k) \times T_{comm},$$

from Lemma 3.1 and Lemma 3.2,

$$T_{MC\_comm} = E(k) \times T_{comm} = \frac{h}{2}\left(2hb + 4(L + 2o + 9g)\right) = h^2 b + 2h(L + 2o + 9g).$$

Clearly, the communication costs for truly random selection are considerable. Thus the need to utilize schemes that maintain high levels of randomness while reducing communication costs should be explored. In section 3.3 and 3.4, we present two such schemes.

LEMMA 3.4. *Given P processors and assuming a lattice size of* $S \times S$ *, let h be equal to* $\frac{S}{\sqrt{P}}$*, the computation cost of one Monte Carlo step of the truly random scheme is*

$$T_{comp} = Ch^2, \ C > 0 \text{ is a constant.}$$

PROOF. The number of spin selections for one Monte Carlo step is h^2. Let the cost of applying the Metropolis algorithm on one spin be C (this cost includes determining the potential energy change, generating a random number and bookkeeping the internal energy). This is a positive constant. Thus, the total cost of computation can be represented simply by

$$T_{comp} = Ch^2 .$$

THEOREM 3.5. *Given P processors and assuming a lattice size of* $S \times S$ *, let h be equal to* $\frac{S}{\sqrt{P}}$*, the total cost of one Monte Carlo step of the truly random scheme is*

$$T_{truly} = (b + C)h^2 + 2h(L + 2o + 9g),$$

where b > 0, C > 0 and they are both constants, L, o and g are parameters of the LogP model.

PROOF. The total cost of one Monte Carlo Step of the truly random scheme is simply the summation of the communication cost and the computation cost. From Lemma 3.3 and Lemma 3.4, we have

$$\begin{aligned} T_{truly} &= T_{MC\_comm} + T_{comp} \\ &= h^2 b + 2h(L + 2o + 9g) + Ch^2 \\ &= (b + C)h^2 + 2h(L + 2o + 9g). \end{aligned}$$

3.2.4 Problems With the Truly Random Scheme

In the truly random scheme, the number of spins being selected on the border is not fixed; it could be any value from 0 to h-1. This causes the message length to be varying.

Since the message length is not fixed, every time the sender wants to send out a message, it needs to provide the message length to the receiver. This ensures that the receiver can anticipate the needed size of the receiving buffer. In our case, the sender sends out a header message first. This message contains an integer value representing the message length of the second message to the receiver. Thus, the receiver is ready to receive the (second) actual message.

This scheme is inefficient and unstable. It is inefficient due to the fact that every time there is a need for communication, it sends out two messages for one piece of information. Instead of incurring a cost of $L + 2o + (k-1)g$ for one piece of k-byte information, it now costs $L + 2o + L + 2o + (k-1)g = 2(L + 2o) + (k-1)g$. Usually, $L + 2o$ is much greater than g. Thus, for small k, the second cost may double the first one. We need to consider new communication schemes with better performance.

This scheme is also unstable due to the fact that it has varying message lengths and the lengths can be as large as $(\frac{S}{\sqrt{P}} - 1)$. If the number of processors (P) is fixed, then the message length is O(S) where $S \times S$ is the size of the lattice. As S grows larger, the message length grows at the same rate. In the real world, the available communication buffer (receiver buffer in particular) is typically limited. If the message length grows over the limit, the system may become unstable and possibly crash. New implementations to break down the long message into fixed size pieces must be developed under such circumstances. This will inevitably add more cost.

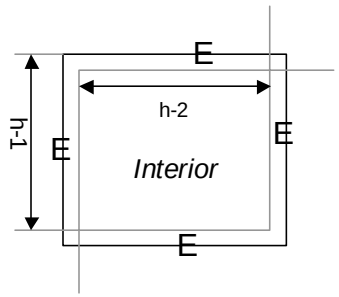
Regarding the above concerns and issues, we designed our own communication scheme with Fixed Message Length. We call this the FML scheme, and we introduce it in the next section.

3.3 The Fixed Message Length (FML) Scheme

In this section, we will introduce the design, implementation and analysis of the FML scheme.

3.3.1 Design of the FML Scheme

We categorize lattice spins into interior spins and exterior spins. Exterior spins are those spins that are on the border of the lattice, while interior spins are not on the boundary. The following figure shows the division of the lattice into exterior and interior regions on processor P_i.



Division of the block into interior and exterior regions. There are (h-2)*(h-2) interior spins, and 2(h-1)-1 exterior spins for the upper left part and lower right part respectively.

Figure 3.2 Interior and Exterior of the block

For any type of random selection schemes, the probability of an interior spin being selected in one Monte Carlo step should be the same as the probability of an exterior spin being selected in one Monte Carlo step.

Our goal is to design an algorithm such that for every iteration of that algorithm, it will update a fixed number of spins on the exterior part, while still satisfying the above condition of equal probability. Once the number of spins being updated on the exterior is fixed, the message length is also fixed. This should substantially improve the efficiency of the parallel algorithm. We describe our design below:

Given a set of spins, let t be the total number of spins, and let x be the number of selections made from those spins. We select spins *with replacement*, i.e. once a spin is selected, it will **not** be excluded from the set. This is exactly the case of lattice spin model, i.e. once a spin is selected, it is not taken away from the lattice, it is still there and available for future selections.

From basic statistics, the probability of a spin being selected (at least once) out of t spins by x selections is:

$$P(x,t) = 1 - \left(1 - \frac{1}{t}\right)^x$$
$$= \frac{t^x - (t-1)^x}{t^x}.$$

From basic statistics [Casella et al. 2001], we know that:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$, n is a non-negative integer

so,

$$(t-1)^x = (-1+t)^x = \sum_{k=0}^{x} \binom{x}{k} (-1)^k t^{x-k}$$

$$= t^x + x(-1)^1 t^{x-1} + \frac{x(x-1)}{2}(-1)^2 t^{x-2} + \frac{x(x-1)(x-2)}{6}(-1)^3 t^{x-3} + \cdots\cdots$$

$$= t^x - xt^{x-1} + \frac{x(x-1)}{2} t^{x-2} - \frac{x(x-1)(x-2)}{6} t^{x-3} + \cdots\cdots$$

$\Rightarrow$

$$P(x,t) = \frac{t^x - (t-1)^x}{t^x}$$

$$= \frac{xt^{x-1} - \frac{x(x-1)}{2} t^{x-2} + \frac{x(x-1)(x-2)}{6} t^{x-3} - \cdots\cdots}{t^x}$$

$$= \frac{x}{t} - \frac{\frac{x(x-1)}{2}}{t^2} + \frac{\frac{x(x-1)(x-2)}{6}}{t^3} - \cdots\cdots$$

$$= \frac{x}{t} - \left(\frac{x(x-1)}{2t^2} - \frac{x(x-1)(x-2)}{6t^3} + \cdots \right)$$

$$\approx \frac{x}{t}, \textit{ when } x << t.$$

So, we can use $\frac{x}{t}$ to approximate the probability $P(x,t)$.

We note that a good approximation occurs when *t* is greater than or equal to *50x*, which means $\frac{x}{t} \le \frac{1}{50} = 0.02$ and $\left(\frac{x}{t}\right)^2 \le \frac{1}{2500} = 0.0004$. The quadratic term is negligible.

As in the truly random scheme, the FML scheme still divides the lattice into upper left parts and lower right parts. This is required since all processors must work on the same part to ensure that neighboring spins are never flipped at the same time. In the FML scheme, one iteration is composed of the following sub steps:

a) For the upper left part, uniformly randomly select $2(h-2)$ spins from the interior.

