

**THE USE OF DATAFLOW ARCHITECTURES
FOR IMPROVEMENT OF SORTING
ALGORITHMS**

By

RENMEI SHU

Diploma

Beijing Institute of Technology

Beijing, China

1975

**Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfilment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 1990**

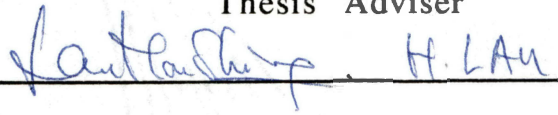
Thesis
1990
S562u
cop 2

THE USE OF DATAFLOW ARCHITECTURES
FOR IMPROVEMENT OF SORTING
ALGORITHMS

Thesis Approved:



Thesis Adviser







Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to express my most deep and sincere gratitude to my major adviser, Dr. George E. Hedrick, for his guidance and support in completing my graduate study, and for his valuable observations and advisement throughout my thesis research.

I also extend my heartfelt thanks to my other committee members, Dr Han-Shiang Lau and Dr. Huizhu Lu, for their helpful advisement and suggestions.

Finally, I would like to express my appreciation to my father, Songgui Shu, and my mother, Yongru Peng, for their support and encouragement. I am also grateful to my husband, Xuzhi Zhang, my daughter, Jie Joy Zhang, and my brother, Junde Shu, for their moral support and patience.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
Background	1
Project Motivation.....	3
Terminology.....	4
Problems Described and Discussed.....	5
II. PRELIMINARY LITERATURE REVIEW.....	6
Control Flow, Dataflow and Reduction.....	6
Static and Dynamic Dataflow Architectures.....	15
Sorting Algorithms.....	17
III. THE RELATIONSHIPS AMONG DATAFLOW ARCHITECTURES AND SORTING ALGORITHMS	22
Basic Concepts.....	22
Computation Organization.....	23
Program Organization.....	27
Machine Organization	32
IV. DATAFLOW SORTING MACHINES.....	40
Special-Purpose Sorting Machines.....	40
General-Purpose Sorting Machines.....	46
Dataflow Database Machines	50
V. SUMMARY, CONCLUSIONS, AND SUGGESTED FUTURE RESEARCH	54
Summary and Conclusions.....	54
Suggested Future Research.....	56
BIBLIOGAPHY.....	57

LIST OF FIGURES

Figure	Page
1. Instruction Execution in a Control Flow Computer.....	8
2. Instruction Execution in a Data-Driven Computer.....	10
3. String Reduction Program.....	12
4. Graph Reduction Program.....	14
5. A Static Dataflow Computer Organization.....	16
6. A Dynamic Dataflow Computer Organization.....	18
7. The Computation Rules Applied to an Expression.....	24
8. Computation Mechanism of Control Flow, Dataflow, and Reduction Program Organization.....	28
9. The Basic Format of a Reference in Dataflow.....	30
10. Centralized Machine Organization.....	33
11. Packet Communication Machine Organization.....	35
12. Expression Manipulation Machine Organization.....	36
13. Dataflow Packet Communication with Token Storage.....	37
14. Dataflow Packet Communication with Token Matching... ..	38
15. A Example Using Parallel Tree Selection Sort.....	42
16. The Sorting-Merging Processors in a Parallel Tree.....	43

Figure	Page
17 A Example of Parallel Binary Merging Sort	44
18. The Dataflow Graph of a Sorting-Merging Processor	45
19. The Outline of a Special-Purpose Dataflow Parallel Tree Sorting Machine.....	47
20. The DPTSM1 Machine.....	48
21. The Outline of a General-Purpose Dataflow Sorting Machine.....	49
22 The GPDSM1 Machine.....	51

CHAPTER I

INTRODUCTION

Background

A very basic computer architectural principle, which was proposed by von Neumann in 1945, has been used successfully for almost 40 years. The development of von Neumann computers has grown amazingly. Very Large Scale Integration (VLSI) technology has enormously improved the processor capabilities and drastically reduced the cost of implementing a CPU.

The growth rate never has satisfied completely the newer and more complex applications developed for computer systems. The nature of both data and processing tasks is changing. Vast quantities of nonnumeric data, such as sentences, symbols, speech, graphics and images are handed. Processing requirements are becoming more demanding with artificial intelligence applications than scientific calculation [47]. Some difficulties have surfaced from the historical computer design which uses only one CPU. Two main difficulties are in the realms of software programming and processor performance [33]. A semantic gap exists between the von Neumann programming languages and the problems in the real world. The approaches to increase the speed of conventional architectures have reached their limit and fail to take advantages of electronic technology.

