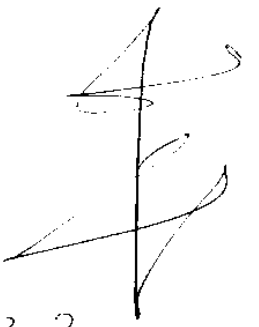
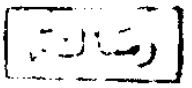
On Parallel Algorithms

Thesis
Submitted in Partial Fulfillment
of the Requirements of the
Award of the M.Sc. Degree
(Computer Science)



Presented By

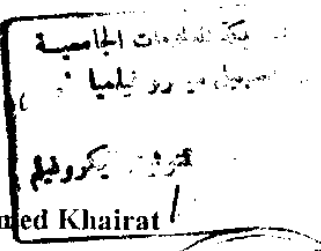
Hazem Mohammed Bahig Abd El-Rahman

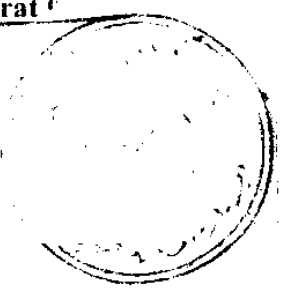


Supervised By

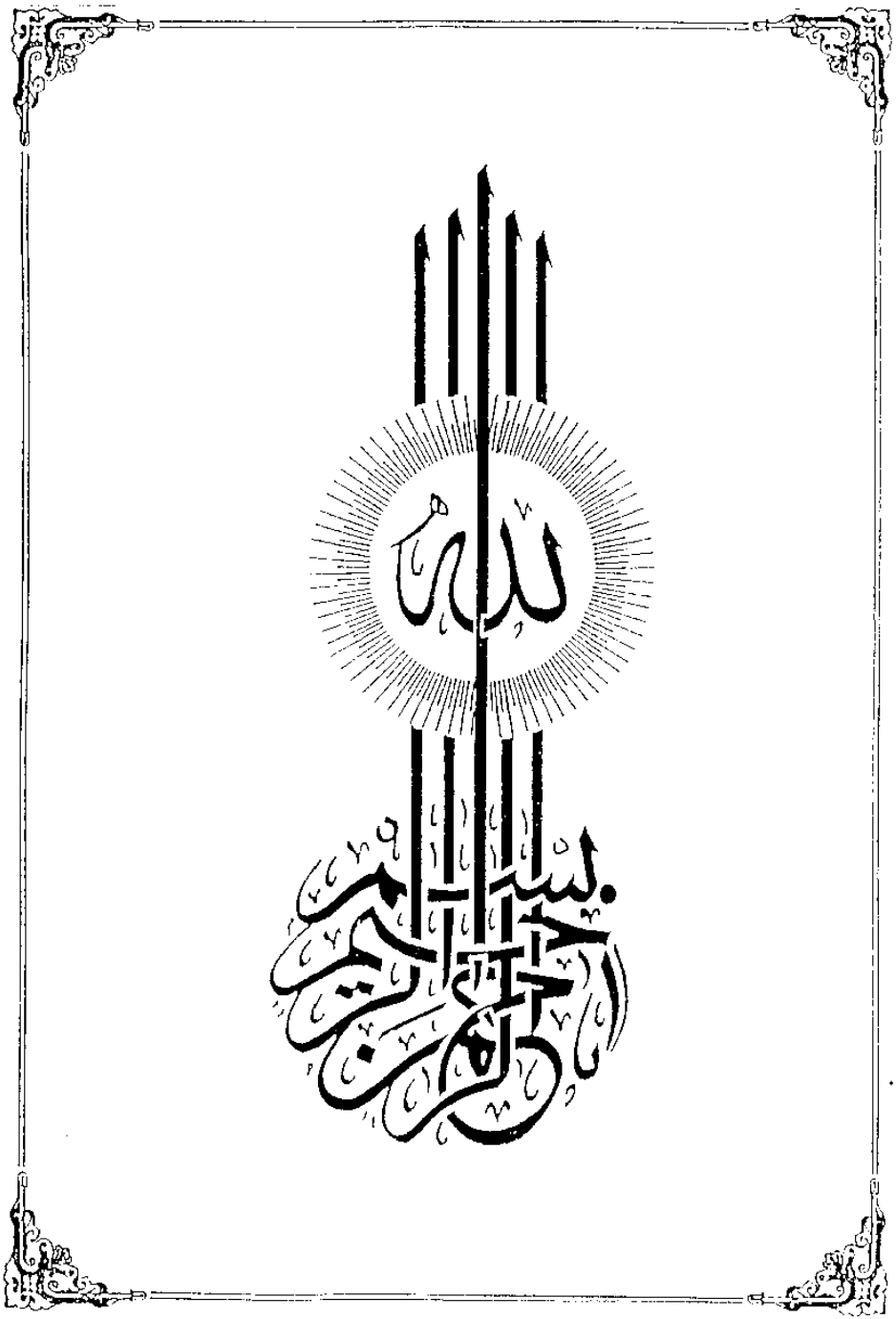
Prof.Dr. Mahmoud Khairat Ahmed Khairat

Department of Mathematics
Faculty of Science
Ain Shams University





Submitted to
Department of Mathematics
Faculty of Science
Ain Shams University
CAIRO, EGYPT
1997



ACKNOWLEDGEMENT

I am deeply thankful to ALLAH for helping me to complete this work.

I would like to thank Prof.Dr. Mahmoud Khairat A. K. for his help and guidance.

I would like to thank Dr. Mohammed H. El-Zhar for this suggestion that led to obtain an improvement in sequential graph coloring alg[illegible]

coloring problem which belongs to the NP-classes. Our result shows how to speed up the time. The improvement was carried out in two steps:

The first step was to obtain a speedup for Berge's sequential algorithm. The second step was constructing a parallel algorithm from the obtained modified sequential one.

In chapter 3, we study the paradigms that can be used to design an efficient parallel algorithm for the internal sorting problem. These paradigms can be used in many computational problems.

In chapter 4, we study the different methods for solving the internal sorting problem and compare between them.

The appendix contains some sequential algorithms needed during the discussion.

INTRODUCTION

Parallelism is a fairly common concept in everyday life. We all tend to think intuitively that two equally skilled people working concurrently can finish a job in half the amount of time required by one person. This is true of many (but not all) human activities [Ak89].

It was natural for people to think of applying the idea of parallelism to the field of computer science. From the dawn of the computer age to this day, computer systems were built that carry out many operations at the same time. Typically, while the central processing unit is busy performing the instructions of a program, a new job is being read and the results of a previous computation are being printed. Recently, however, a new meaning has been given to the concept of parallelism within computers, which is called "parallel processing" that gives high performance processing system [Ak89].

A parallel computer consists of a collection of processing units, or processors, that communicate and cooperate to solve a problem fast. This computer was born due to several reasons:

1. The time required to solve certain problems by a single processor is slow due to technical and physical limitations.
2. The natural solution of many computational problems is a parallel one.
3. The cost and size of computer components (such as VLSI: very-large-scale-integration technology) have declined so sharply