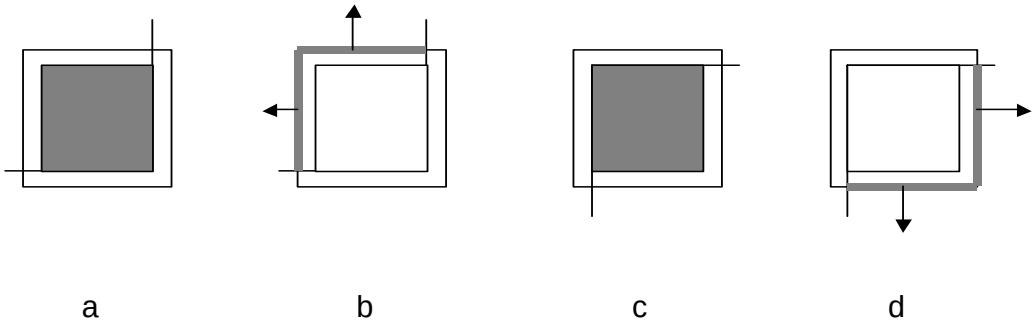
b) Select x spins from both exterior borders of the upper left part. Communicate with upper neighbor and left neighbor (Send and receive messages of length x).

c) For the lower right part, uniformly randomly select $2(h-2)$ spins from the interior.

d) Select x spins from both exterior borders of the lower right part. Communicate with lower neighbor and right neighbor (Send and receive messages of length x).

The following figure shows one iteration in the FML scheme:

Sub steps of one iteration in the FML scheme



Shaded areas in each sub step represent the set of spins being selected in that sub step. Arrows represents communication.

Figure 3.3 One iteration of the FML scheme

Now, we determine the value of x, i.e., the number of spins selected on the exterior borders.

According to the above analysis, $P(x,t) = \frac{x}{t}$.

Thus, the probability of an interior spin being selected throughout the whole iteration is

$$P_{interior} = P(4(h-2),(h-2)^2) = \frac{4(h-2)}{(h-2)^2} = \frac{4}{h-2},$$

and the probability of an exterior spin being selected throughout the whole iteration is

$$P_{exterior} = P(x, 2(h-1)-1) = \frac{x}{2h-3}.$$

and,

$$P_{interior} = P_{exterior}$$
$$\Leftrightarrow \frac{4}{h-2} = \frac{x}{2h-3}$$
$$\Leftrightarrow \quad x = \frac{4(2h-3)}{h-2} = \frac{8(h-2)+4}{h-2} = 8 + \frac{4}{h-2}.$$

This implies that we need to select 8 spins on the exterior borders and select the 9th spin with probability $\frac{4}{h-2}$. By doing so, we satisfy the equal probability condition. In regards to the message length, we fix it to be 9 integers, which is 36 bytes on a 32-bit machine.

The total number of spin selections per iteration is $4(h-2)+2(8+\frac{4}{h-2}) \approx 4h$, and the total number of spin selections required for one MC step is h^2. So number of iterations is $\frac{h^2}{4h} = \frac{h}{4}$.

3.3.2 Algorithm Pseudo Code

This section introduces pseudo code for the FML scheme.

3.3.2.1 Assumptions and Definitions

Each processor holds a $h \times h$ block of the $S \times S$ lattice and $h = S/\sqrt{P}$.

Upper-left part : {(0,0), (h-2,h-2)}
Lower-right part: {(1,1), (h-1,h-1)}
Interior: {(1,1), (h-2,h-2)}
Exterior: 4 borders {(0,0), (0,h-2)}, {(0,0),(h-2,0}, {(h-1,1),(h-1,h-1)} and {(1,h-1),(h-1,h-1)}

3.3.2.2 Algorithm Pseudo Code

Algorithm FML SCHEME

INPUT: A lattice of size $S \times S$ **,** and *P* processors. *S* must be divisible by *P*.
OUTPUT: Dynamic properties of the Ising spin system.

Begin

```
/*
Initialization:  Each processor initializes its block of the sub-lattice and send/receive boundaries from its neighbors.
*/

For each processor Pi DO

  Initialize () //function definitions can be found at the end of the pseudo code

ENDFor

  MC = 1

For MC <= Total Monte Carlo Step DO

Begin
```

(Continued from previous page)

/* For P_0,
Randomly generate a number k from [0,h/4) and broadcast it to each node. Then, at the kth iteration of that MC Step, each node will examine the upper-right corner (0,h-1) and lower left corner (h-1, 0) of the block it holds.
*/

For P_0 **DO**

Begin

K = rand()*h/4
Broadcast(K,1)

ENDFor

For each processor P_i **DO**

Begin

j = 1

While j <= h / 4 **DO**

Begin

/*
local updates on the upper left part
Uniformly randomly select $2(h-2)$ spins from the interior part and apply sequential Metropolis algorithm on those $2(h-2)$ number of selections.
Uniformly randomly select 8 spins from the exterior (top and left borders) and select the 9^{th} one with probability 4/(h-2). Apply sequential Metropolis algorithm on those 8(9) spins selected.
*/

RandomSelectAndApplyMetropolis (2(h-2),(1,1),(h-2,h-2));

IF (*rand()* < 4/(h-2)) **DO**
NUM = 9
ELSEIF
NUM = 8
ENDIF

NUM1 = *rand()**NUM
NUM2 = NUM – NUM1

RandomSelectAndApplyMetropolis (NUM1,(0,0),(0,h-2));
RandomSelectAndApplyMetropolis (NUM2,(0,0),(0,h-2));

/*
Communication with neighbors
Pi sends spin update information of its top border and left border to its upper neighbor and left neighbor respectively.
Pi receives spin update information from its lower neighbor and right neighbor.
*/

(Continued from previous page)

Array1= *ScanUpdated (top_border)*
Array2= *ScanUpdated (left_border)*

*Send (*Array1, upper)
*Send (*Array2, left)
Receive (lower, bottom_array)
Receive (right, right_array)

/*
Uniformly randomly select $2(h-2)$ spins from the interior part and apply sequential Metropolis algorithm on those $2(h-2)$ number of selections.
Uniformly randomly select 8 spins from the exterior (bottom and right borders) and select the 9th one with probability 4/(h-2). Apply sequential Metropolis algorithm on those 8 (9) spins selected.
*/

RandomSelectAndApplyMetropolis (2(h-2),(1,1),(h-2,h-2))

IF (*rand()* < 4/(h-2)) **DO**
NUM = 9
ELSEIF
NUM = 8
ENDIF

NUM1 = *rand()**NUM
NUM2 = NUM – NUM1

RandomSelectAndApplyMetropolis (NUM1,(h-1,0),(h-1,h-1))
RandomSelectAndApplyMetropolis (NUM2,(0,h-1),(h-1,h-1))

/*
Pi sends spin update information of its bottom border and right border to its lower neighbor and right neighbor respectively.
Pi receives spin update information from its upper neighbor and left neighbor.
*/

Array1= *ScanUpdated (bottom_border)*
Array2= *ScanUpdated (right_border)*

*Send (*Array1, lower)
*Send (*Array2, right)
Receive (upper, top_array)
Receive (left, left_array)

/*
Pi selects those two corner points of (0,h-1) and (h-1,0) and apply Metropolis algorithm on them.
Pi sends to upper and right neighbor corner (0,h-1)
Pi sends to lower and left neighbor corner (h-1,0)
P_i receives from its neighbors corner spins
*/

IF j == k **DO**

Metropolis ((0,h-1))
Metropolis ((h-1,0))

(Continued from previous page)

Send ((0,h-1), upper)
Send ((0,h-1), right)
Send ((h-1,0), lower)
Send ((h-1,0), left)

Receive (upper, top_array[0])
Receive (left, left_array[0])
Receive (lower, bottom_array[h-1])
Receive (right, right_array[h-1])

ENDIF

j = j+1

ENDWhile

ENDFor // One Monte Carlo Step

MC = MC+1

IF (MC>= WARM_UP && MC % SAMPLE_INTERVAL == 0) **DO**
// Run function(s) to measure all desired dynamic properties.
Compute_Correlation();

ENDIF

ENDFor //MC<=Monte Carlo Step

ENDAlgorithm

Function Definitions:

Initialize ()
Initialize the local lattice. Each spin will be randomly assigned a value of +1 or –1.

Metropolis ((x, y))
Apply Metropolis Monte Carlo Algorithm (Algorithm 1.1) on the spin S (i,j)

RandomSelectAndApplyMetropolis (N, From, To)
Uniformly randomly selection N spins on the rectangle specified by upper-left point "From" and lower-right point "To", apply Metropolis Monte Carlo Algorithm (Algorithm 1.1) on each selected spin.

Send (DATA , DESTINATION)
Send the data buffer "DATA" to the processor specified by "DESTINATION"

Receive(SOURCE, BUFFER)
Receive data from the processor specified by "SOURCE" and store the data in "BUFFER"

Compute_Correlation ()
Compute the correlation of the spin system, which is a dynamic property. This function can be replaced by any other customized functions to measure different dynamic properties.