However radical advances in technology and programming languages continue to open new possibilities for computer architectures. Progress toward some new computer architectures is expected to be made quickly [47].

One kind of architectures under consideration is parallel von Neumann machine which is a number of sequential processors tied to a single memory (shared memory) or each with its own memory (distributed memory). This architecture has attracted wide interest, but has many drawbacks. Interdependencies of instructions in programs reduce the opportunity for a conventional multiprocessor to attain a high level of concurrence. Interdependencies between address spaces of processes causes a processor-memory interconnection problems.

Conventional languages such as Fortran are based on a global state model of computer operation, these languages are unsuitable for the next generation of computers, and can eventually be abandoned for VLSI scientific computation. Now functional, or applicative, programming languages and dataflow models of computation are the only known foundation appropriate for a computer base language [20].

A possible solution to the problem of efficiently exploiting concurrence of computation on a large scale is dataflow architectures, which show promise of making use of VLSI and parallelism. They are compatible with modern concepts of program structure; therefore, they should not suffer from the difficulties of programming that have hampered other approaches to highly parallel computation [20].

Highly concurrent computation is a natural consequence of the dataflow concept [20]. In a dataflow computer, an instruction is ready for execution as soon as all its operands have arrived. There is no concept of control flow. Dataflow computers do not have program location counters. Many instructions of a dataflow program may be available for execution at once.

There exists a widespread agreement that future generation computers should have at least the following features [49]:

i) high parallelism and scalability exploiting the potentialities of VLSI (Very Large Scale Integration) and communication technologies;

ii) efficient and reliable support to very high level programming languages, logic and/or functional and object oriented.

Many computer scientists believe that the next generation of computers will be based on a non-von Neumann architecture. The fifth-generation architectures possible include both data-driven and reduction computers [48].

Because the research topic of this thesis is based on the dataflow architectures, in the next chapter the dataflow architectures will be discussed in detail.

Project Motivation

The idea of data-driven computation is old [30] [41]. Only in recent years have architectural schemes developed because of their attractive anticipated performance and their capability of supporting general user languages. Research on data flow is in progress in at

least a dozen of laboratories in the US, Europe, and Japan. Several dataflow computers have been built, and more hardware projects are being planned.

Sorting is very frequently a fundamental data processing step. Much effort has been devoted to improve sorting algorithms because of its practical importance as well as its theoretical interest [28]. Sorting may be one of the first large scale uses of dataflow in the real world. The use of dataflow architectures for performance improvement of sorting is a very attractive area of research in computer science. It can accelerate development of dataflow computer from research period to its practical application period.

“Do the dataflow architectures improve the sorting algorithms?” and “How to improve the sorting algorithms on the new architectures?” are the new research problems which confront us.

Terminology

Control-driven computers: a computer in which an instructions is ready for execution as soon as it is selected by program counter.

Data-driven computers: a computer in which an instruction is ready for execution as soon as all its operands have arrived.

Demand-driven; i.e , reduction computers: a computer in which the requirement for a result triggers the operation that will generate it.

Computation rule: the rule which selects a subset of instructions in the program for possible execution.

Firing rule: the rule for making a decision whether to execute an instruction based on examination of each instruction's actual arguments.

Static dataflow architecture: a dataflow architecture which allows at most one token per arc in a dataflow graph.

Dynamic dataflow architecture: a dataflow architecture which tags each token and keeps them in a common pool of storage.

Sorting: a process to arrange items in a set according to a predefined ordering relation.

Internal sorting algorithms: sorting algorithms which arrange data in main memory.

External sorting algorithms: sorting algorithms which arrange data in external storage.

Problems Described and Discussed

Chapter II gives a preliminary literature review. Chapter III contains the relationships among dataflow architectures and sorting algorithms. In Chapter IV two dataflow sorting machines are designed and the dataflow database machines is introduced. Chapter V present the summary, conclusions and suggested future research.