in recent years so that parallel computers with a large number of processors have become feasible.

4. It's possible in parallel processing to select the parallel architecture that is best suited to solve the problem or the class of problems under consideration.

Many applications, that offer sufficient parallelism to keep many processing elements busy simultaneously, fall into the broad categories, numerical processing and symbolic processing, distinguished by the ratio of calculation to data motion among the processing elements.

The numerical processing applications include large scientific and engineering computation. For example, numerical integration, fluid dynamics, quantum chromodynamics, evaluation of galaxies, computer-aided design and many other problems.

The symbolic processing applications include database systems, applied artificial intelligence and many other problems.

Work on parallel computation spans several broad areas, such as the design of parallel models, the design and analysis of parallel algorithms and parallel software (languages and programming environments, compilers and operating systems). The main emphasis of the thesis is the design and analysis of parallel algorithms.

CHAPTER 1

PARALLEL COMPUTATION MODELS

There are many parallel computers each of which may be built as a collection of processors interconnected in a certain fashion dedicated to solving a single problem at a time. We need to select one of those parallel models to design an efficient parallel algorithms. In this chapter we have three sections. The first section studies some taxonomies for parallel models. The second section shows the relations between different parallel models. The third section select the model which will be used to design a parallel algorithm in this thesis.

1.1. Taxonomies of Parallel Computers

A taxonomy is an organization of entities into a hierarchical structure which reflects the relationships between those entities [Wi91]. There are many taxonomies for the parallel computers. The difference between these taxonomies is in the way they describe the parallel computers. For example, describing the computers according to hardware (in relation to kind of communication between the different processors, control unit, and so on), how the applications use hardware and so on. The taxonomies to be studied are:

1.1.1. Processor Interconnections

Parallel computers are classified according to the kind of the communication between the different processors. This communication is very important for several reasons:

1. The processors do not work in total seclusion, since some processors need some data from other processors.

2. The performance of a parallel algorithm may vary greatly under different communication methods.

There exist two kinds of communication between the processors which are shared memory (SM) and non-shared memory (N-SM).