Algorithm 3.2 FML SCHEME

3.3.3 Communication Pattern Design

The FML scheme has a more efficient and concise communication pattern than the truly random scheme. For the FML scheme, there are at most 9 selections of exterior spins in one iteration. Because each selection can be mapped into an integer number using the offset of the selected spin, we can use an array of 9 integers for each border. Initially, every key value of the array is –1. Whenever a spin on that border is selected and updated, we put the offset of that spin into the array. Processor Pi sends the array to its corresponding neighbor. The neighboring processor will read the array and update spins until it meets the first –1.

Message length is fixed to be 9 integers (36 bytes) in the FML scheme

3.3.4 Algorithm Analysis of the FML Scheme

This section provides performance analysis of the FML scheme. In Section 3.3.5 we will compare the results of the FML scheme with those of the truly random scheme.

LEMMA 3.6. *Given P processors and assuming a lattice size of $S \times S$, the communication cost of one iteration of the FML scheme is*

$$T_{comm\_iteration} = 4(L + 2o + 35g),$$

where L, o, and g are parameters of the LogP model.

PROOF. For the FML scheme, during each iteration, processor P_i works on both the UL part **and** the LR part of their sub-block. It communicates with 4 neighbors. P_i sends out 4 messages each of size 36 bytes.

$$T_{comm\_iteration} = 4(L + 2o + 35g).$$

LEMMA 3.7. *Given P processors and assuming a lattice size of $S \times S$, let h be equal to $\frac{S}{\sqrt{P}}$, the communication cost of one Monte Carlo step of the FML scheme is*

$$T_{comm} = h(L + 2o + 35g),$$

where L, o, and g are parameters of the LogP model.

PROOF. By the design of the FML scheme, we note that there are totally $\frac{h}{4}$ iterations for one MC step. From Lemma 3.6, we obtain the communication cost of one Monte Carlo step of the FML scheme:

$$T_{comm} = \frac{h}{4} \cdot 4(L + 2o + 35g) = h(L + 2o + 35g).$$

LEMMA 3.8. *Given P processors and assuming a lattice size of* $S \times S$ *, let h be equal to* $\frac{S}{\sqrt{P}}$*, the computation cost of one Monte Carlo step of the FML scheme is*

$$T_{comp} = Ch^2,$$

where C > 0 is a constant.

PROOF. Similar to the proof in the truly random case, let the cost of applying the Metropolis algorithm on one spin be *C* (this cost includes determining the potential energy change, generating a random number and bookkeeping the internal energy), which is a constant number independent of the algorithm. There are totally h^2 such selections in one Monte Carlo step of the FML scheme, thus the total cost of computation is

$$T_{comp} = Ch^2.$$

THEOREM 3.9. *Given P processors and assuming a lattice size of* $S \times S$ *, let h be equal to* $\frac{S}{\sqrt{P}}$*, the total cost of one Monte Carlo step of the FML scheme is*

$$T_{FML} = Ch^2 + h(L + 2o + 35g),$$

where C > 0 is a constant and L, o, and g are parameters of the LogP model.

PROOF. The total cost of the FML algorithm is the summation of its communication cost and computation cost, thus from Lemma 3.7 and Lemma 3.8,

$$T_{FML} = T_{comp} + T_{comm} = Ch^2 + h(L + 2o + 35g).$$

3.3.5 Performance Comparison Between FML and Truly Random

$$\begin{aligned} T_{truly} - T_{FML} &= (1 + C)h^2 + 2h(L + 2o + 9g) - (Ch^2 + h(L + 2o + 35g)) \\ &= h^2 + h(L + 2o - 17g) \\ &= h(h + L + 2o - 17g) > 0, \text{when } h + L + 2o - 17g > 0 \end{aligned}$$

In most cases, $h + L + 2o - 17g >> 0$, so T_{truly} is much slower than T_{FML}

3.3.6 Problems of the FML Scheme

The FML scheme brings clear improvements in regards to algorithm efficiency and system stability. However, it imposes a rigid spin selection pattern of interleaving interior selections and exterior selections. It selects 2(h-1) interior spins, then 8 or 9 exterior spins, then interior again, then exterior, etc. This introduces a level of sequentiality into the algorithm, i.e. "a certain lack of randomness". This affects the measurement of dynamic properties (since measuring dynamic properties requires random selections). However, if we want to have fixed message lengths and efficient communication patterns, clearly some loss of randomness will be incurred. However, we want to minimize this loss. Therefore, we need to design a better scheme that incorporates more randomness than the FML scheme while still maintaining efficiency. As such, we will present the α-scheme in the next section.

3.4 The α-Scheme

This section presents the design, pseudo code and analysis of a variant of the FML scheme which we will call the α-scheme.

3.4.1 Design of the α-Scheme

As in the FML scheme, we categorize lattice sites into exterior and interior. We still fix the number of exterior selections to be $8+\frac{4}{h-2}$. So, we maintain fixed message length. However, unlike the FML scheme, the α-scheme does not have the rigid sequences of exterior and interior selections. Instead, it creates more diversity by adding a *varying ratio*. We call this ratio α.

Assume that the FML scheme has the selection sequence given below:

2(h-2)I, 8E, 2(h-2)I, 8E, …… 2(h-2)I, 8E , (1)
('I' refers to the interior, 'E' refers to the exterior, '2(h-2)' refers to 2(h-2) selections)
Each (2(h-2)I , 8E) pair forms a local iteration at the end of which processors communicate with their neighbors.

The goal of α-scheme is to break the (2(h-2)I , 8E) pair into an "unpredictable" number of small pairs having a random order of interior and exterior. We introduced a ratio α that divides the original pair into smaller pairs:

If we fix $\alpha = 2$, we would have the following new selection sequence:

(h-2)I, 4E, (h-2)I, 4E, (h-2)I, 4E , ……. , (h-2)I, 4E, (h-2)I, 4E, (2)

It splits the (2(h-2)I , 8E) pair into 2 pairs of $((h-2)I, 4E)$.

Comparing sequence (2) with sequence (1), we see that in sequence (1), exterior spins will only be selected after 2(h-2) interior selections. However, in sequence (2), exterior spins will be selected twice during the first 2(h-2) interior selections. Sequence (2) creates more randomness than sequence (1).

Furthermore, instead of fixing α, we should vary α to introduce even more randomness into the sequence. To break the (2(h-2)I , 8E) pair into small pairs of ((Intk)I, (Extk)E), the only condition we need to satisfy is $\sum_k Ext_k = 8$, and $\sum_k Int_k = 2(h-2)$. So, we uniformly randomly generate a series of random numbers X_k in the range of [1,8] such that $\sum_k X_k = 8$. Let $Ext_k = X_k$ and $Int_k = 2(h-2)\cdot\frac{Ext_k}{8} = \frac{Ext_k \cdot (h-2)}{4}$. By doing this, we split one original pair of (2(h-2)I , 8E) into small pairs of ((Int_1) I, (Ext_1)E), ((Int_2) I, (Ext_2)E) …. ((Int_k) I, (Ext_k)E) and more importantly, those Int_k and Ext_k are randomly generated.

Finally, in addition to a varying α, we used random bit flipping to decide the order of selection – interior first or exterior first. For each small pair of ((Int_k)I, (Ext_k)E), the α-scheme uniformly randomly decides the order of this pair : ((Int_k)I, (Ext_k)E) or ((Ext_k)E , (Int_k)I).

3.4.2 Algorithm Pseudo Code

This section introduces the pseudo code of the α-scheme

3.4.2.1 Assumptions and Definitions

Assume each processor holds a $h \times h$ block of the $S \times S$ lattice and $h = \frac{S}{\sqrt{P}}$.

Upper-left part : {(0,0), (h-2,h-2)},
Lower-right part: {(1,1), (h-1,h-1)},
Interior: {(1,1), (h-2,h-2)},
Exterior: 4 borders {(0,0), (0,h-2)}, {(0,0),(h-2,0}, {(h-1,1),(h-1,h-1)} and {(1,h-1),(h-1,h-1)}.