CHAPTER II

PRELIMINARY LITERATURE REVIEW

About 20 years ago, R. M. Karp and R. E. Miller [30], J. Rodriguez at MIT and D. Adams at Stanford [41] begin to work on research that eventually led to the development of concepts in dataflow systems [1]. The first designs by J. B. Dennis [21] and J. Rumbaugh [42] were made at MIT. The first dataflow computer began work in July, 1976 [19]. Important advances have been made since that time. Many researchers are investigating dataflow concepts as an alternative to von Neumann systems and languages. Comparison of dataflow computers are in [20], [26] and [44]. Excellent surveys can be found in [48], [51] and [52].

Control Flow, Dataflow and Reduction

There are two types of data flow architectures: data-driven and demand-driven (reduction) [48]. Control flow computation requires a different approach in each of these two types of data flow architectures. Their simple operational computation are presented in this section.

Control Flow Computation

Until recently, most computers used very basic architectural principles proposed by von Neumann in 1945.

The concepts of control flow and data flow computing are distinguished by the control of computation sequences in two distinct program representations [45]. The control flow program representation of the statement $Z=\min(a,b,c,d)$ is shown in Figure 1. In the traditional sequential control flow model, there is a single thread of control which is passed from instruction to instruction (Figure 1a). Explicit control transfers are caused by using operators such as GOTO. In the parallel control flow model (Figure 1b), Special parallel control operators such as FORK and JOIN are used to specify parallelism explicitly. The operators allow more than one thread of control to be active at an instant, and to provide means for synchronizing these threads, as demonstrated in Figure 1b. Special features are identified below for both the sequential and the parallel control flow model [29] [48]:

(1) Data is passed between instructions via references to shared memory cells.

(2) Flow of control implicitly is sequential, but special control operators can be used explicitly for parallelism.

(3) Program counters are used to sequence the execution of instruction in a centralized control environment.

(4) The flows of data and control are separate, they can be made identical or distinct.

Control flow computers have a control-driven organization that the program has complete control over instruction sequencing. Synchronous computations are performed in control flow computers using centralized control.

Dataflow Computation

In a data-driven computation, instructions are activated by the availability of data tokens as indicated by the black dots in Figure 2. Data flow programs are represented by directed graphs, which show the flow of data between instructions. Each instruction consists of an operator, one or two operands, together with one or more destinations to which the result (data token) will be sent. Five interesting features in the data flow model are listed below [29] [48]:

(1) Intermediate or final results are passed directly as data tokens among instructions.

(2) There is no concept of shared data storage as embodied in the traditional notion of a variable.

(3) Program sequencing is constrained only by data dependency among instructions.

(4) Execution consumes data tokens. The values are no longer available as inputs to this or any other instruction.

(5) Flows of control are bound to the flow of data.

Data flow computers have a data-driven organization that is characterized by a passive examination stage.