1.1.1.1. SM Model

The first kind of communication is SM which means, roughly, that certain data are shared by many processors [Har87]. The characteristic of SM model is a global memory that can be accessed by all processors. This model consists of p processors numbered 1,.., p. Each processor in this model is the same as a universal sequential machine[1] with a set of global parallel instructions that allow accesses by processors to the memories of other processors, these instructions are read from or written to the SM. The processors may operate synchronously (all processors perform a step of computation at the same time, driven by a local clock) or asynchronously (each processor computes at its own speed). If all processors operate synchronous under the control of a common clock, we call this model ***parallel random access machine*** (**PRAM**). Otherwise, the processors operate under a separate clock and the model is called ***multiprocessors*** or ***tightly coupled*** machine.

The processor executes a sequence of steps, each consisting of the following three substeps:

1. *Read substep*: Each processor p_i reads r_i shared memory locations, where the locations are known at the beginning of the substep. The instruction for read is ***globalread***(x,y) - read the data x from SM and store it in the local variable y in local memory-.
2. *Compute substep*: Each processor p_i performs c_i RAM operations involving only its private state and private memory.
3. *Write substep*: Each processor p_i writes to w_i shared memory locations, where the locations and values written are known at the

[1] In sequential computation, there are standard models of computation such as RAM, RASPM, and TM. All of these models equivalent in computing power.

beginning of the substep. The instruction for we is *globalwrite*(x,y) - write the local data x into the SM location y.

SM model is classified according to types of memory access conflicts:

1. **Exclusive** read/write (**ER,EW**): Each location can be read or written by at most one processor in each unit-time PRAM step.
2. **Concurrent** read/write (**CR,CW**): Each location can be read or written by any number of processors in each unit-time PRAM step.
3. **Queue** read/write (**QR,QW**): Each location can be read or written by any number of processors in each step. Concurrent reads or writes to a location are serviced one-at-a-time.

These three rules can be applied independently to reads and writes; the resulting models are : EREW, CREW, QREW, ERCW, ERQW, CRQW, QRCW, CRCW, QRQW.

In the case of concurrent writing, different assumptions are made about which processor's value is written into the memory location to resolve write conflicts. Some of these policies are:

1. ***Priority rule***: The rule is to assign priorities to processors and, if more than one processor attempts to write to the same memory location, then the one with highest priority will succeed. Without loss of generality, we can assume that priorities are assigned in order of processor index, with highest priorities given to the processor of lowest index. The model with this rule is called **PRIORITY** model.
2. ***Equality (Common) rule***: The rule allows simultaneous writes to the same memory location only if all processors doing so are writing an equal (common) value. The model with this rule is called **COMMON** model.
3. ***Arbitrary rule***: The rule is if more than one processor attempts to write to the same memory location, an arbitrary one will succeed. The model with this rule is called **ARBITRARY** model.

Remark 1.1: There exist another types of PRAM such as parallel random access machines with **Owned global memory**

(CROW)[KRS88], **Collision CRCW** model [FRW88], and **Week** and **Strong CRCW** [EG88].

1.1.1.2. N-SM Model

The second kind of communication between the processors is N-SM. The characteristic of N-SM model is that there is no SM, but its memory is partitioned into p modules, one associated with each processor; for this characteristic the model is called ***distributed memory*** model. Each processor communicate through an interconnection network consisting of communication links in two-way joining certain pairs of processors; for this characteristic, the model is called ***network*** model. The processors may operate synchronously or asynchronously.

There exist two instructions used to communicate certain processor with another which are:

Send(x,i) which means that a processor p executing the send instruction sends a copy of x to a processor p_i, then resumes the execution of the next instruction immediately [Ja92] and [KRS88].

Receive(x,i) which means that a processor p executing the receive instruction suspends the execution of its program until the data from a processor p_i are received. It then stores the data in x and resumes the execution of its program [Ja92] and [KRS88].

There are several N-SM models according to the different connections between the processors. These models can be classified into two kinds. One of them is used in a wide variety of applications which is called ***general-purpose architecture*** and the other is used to solving certain problems which is called is ***special-purpose architecture***. The N-SM model can be represented by a graph $G = (V,E)$ where V is the set of vertices, each vertex $i \in V$ represents a processor p_i and E is the set of edges, each edge $(i,j) \in E$ represents a two-way communication links between processor p_i and p_j. Now, we study some of these models.