3.4.2.2 Pseudo Code

Algorithm α-SCHEME

INPUT: A lattice of size $S \times S$, and *P* processors. *S* must be divisible by *P*.
OUTPUT: Dynamic properties of the Ising spin system.

Begin
/*
Initialization: Each processor initializes its block of the sub-lattice and send/receive boundaries from its neighbors.

(Continued from previous page)

For each processor P_i **DO**

Initialize ()

ENDFor

MC = 1

For MC <= Total Monte Carlo Step **DO**

Begin

/* On processor 0:
Randomly generate a number C from [0,h/4) and broadcast it to each node. Then, at the Cth iteration of that MC Step, each node will examine the upper-right corner (0,h-1) and lower left corner (h-1, 0) of the block it holds.
*/

For P_0 **DO**

Begin

C = *rand()**h/4
Broadcast(K,1)

ENDFor

For each processor P_i **DO**

j = 1

While j <= h / 4 **DO**

Begin

/*
Determine the total number of spins to select on upper left exterior (Ext):
Generate a random number N uniformly from [0,1], if N<4/(h-2), Ext = 9 else Ext=8
*/

N = *rand()*

IF (N < 4/(h-2)) **DO**
Ext = 9
ELSEIF
Ext = 8
ENDIF

U=Ext

//Work on the upper left part first
While U>0 **DO**
Begin

(Continued from previous page)

```
X = rand()*U+1

U=U-X

N=rand()

if (N<0.5) DO //interior first

 /*
Select 2(h-2)X/8 number of spins in the interior, and apply Metropolis algorithm on each
selection
Select X number of spins on the upper left exterior (top border and left border), and apply
Metropolis algorithm on each selection
 */

RandomSelectAndApplyMetropolis (2(h-2)X / 8,(1,1),(h-2,h-2))

X1 = X*rand()
X2 = X-X1
RandomSelectAndApplyMetropolis (X1,(0,0),(0,h-2))
RandomSelectAndApplyMetropolis (X2,(0,0),(h-2,0))

 ELSEIF //exterior first

 /*
Select X number of spins on the upper left exterior (top border and left border), and apply
Metropolis algorithm on each selection
Select 2(h-2)X/8 number of spins in the interior, and apply Metropolis algorithm on each
selection
*/

 X1 = X*rand()
 X2 = X-X1
 RandomSelectAndApplyMetropolis (X1,(0,0),(0,h-2))
 RandomSelectAndApplyMetropolis (X2,(0,0),(h-2,0))

 RandomSelectAndApplyMetropolis (2(h-2)X / 8,(1,1),(h-2,h-2))

 ENDIF

ENDWhile  //While U>0

/*
//Communication with upper and left neighbors
Pi sends spin update information of its top border and left border to its upper neighbor and left
neighbor respectively.
Pi receives spin update information from its lower neighbor and right neighbor.
*/

Array1= ScanUpdated (top_border)
Array2= ScanUpdated (left_border)

Send (Array1, upper)
```

(Continued from previous page)

```
   Send (Array2, left)
   Receive (lower, bottom_array)
   Receive (right, right_array)

/*
//local update on the lower right part
Determine the total number of spins to select on lower right exterior (Ext):
Generate a random number N, if N<4/(h-2), Ext = 9 else Ext=8
*/
   N = rand();

   IF (N < 4/(h-2) ) DO
         Ext = 9
   ELSEIF
         Ext = 8
   ENDIF

   U=Ext

While U>0 DO

Begin

    X = rand()*U+1

    U=U-X

    N=rand()

    IF N<0.5 DO //interior first

    /*
    Select 2(h-2)X/8 number of spins in the interior, and apply Metropolis algorithm on each
    selection
    Select X number of spins on the lower right exterior (bottom border and right border), and
    apply Metropolis algorithm on each selection
    */

    RandomSelectAndApplyMetropolis (2(h-2)X / 8,(1,1),(h-2,h-2))

    X1 = X*rand()
    X2 = X-X1
    RandomSelectAndApplyMetropolis (X1,(h-1,1),(h-1,h-1))
    RandomSelectAndApplyMetropolis (X2,(0,h-1),(h-1,h-1))

     ELSEIF //exterior first
     /*
     Select X number of spins on the lower right exterior (bottom border and right border), and
     apply Metropolis algorithm on each selection
     Select 2*(h-2)*X/8 number of spins in the interior, and apply Metropolis algorithm on each
     selection
    */
```

```
    X1 = X*rand()
(Continued from previous page)

    X2 = X-X1
    RandomSelectAndApplyMetropolis (X1,(h-1,1),(h-1,h-1))
    RandomSelectAndApplyMetropolis (X2,(0,h-1),(h-1,h-1))

    RandomSelectAndApplyMetropolis (2(h-2)X / 8,(1,1),(h-2,h-2))

    ENDIF

END While U>0

/*
//Communication with lower and right neighbors
Pi sends spin update information of its bottom border and right border to its lower neighbor and
right neighbor respectively.
Pi receives spin update information from its upper neighbor and left neighbor.
*/

   Array1= ScanUpdated (bottom_border)
   Array2= ScanUpdated (right_border)

   Send (Array1, lower)
   Send (Array2, right)
   Receive (upper, top_array)
   Receive (left, left_array)

//Determine whether to check the upper right corner and lower left corner
  if ( j == C ) DO //C is the corner iteration

/*
  Pi selects those two corner points of (0,h-1) and (h-1,0) and apply Metropolis algorithm on them.
  Pi sends to upper and right neighbor corner (0,h-1)
  Pi sends to lower and left neighbor corner (h-1,0)
  Pi receives from its neighbors corner spins
*/

  Metropolis ((0,h-1));
  Metropolis ((h-1,0));

  Send ((0,h-1), upper);
  Send ((0,h-1), right);
  Send ((h-1,0), lower);
  Send ((h-1,0), left);

  Receive (upper, top_array[0]);
  Receive (left, left_array[0]);
  Receive (lower, bottom_array[h-1]);
  Receive (right, right_array[h-1]);

  ENDIF

j = j+1
```

ENDWhile // While j <= h/4
(Continued from previous page)

ENDFor // One Monte Carlo Step

MC = MC+1

ENDFor // For MC<=Monte Carlo Step

ENDAlgorithm

Function Definitions:

Initialize ()
Initialize the local lattice. Each spin will be randomly assigned a value of +1 or –1.

rand()

Generates a uniform random number from [0,1]

Metropolis ((x, y))
Apply Metropolis Monte Carlo Algorithm (Algorithm 1.1) on the spin S (i,j)
(Continued from previous page)

RandomSelectAndApplyMetropolis (N, From, To)
Uniformly randomly selection N spins on the rectangle specified by upper-left point "From" and lower-right point "To", apply Metropolis Monte Carlo Algorithm (Algorithm 1.1) on each selected spin.

Send (DATA , DESTINATION)
Send the data buffer "DATA" to the processor specified by "DESTINATION"

Receive(SOURCE, BUFFER)
Receive data from the processor specified by "SOURCE" and store the data in "BUFFER"

Compute_Correlation ()
Compute the correlation of the spin system, which is a dynamic property. This function can be replaced by any other customized functions to measure different dynamic properties.

Algorithm 3.3 The α-scheme

3.4.3 Algorithm Analysis of the α-Scheme

This section introduced the performance analysis of α-scheme.

LEMMA 3.10. *Given P processors and assuming a lattice size of $S \times S$, let h be equal to $\frac{S}{\sqrt{P}}$, the communication cost of one Monte Carlo step of the α-scheme is*

$$T_{comm} = h(L + 2o + 35g)$$

where C > 0 is a constant.

PROOF. Because the α-scheme's communication pattern is exactly the same as the FML scheme, the communication cost in α-scheme is the same as that in the FML scheme, i.e.,

$$T_{comm} = h(L + 2o + 35g).$$

LEMMA 3.11. *Given P processors and assuming a lattice size of* $S \times S$ *, let h be equal to* $\frac{S}{\sqrt{P}}$*, the computation cost of one Monte Carlo step of the α-scheme is*

$$T_{computation} = Ch^2 + 9hR.$$

where C > 0 and R > 0 are constants.

PROOF. The basic computation cost is the same as the FML scheme, which is

$$T_{basic\_comp} = Ch^2,$$

where C is the cost of applying the Metropolis algorithm on one spin (this cost includes determining the potential energy change, generating a random number and bookkeeping the internal energy). This is a positive constant.