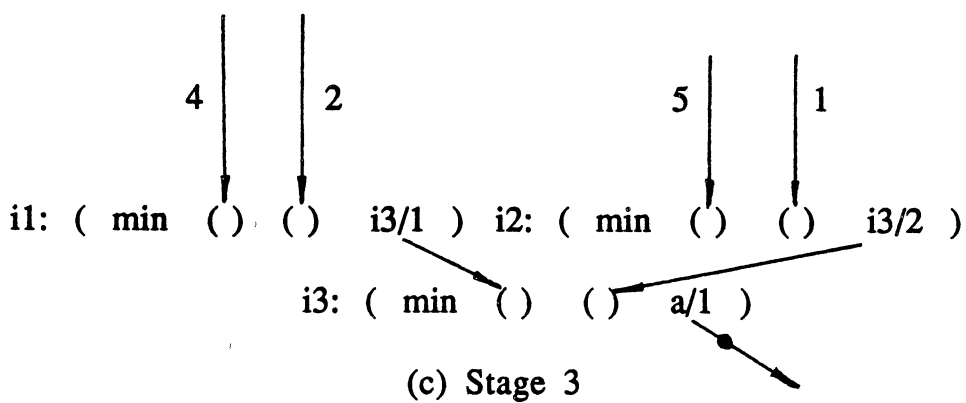
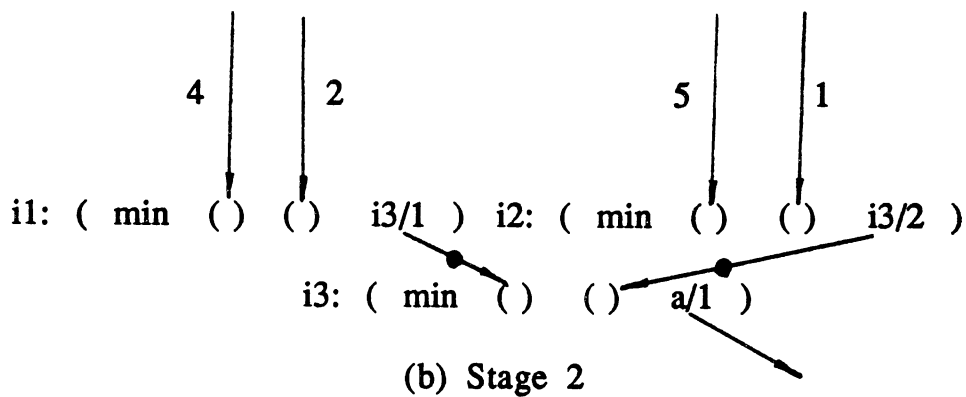
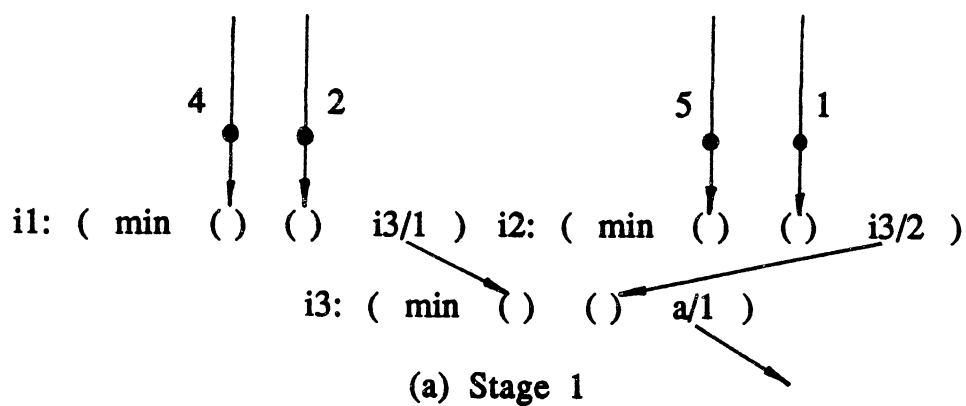


Figure 2. Instruction Execution in a Dataflow Computer for $Z=\min(a,b,c,d)$

Instructions are examined to reveal the operand availability, upon which they are executed immediately as soon as the functional units are available.

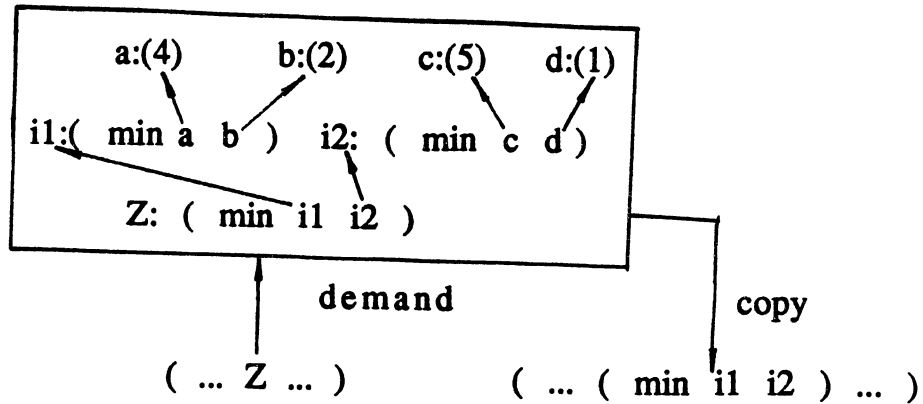
Reduction Computation

The demand-driven architectures have their foundation in functional languages. These languages are based on the lambda calculus. Programming is effected by using them to write mathematical equations rather than by conventional programming. The expressions represented as graphs are reduced by evaluation of their branches or sub-graphs. The reduction is done only when the result of the sub-graph is required; that is, on demand. Different parts of the graphs can be reduced or evaluated in parallel.

There are two reduction models [45]: string reduction and graph reduction. Both forms of recurrence have a recurrent control mechanism.