1.1.1.2.1. Fully Connected Computer

Fully connected computer (**FCC**) consists of p processors indexed from 0 to p-1 forming a complete graph, see figure 1.1.

Remarks 1.2:

1. This model can be viewed as SM model, i.e. M locations of the SM are distributed among the p processors , each receiving M/p locations.
2. This model is feasible for small number of processors. Otherwise, the model is unrealistic in practice, because the number of connections to a processor is limited [Ak89].

1.1.1.2.2. Tree Connected Computer

A complete binary tree connected computer (**TCC**) consists of $p = 2^k-1$ processors indexed from 0 to p-1, where $2 \leq k$, see figure 1.1 (b).

Remarks 1.3:

1. TCC have been proposed to support the parallel execution of algorithms for searching, sorting, image processing and other algorithms amenable to divide-and-conquer approach [Du92].
2. There exist many tree-structures that may be used other than a complete binary tree. For example star computer, see figure 1.1 (c).

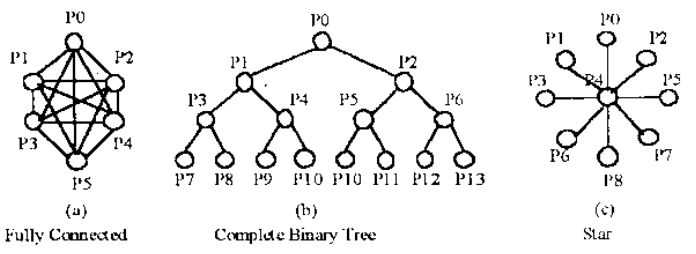
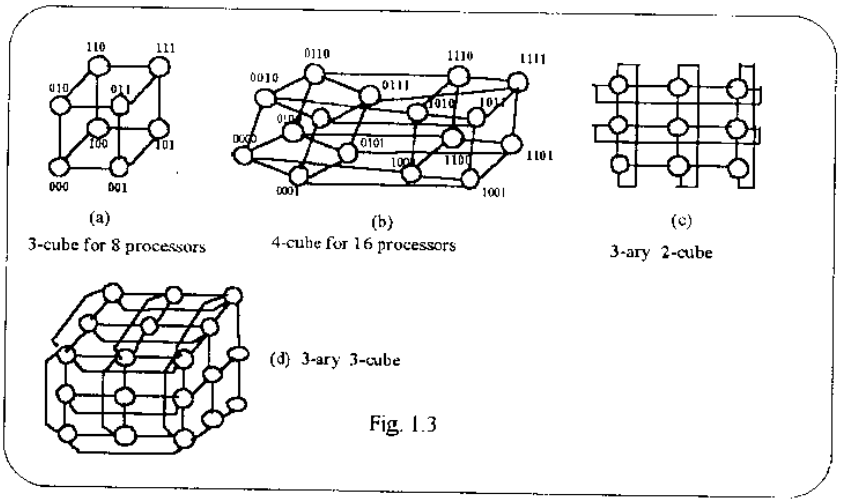


Fig. 1.1

dimension consists of k processors. Each processor p_i is identified by an n-digit radix k number, the bth digit of the number represents the vertices position in the bth dimension. The processor p_i whose k-radix representation is $i_{n-1}...i_0$ is connected to the processors p_j if the index j represented by k-radix representation as $i_{n-1}\, i_{n-2}\, ...i_{b+1}\, i_b^+\, i_{b-1}...i_0$ and $i_{n-1}\, i_{n-2}\, ...i_{b+1}\, i_b^-\, i_{b-1}...i_0$, $\forall\ 0 \le b \le n-1$, where $i_b^+ = (i_b + 1)$ MOD k and $i_b^- = (i_b - 1)$ MOD k. For example see figure 1.3 (c) and (d).



(a) 3-cube for 8 processors

(b) 4-cube for 16 processors

(c) 3-ary 2-cube

(d) 3-ary 3-cube

Fig. 1.3

1.1.1.2.5. Shuffle-Exchange Connected Computer

p-Shuffle-exchange connected computer (**SECC**) consists of $p=2^k$, $k \ge 1$, processors indexed from 0 to p-1. The SECC is based on two kinds of interconnection between the processors. The first kind of interconnection is ***shuffle*** function. In shuffle function the processor p_i is connected to the processor p_j

$$j = \begin{cases} 2i & \forall\ 0 \le i \le p/2 - 1 \\ 2i+1-p & \forall\ p/2 \le i \le p - 1 \end{cases}$$