For the α-scheme, extra costs are incurred, such as the cost for generating a sequence of random number Ext_k and doing bit flippings for each interior and exterior pair. Extra costs are related with random number generation. Let the cost of generating one random number be R. In the worst case, we have *9* exterior spins. The original pair of (2(h-2)I, 9E) can be split into at most *9* small pairs. Each split requires generating one random number. For each small pair, a random number is generated to decide the order of that pair. Thus, there are at most *9 + 9 = 18* random number generations for splitting one original pair. This cost is $18R$.

For each iteration of the α-scheme, the algorithm works on two parts of the lattice (UL part and LR part). Therefore, there are two original pairs to split in one iteration. Thus, we obtain the total extra computation cost of one Monte Carlo step as follows:

$$\begin{aligned} T_{extra\_comp} &= NumOfIterations \times CostPerIteration \\ &= \frac{h}{4} \cdot (2 \cdot 18R) = 9hR. \end{aligned}$$

Thus, the total cost of computation of one Monte Carlo step of the α-scheme is:

$$T_{computation} = T_{basic\_comp} + T_{extra\_comp} = Ch^2 + 9hR.$$

THEOREM 3.12. *Given P processors and assuming a lattice size of* $S \times S$ *, let h be equal to* $\frac{S}{\sqrt{P}}$*, the total cost of one Monte Carlo step of the* α*-scheme is*

$$T_\alpha = Ch^2 + 9hR + h(L + 2o + 35g),$$

where C > 0 and R > 0 are constants.

PROOF. The total cost of one Monte Carlo step of the α-scheme is the summation of the computation cost and communication cost. Thus, from Lemma 3.10 and Lemma 3.11, we have,

$$T_\alpha = T_{computation} + T_{comm} = Ch^2 + 9hR + h(L + 2o + 35g).$$

3.4.4 Comparison With the FML Scheme

Comparing run-time with the FML scheme, the only difference is the extra computational cost related with random number generation. This difference is:

$$T_\alpha - T_{FML} = 9hR.$$

The cost of the FML algorithm is $O(h^2 + h(L + o + g))$. Clearly the difference of $9hR$ is not particularly significant. However, the randomness we achieved through the α-scheme over the FML scheme is significant, and we will provide the analysis in the next section.

3.4.5 Randomness Analysis and Measurement

Thus far, we have presented three schemes: (a) Truly random scheme, (b) FML scheme, and (c) α-scheme. Clearly, the theoretical run-time of sweep selection schemes presented in Chapter 2 is better than any of these 3 random selection schemes' theoretical run-time. We reiterate that random schemes are needed in order to measure dynamic properties. As such, it is now quite important to be able to measure how close each of the 2 schemes (the FML scheme and the α-scheme) is to that of being truly random. This provides guidance on how realistically either method will be in its measurements of dynamic properties. Therefore, in this section, we measure the randomness of all these three schemes by using autocorrelation functions.

Next, we try to model the spin selection process by a series of random numbers. We need to map the selection of one spin to an integer number. In order to do so, we number the lattice sites as integers. Each lattice site is given a unique integer number. For example, for the upper left part of the lattice, we give all exterior lattice sites numbers that are in the range of $[0, 2h-4]$ and all interior lattice sites numbers in the range of $[2h-3, h^2-2h]$. By doing this, all exterior lattice sites are put into one range while all

interior lattice sites are put into another range. We define these numbers as the "site numbers" of spins.

In order to measure randomness, we inserted profiling code into these algorithms. Whenever a spin is selected, the algorithm outputs the site number of that spin. So, the whole spin selection process of that algorithm will generate a series of integer numbers. If the selection scheme is truly random, the integer number series should also be truly random. Randomness of the spin selection scheme can be measured by the randomness of the integer number series it generates.

From statistics, it is known that autocorrelation can be used to measure the randomness of a time series [Box et al. 1976]. The definition of autocorrelation at lag k for a series of N_0, N_1, N_2, …. N_n is:

$$A(k) = \frac{\sum_{i=0}^{n-k} \{(N_i - \bar{N})(N_{i+k} - \bar{N})\}}{\sum_{i=0}^{n} (N_i - \bar{N})^2}$$, where $\bar{N}$ is the mean of all N_i(s).

We take the case where the lattice size is 720×720 and the number of processors is 4. In this case, each processor holds a block of 360×360. According to the FML scheme, it will select 2(360-2) = 716 interior spins and then 8 exterior spins. We expect to see large autocorrelation values at lags 1 to 8 and lags 717 to 725. If we measure autocorrelation for all lags from lag 1 to lag 730, we expect to see some peak values at the head and tail of the plot for the FML scheme. However, for the truly random scheme, we should not see any obvious peak values on the autocorrelation plot. For the α-scheme, we might see some peak values, but there should be significantly less peak values for the α-scheme versus the FML scheme.

Furthermore, if we calculate the average of the absolute values of autocorrelation at all lags, we should be able to observe that the Truly random scheme has the smallest average autocorrelation value while the FML scheme has the largest one.

We profiled all of those 3 schemes and measured autocorrelation values from lag 1 to lag 730. We calculated the average autocorrelation. The numbers are listed below:

Average absolute autocorrelation value for Truly random scheme, FML scheme and α-scheme:

1. *Truly random scheme: 0.002961*
2. *α-scheme: 0.003058*
3. *FML scheme: 0.003413*

The difference between α-scheme and the truly random scheme is 0.000097, while the difference between FML scheme and the truly random scheme is 0.000452. So, the

improvement of randomness of the α-scheme over the FML scheme is $\frac{0.000452 - 0.000097}{0.000452} = \frac{0.000355}{0.000452}$ =78.5%. This implies that the α-scheme achieved significantly more randomness than the FML scheme, while still keeping the message length fixed (our goal).

The following are some autocorrelation plots for all three schemes.

1. Autocorrelation plot for the truly random scheme

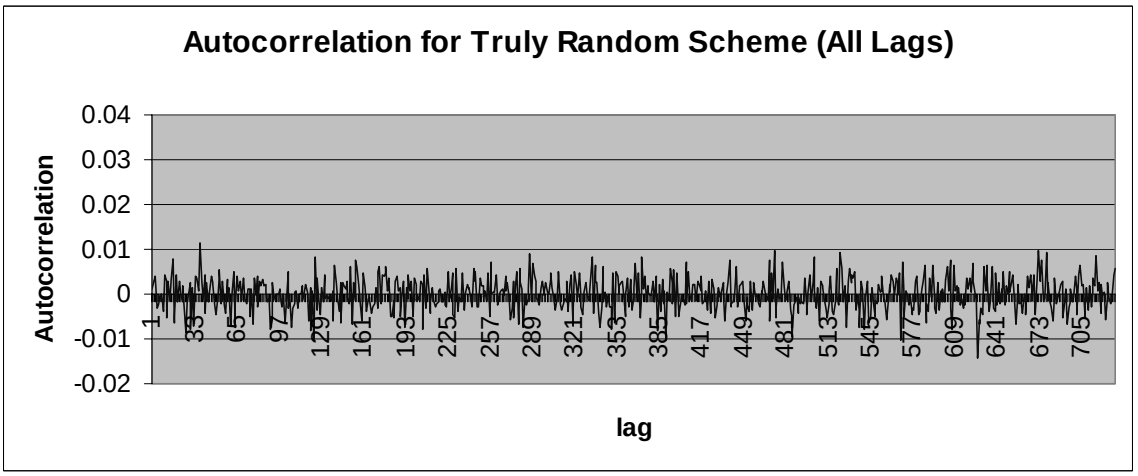


Figure 3.4 Autocorrelation plot for the truly random scheme (all lags)