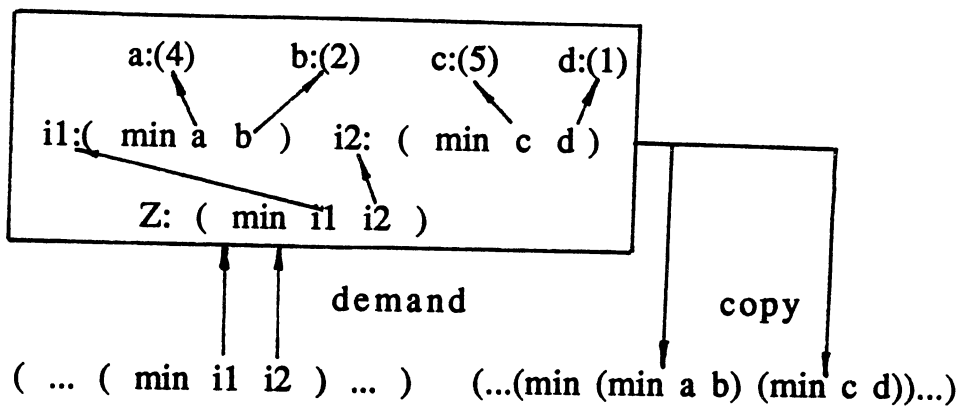
String reduction has a "by value" data mechanism. In string reduction, the instruction accessing a particular definition will make and manipulate a separate copy of the definition. Each instruction consists of an operator followed by literals or embedded references used to demand the corresponding input operands. The example (Figure 3) shows the evaluation of the definition a by using string reduction. The reference, Z, is overwritten by the definition. Next, the operation of a minimum operator is suspended while its arguments I1 and I2 are evaluated. Finally, the expression is said to

definition



(a) Stage 1

definition



(b) Stage 2

====> (... (min (min 4 2) (min 5 1)) ...)

====> (... (min 2 1) ...)

====> (...1...)

(c) Stage 3 to 5

Figure 3. String Reduction Program
for $Z = \min(a, b, c, d)$

be reduced when all the arguments of the expressions are replaced by literal values and the expression is evaluated.

In graph reduction, the instruction accessing a particular definition manipulates references to that definition. The arguments are by references, using pointers unlike string reduction by values. In the example (Figure 4), some instruction demands the value associated with Z, but, instead of making a copy of the definition, the reference is traversed until it is reduced and value is returned. One way to identify the original source of the definition is to embed a reference in the definition. This traversal of the reference is continued until the expression is reduced and the value is returned.

The main features of reduction are [48]:

(1) program structures, instructions, and arguments all are expressions;

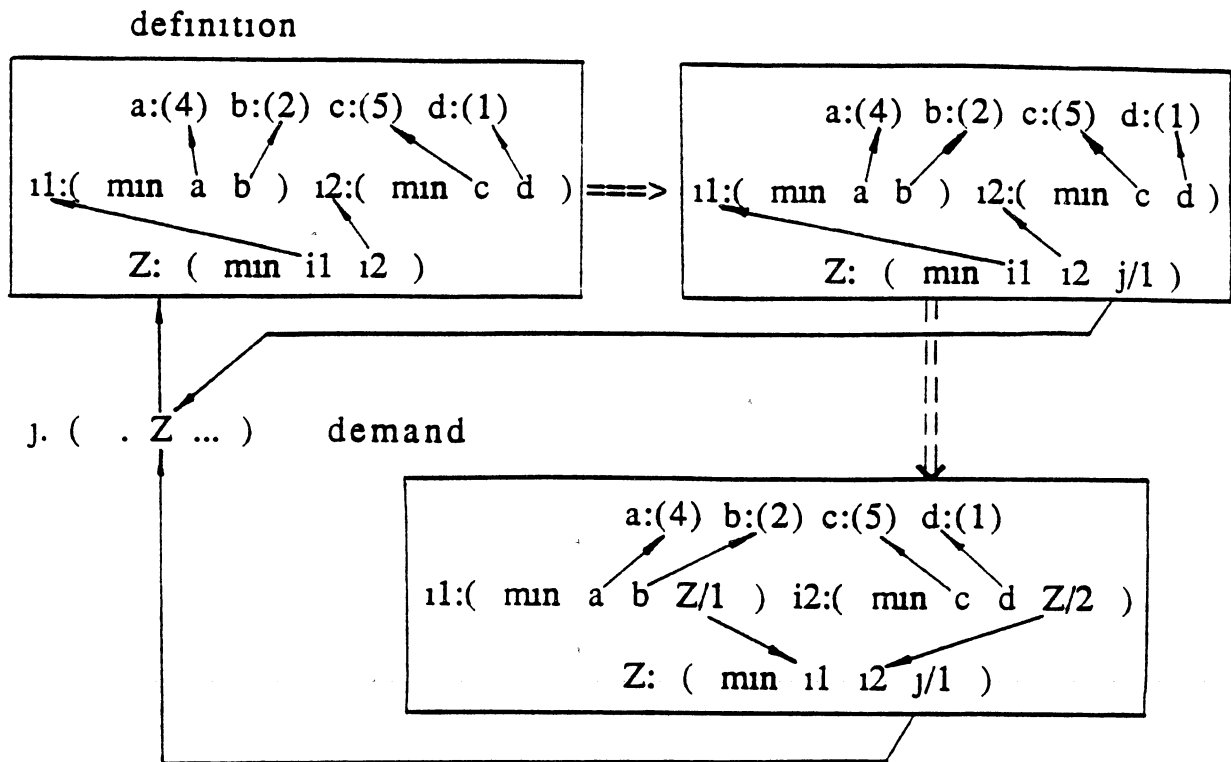
(2) there is no concept of data storage (variables may exist but are not necessarily associated with a storage location);