If the index i is represented by binary as $i = i_{k-1}\, i_{k-2} \ldots i_0$, $\forall\ 0 \le i \le p-1$, then the shuffle function is defined as $S(i_{k-1}\, i_{k-2} \ldots i_0) = i_{k-2} \ldots i_0\, i_{k-1}$. The

second kind of interconnection is ***exchange*** function. In exchange function the processor p_i is connected to its successor in two-way communication links if the index i is even. If the index i is represented by binary as $i = i_{k-1}\ i_{k-2} \ldots i_0, \forall\ 0 \le i \le p\text{-}1$, then the exchange function is defined as $E(i_{k-1}\ i_{k-2} \ldots i_0) = i_{k-1}\ i_{k-2} \ldots i_0^{-}$, where i_0^{-} is the complement of i_0.

The graph representation of SECC is

- the set of vertices V corresponds to all processors such that each vertex is represented as binary string of length k.
- the set of edges E, where two vertices w and w' are connected by edge if either of them the vertex w' is obtained by cyclically shifting of w, i.e. if $w = xa$ then $w' = ax$ where x is a binary string of length k-1 and $a \in \{0,1\}$ - this edge is called *shuffle edge*, or the vertex w' differ w in the least significant bit, - this edge is called *exchange edge*. See figure. 1.4 (a).

Remarks 1.6:

1. Sometimes the SECC is called ***perfect shuffle*** computer.
2. In some interconnection networks there exist another interconnection which is called ***shuffle-exchange*** interconnection. If the index i is represent by binary as $i = i_{k-1}\ i_{k-2} \ldots i_0, \forall\ 0 \le i \le p\text{-}1$, then the shuffle-exchange function is defined as $SE(i_{k-1}\ i_{k-2} \ldots i_0) = i_0^{-}\ i_{k-1}\ i_{k-2} \ldots i_1$ where i_0^{-} is the complement of i_0. The computer which is based on the shuffle-exchange and shuffle interconnections is called ***DeBruijn*** connected computer (**DBCC**). The set of vertices V correspond to all processors such that each vertex represent as binary string of length k. The set of edges E, where two vertices w and w' are connected by edge if either of them the node w' is obtained by cyclically shifting of w, i.e. if $w = xa$ then $w' = ax$ where x is a binary string of length k-1 and $a \in \{0,1\}$- this edge is called *shuffle* edge, or the vertex w' is obtained by shuffle-exchange of w, i.e. if $w = xa$ then $w' = bx$ such that $b \neq a$ - this edge is called *exchange edge*. See figure. 1.4 (b).

model of computer can execute several independent programs simultaneously.

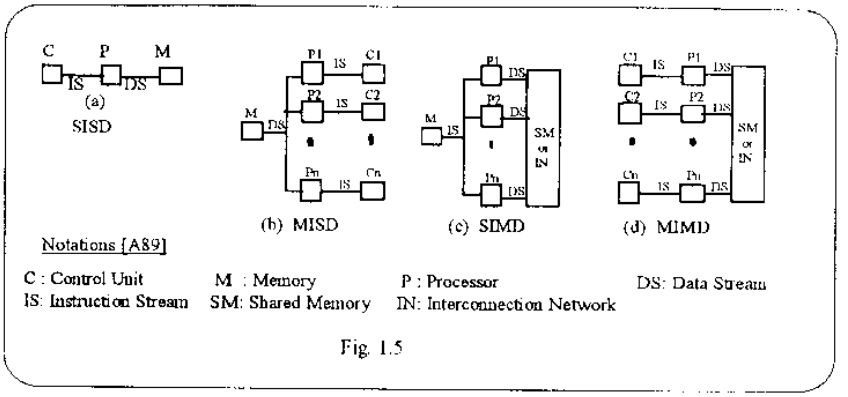


Fig. 1.5

1.1.4. Wilson's Taxonomy

Wilson [Wi91] describe the serial and parallel computers according how they are used, rather than on what they contain. This taxonomy based on the following criteria.

1. *The number of programs* which the architecture can be execute simultaneously (***P***). This is measured by the arity of the next-instruction function which takes a set of possible instructions and an execution state and returns the next instruction code(s). The term "program" will always be taken to mean something which runs on a single processor, and whose next-instruction function must be unary; the term "multiprogram" will be used for collection of programs running cooperatively on the same or separate processors, whose collective next-instruction function has an arity greater than one.
2. *The number of execution states* which the architecture can maintain simultaneously (***S***). An execution state is an instruction pointer indicating a program location, plus variables which the programmer thinks of as being locally-scoped, short-lived, and private.