2. Autocorrelation plots for the FML scheme

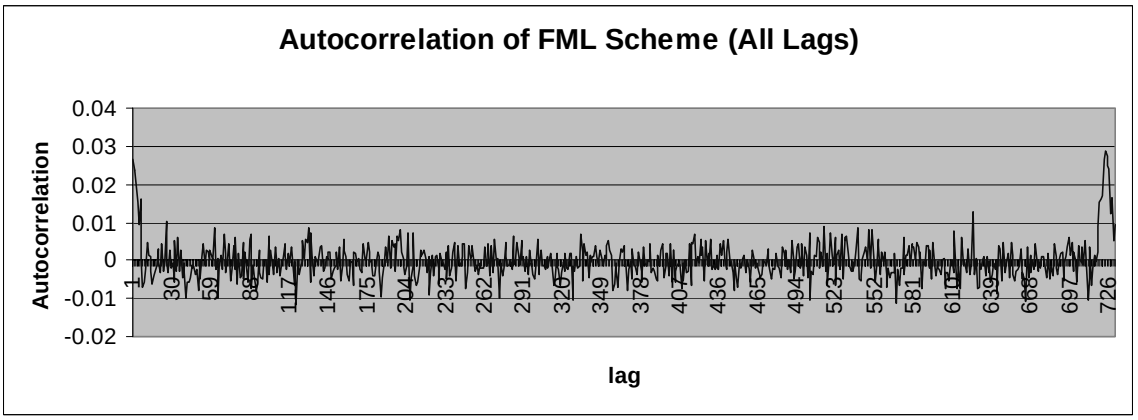


Figure 3.5 Autocorrelation plot for the FML scheme(all lags)

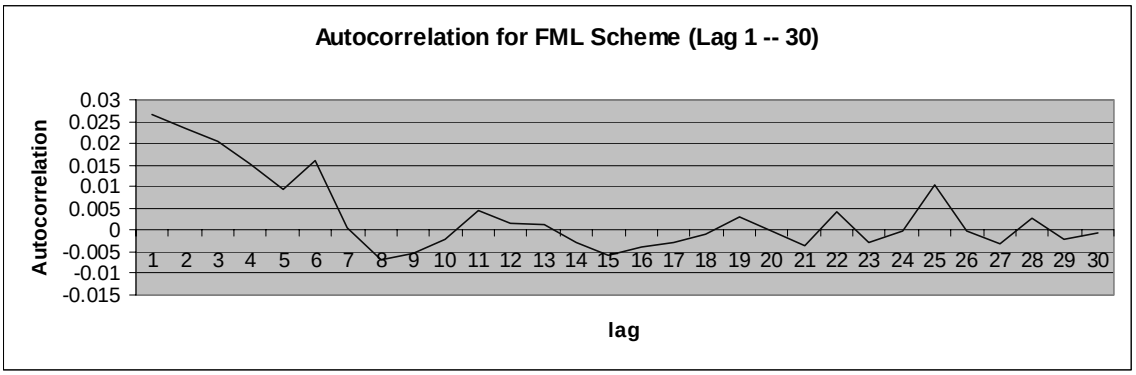


Figure 3.6 Autocorrelation plot for the FML scheme (lags 1–30)

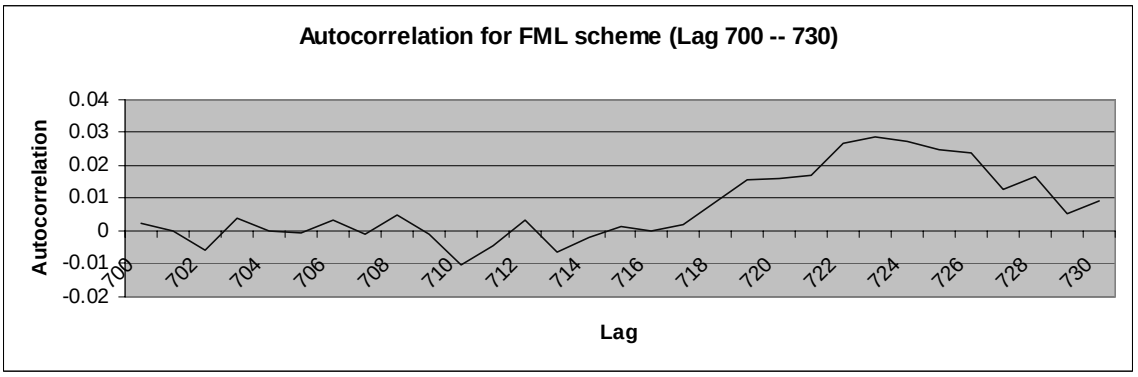


Figure 3.7 Autocorrelation plot for the FML scheme (lags 700-730)

We observed large autocorrelation values at the head and tail of the FML scheme.

3. Autocorrelation plots for the α-scheme

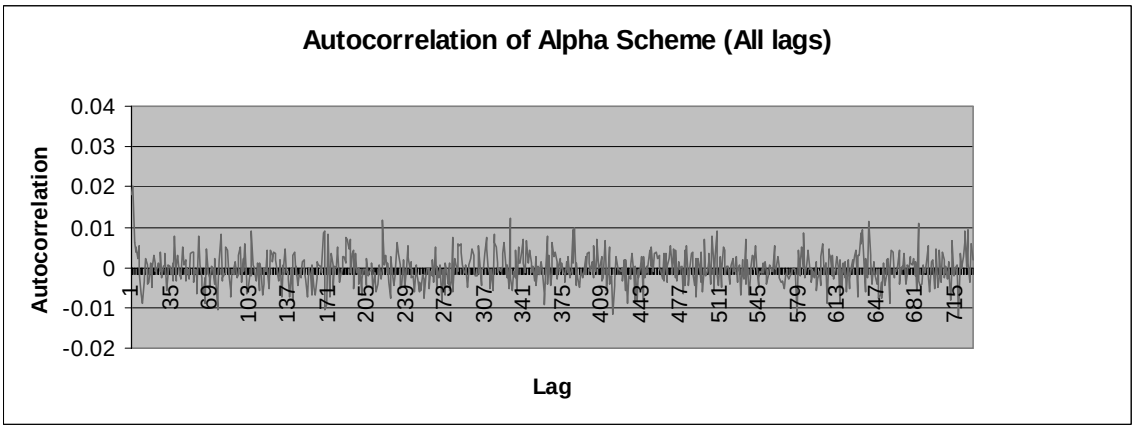


Figure 3.8 Autocorrelation plot for the α-scheme (all lags)

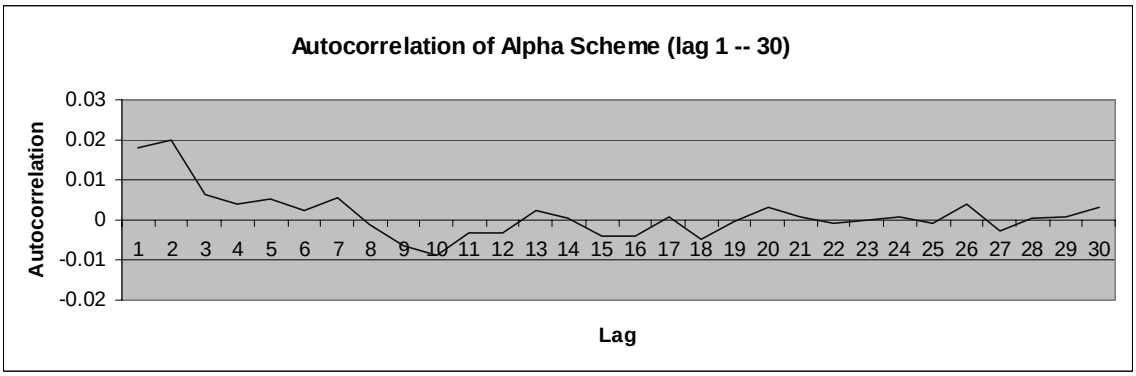


Figure 3.9 Autocorrelation plot for the α-scheme (lags 1-30)

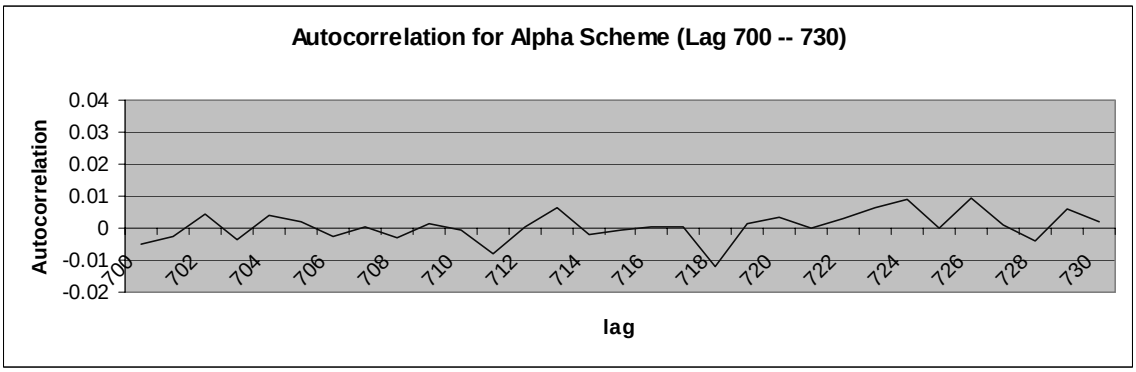


Figure 3.10 Autocorrelation plot for the α-scheme (lags 700-730)

Observing the autocorrelation plots for the α-scheme, we can see that the α-scheme eliminated large autocorrelations at lags of 700 to730 whereas the FML scheme exhibits large autocorrelations. For the head of the plot, the α-scheme eliminated most of the large autocorrelations exhibited by the FML scheme.

3.5 Algorithm Performance Results

We have analyzed the performance of all three schemes on the LogP model in previous sections. We predicted that for sufficiently large lattices the FML scheme is faster than the α-scheme and the α-scheme is faster than the truly random scheme. For small lattice sizes (e.g. 720*720), the difference of run-time between the α-scheme and the FML scheme is negligible, and sometimes FML scheme may be slightly slower than the α-scheme due to network fluctuations. The following are the actual performance data collected using our workstation cluster.

Lattice Size	Truly Random	α	FML
720	34.04	22.61	22.98
1080	70.61	49.25	45.78
1440	126.58	93.00	85.71
1800	226.98	131.12	125.38
2160	280.19	189.32	175.79
2520	391.58	275.91	236.38

Table 3.1 Results for 4 processors (in seconds)

Lattice Size	Truly Random	α	FML
720	25.65	13.37	14.15
1080	56.61	24.66	25.86
1440	94.54	39.66	40.08
1800	156.64	62.20	63.15
2160	221.41	87.15	83.85
2520	277.13	134.83	115.89

Table 3.2 Results for 9 processors (in seconds)

Lattice Size	Truly Random	α	FML
720	15.97	8.15	9.84
1080	39.41	15.77	14.88
1440	61.04	26.23	25.02
1800	86.16	35.10	33.18
2160	125.45	52.55	48.84
2520	172.00	96.97	94.26

Table 3.3 Results for 16 processors (in seconds)

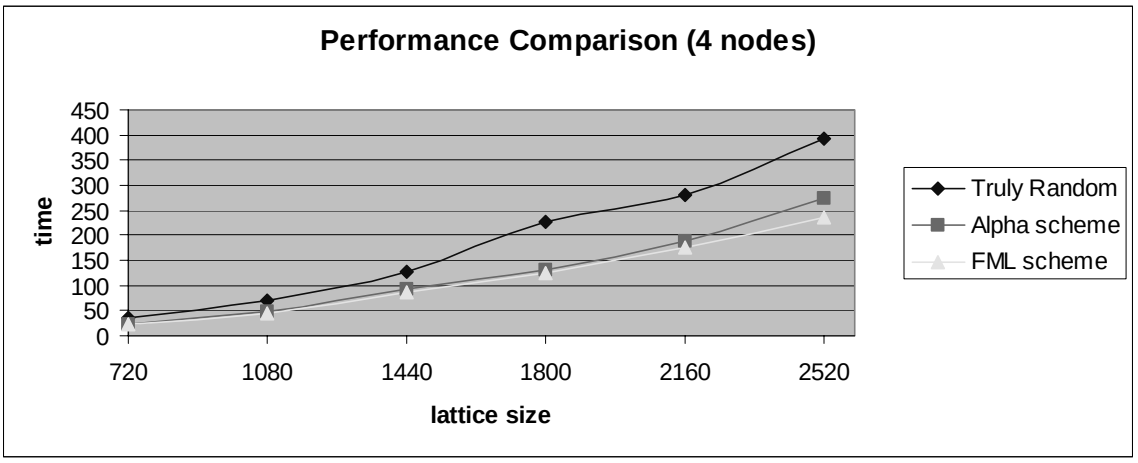


Figure 3.11 Performance comparison (4 processors)

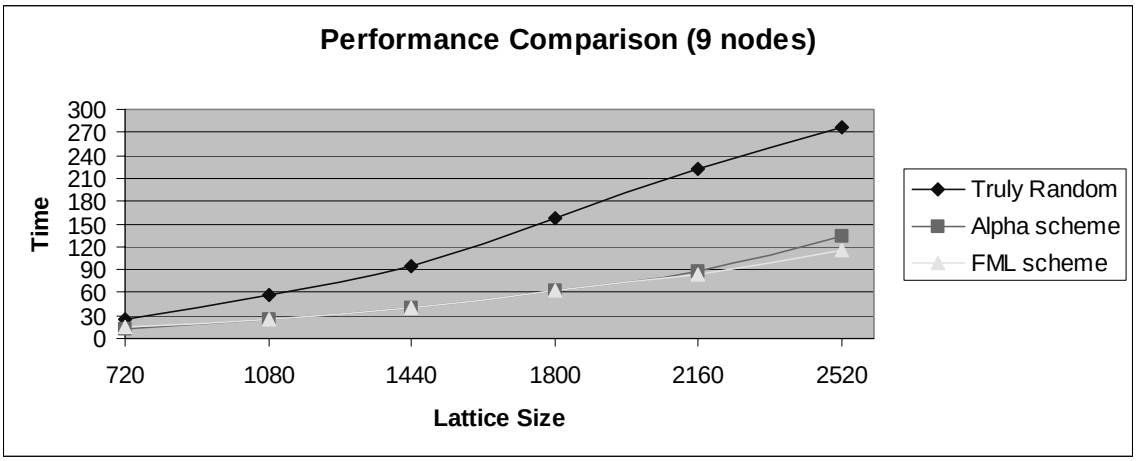


Figure 3.12 Performance comparison (9 processors)

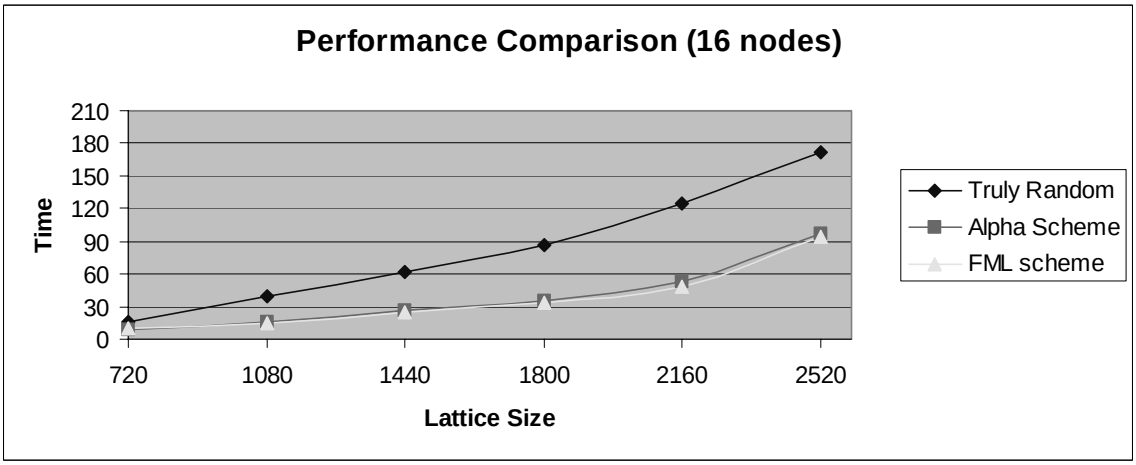


Figure 3.13 Performance comparison (16 processors)

From the performance data, we can see that the truly random scheme is the least efficient one while both the FML scheme and the α-scheme have significant performance improvements over the truly random one. Comparing the α-scheme with the FML scheme, the α-scheme is only slightly slower than the FML scheme at large lattice sizes, i.e. 2.18% for lattice size 2520×2520 on 16 nodes. However, the α-scheme achieved much more randomness than the FML scheme.

Chapter 4 Concluding Remarks

In this thesis, we have studied the parallelization of 2D Ising spin systems. We were able to derive rigorous theoretical analysis of algorithm performance on a practical, general, and portable parallel machine model, namely LogP.

Moreover, we have derived a guiding equation (based on the LogP model) for choosing a better data layout for the sweep selection scheme. This equation can be widely applied to any 2D Ising spin system utilizing sweep selection.

Furthermore, we designed and implemented highly efficient parallel algorithms for random selection schemes. To the best of our knowledge, there has been no previous research in this field to address performance issues for the random selection scheme, particularly on a distributed computing environment. We were able to design efficient communication pattern in our parallel algorithms while still keeping high levels of randomness. We presented our algorithms and made comparisons among them in our study. We also provided rigorous performance analysis for these algorithms on the LogP model.

Finally, performance data was gathered and analyzed which pinpointed design issues and validated the theoretical analyses provided.

To re-iterate, the specific major results in this thesis include:

1) *For measuring static properties of the Ising spin system.*

- Designed and implemented parallel algorithms for sweep selection on both blocked data layout and stripped data layout.
- Analyzed above algorithms on the LogP model, pinpointed a formula for use in choosing a better data layout under certain settings of network parameters.
- Measured the network parameters *L*, *o* and *g* on our workstation cluster, and compared the experimental results with theoretical analyses.
- Verified that the mathematical guiding equation for data layout accurately determines the appropriate data layout.

2) *For measuring dynamic properties of the Ising spin system:*

- Introduced two new random selection schemes; namely FML (Fixed Message Length) and the α-scheme.

    - Designed and implemented parallel algorithms for the FML scheme. The FML scheme has a sequential pattern of spin selection.
    - Developed the α-scheme through modification of the FML scheme in order to achieve better randomness while maintaining efficiency.

- Designed and implemented a truly random selection scheme which does not have a fixed message length. Clearly, it is much less efficient than either the FML scheme or the α-scheme.
- Analyzed the randomness of the FML scheme, the α-scheme, and the truly random scheme by mapping this problem into a statistical time series analysis problem. Measurements of autocorrelation values at various lags were taken and plotted. Results showed that the difference between the α-scheme and the truly random scheme in regards to randomness is negligible.
- Showed that the theoretical results predicted actual performance results.

Based on the investigations performed in this study, similar approaches and methods can be applied to spin systems of three dimensions and higher. Furthermore, parallel algorithms for Q-state Potts model, or other similar spin models can be developed using variations of methods and principles introduced in this study. For statistical physicists or researchers who are interested in measuring various properties of spin systems, they will be able to simply plug in their evaluation formulas into our parallel programs and then obtain the desired results[5].

Furthermore, we note that our work on the parallelization of 2D Ising spin systems showed the validity and usefulness of the LogP model for analyzing and predicting parallel algorithm performance, as well as, determining important network characteristics in order to predict network performance and the design of efficient communication schedules. We observed that the experimental data of our parallel algorithms' run-times match the theoretical analyses of those algorithms on the LogP model. We presented a simple but effective way of measuring network parameters for the LogP model with the presence of system jitters. As stated previously, the approaches and methods developed in this thesis for the design and analysis of efficient parallel algorithms on the LogP model will be of great interest for any computer scientists whose research intersects any area in parallel processing, networking, as well as, scientific computing.

We now conclude with the following observation. Clearly the work in this thesis provides the necessary foundation for realistic parallel schemes for spin systems. In this work, we not only considered fundamental parallel computing issues (such as communication scheduling, networking, run-time efficiency, portability, theoretical modeling, etc.), we also ensured that the major constraints and criteria of 2D Ising spin systems were incorporated into our study. This realism in both fields has rarely been reflected in previous research for this problem. As such, this thesis spotlights the importance of interdisciplinary collaboration between computational and computer scientists in order to obtain realistic results spanning both fields.

[5] For interested researchers, the source code of all parallel implementations performed in this thesis can be obtained via email to the author.

References

[Binder 1997] K. Binder. Applications of Monte Carlo Methods in Statistical Physics. *Reports on Progress in Physics*, 60:487-559.

[Binder et al. 1997] K. Binder and D. W. Heermann. *Monte Carlo Simulation in Statistical Physics*. New York:Springer, 1997. page 24.

[Box et al. 1976] G. E. P. Box and G. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.

[Bultan et al. 1991] T. Bultan and C. Aykanat. Parallel Mean Field Algorithms For the Solution of Combinatorial Optimization Problems. *Proceedings of the International Conference on Artificial Neural Networks*, 1:591-596, 1991.

[Casella et al. 2001] G. Casella and R. L. Berger. *Statistical Inference*. Duxbury, second edition, 2001

[Culler et al. 1996] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken. LogP: A Practical Model of Parallel Computation. *Communications of the ACM*, November 1996.

[Domb et al. 1974] C. Domb and M. S. Green. Ising Model. *Phase Transitions and Critical Phenomena*, 3:357-484, 1974.

[Fisher 1967] M. E. Fisher. The Theory of Equilibrium Critical Phenomena. *Reports on Progress in Physics*, 30:615-730, 1967.

[Gonis et al. 1995] A. Gonis, P. P. Singh, P. E. A. Turchi, and X.-G. Zhang. The Use of the Ising Model in the Study of Substitutional Alloys. *Physics Review*, B51:2122, 1995.

[Gropp et al. 1996] W. Gropp, E. Lusk, N. Doss and A. Skjellum, A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789-828, September 1996.

[Gropp et al. 1996a] W. D. Gropp and E. Lusk. User's Guide for MPICH a Portable Implementation of MPI. Technical Report ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[Heermann et al. 1991] D. W. Heermann and A. N. Burkitt. *Parallel Algorithms in Computational Science*. Springer-Verlag, 1991.

[Ising 1925] E. Ising. *Z. Physik*, 31:253, 1925.

[Karp et al. 1993] R. M. Karp, A. Sahay, E. E. Santos, and K. E. Schauser. Optimal Broadcast and Summation in the LogP Model. *Proceedings of the 5th annual ACM symposium on Parallel algorithms and architectures,* August 1993.

[Kielmann et al. 2000] T. Kielmann, H. E. Bal, and K. Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. *International Parallel and Distributed Processing Symposium*, 2000.

[Matsumoto et al. 1998] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator, *ACM Transactions on Modeling and Computer Simulation,* 8(1):3-30, January 1998.

[Metropolis et al. 1953] N. Metropolis and A. W. Rosenbluth. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics,* 21:1087-1092, 1953.

[Okano et al. 1997] K. Okano, L. Schülke, K. Yamagishi, and B. Zheng. Universality and Scaling in Short-Time Critical Dynamics. *Nuclear Physics B,* 485:727-746, 1997.

[Potts 1952] R. B. Potts. Some Generalized Order-Disorder Transformations. *Proceedings of the Cambridge Philosophy Society,* 48(106), 1952.

[Santos 1999] E. E. Santos. Optimal and Near-Optimal Algorithms for k-Item Broadcast. *Journal of Parallel and Distributed Computing,* 57(2):121-139, 1999.

[Santos 2001] E. E. Santos. A Classification Framework for Parallel & Distributed Algorithm Design. *International Conference on Parallel and Distributed Processing, Techniques, and Applications,* 2001.

[Wang et al. 1990] J.-S. Wang and R. H. Swendsen. Cluster Monte Carlo Algorithms. *Physica A,* 167:565-679, 1990.

[Wolf 1989] U. Wolf. Collective Monte Carlo Updating for Spin Systems. *Physical Review Letters,* 62:361-364, 1989.

VITA

Shuangtong Feng was born on January 25, 1976 in Shanghai, China. He received a BS degree in Computer Science and Application from Shanghai JiaoTong University in 1998. He was a software technical consultant at Computer Associates International Inc. in Shanghai, China. He began graduate studies in Lehigh University in August 1999 and then transferred to Virginia Tech in August 2000. The work reported in this thesis was completed between September 2000 and December 2001. He was a software design engineer intern in Microsoft Inc. from May 2001 to August 2001. He has served as both a teaching assistant and a research assistant during his study in graduate school.

His research interests include parallel and distributed processing, network clusters, performance and computer network.