(3) there are no additional sequencing constraints over and above those implied by demands for operands;

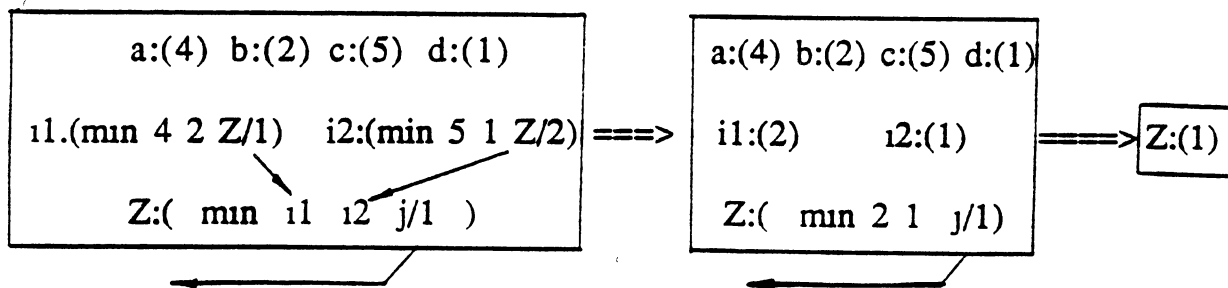
(4) demands may return simple or complex arguments such as a function (as input to a higher-order function).

Control Flow vs. Dataflow

The concept of dataflow systems is different from the concept of conventional von Neumann systems. Dataflow computers operate asynchronously without sequential control and use a distributed memory instead of a single updatable memory.



(a) Stage 1 to 3



(b) Stage 4 to 6

Figure 4. Graph Reduction Program for $Z = \min(a, b, c, d)$

The fundamental difference in the two families of architectures is that instruction execution in a conventional computer is under program-flow control, whereas in a data flow computer is driven by the availability of data.

Static and Dynamic Dataflow Architectures

Static and dynamic dataflow architectures are two distinct types of implementations of the abstract dataflow model [8].

Static Dataflow Architecture

Static dataflow allows at most one token per arc in dataflow graphs. It provides a fixed amount of “storage” per arc. A static dataflow computer organization is shown in Figure 5.

In a static dataflow machine data tokens move along the arcs of the data flow program graph to the operator nodes. The nodal operation is executed when all its operand data are present at the input arcs. Only one token is allowed to exist on any arc at any given time; otherwise, the successive sets of tokens cannot be used to acknowledge the proper timing in transferring data tokens from node to node [29].

Dynamic Dataflow Architecture

Dynamic architectures tag each token and keeps it in a common storage pool. They provide dynamic allocation of token storage from the common pool. Tokens carry tags to indicate their logical position

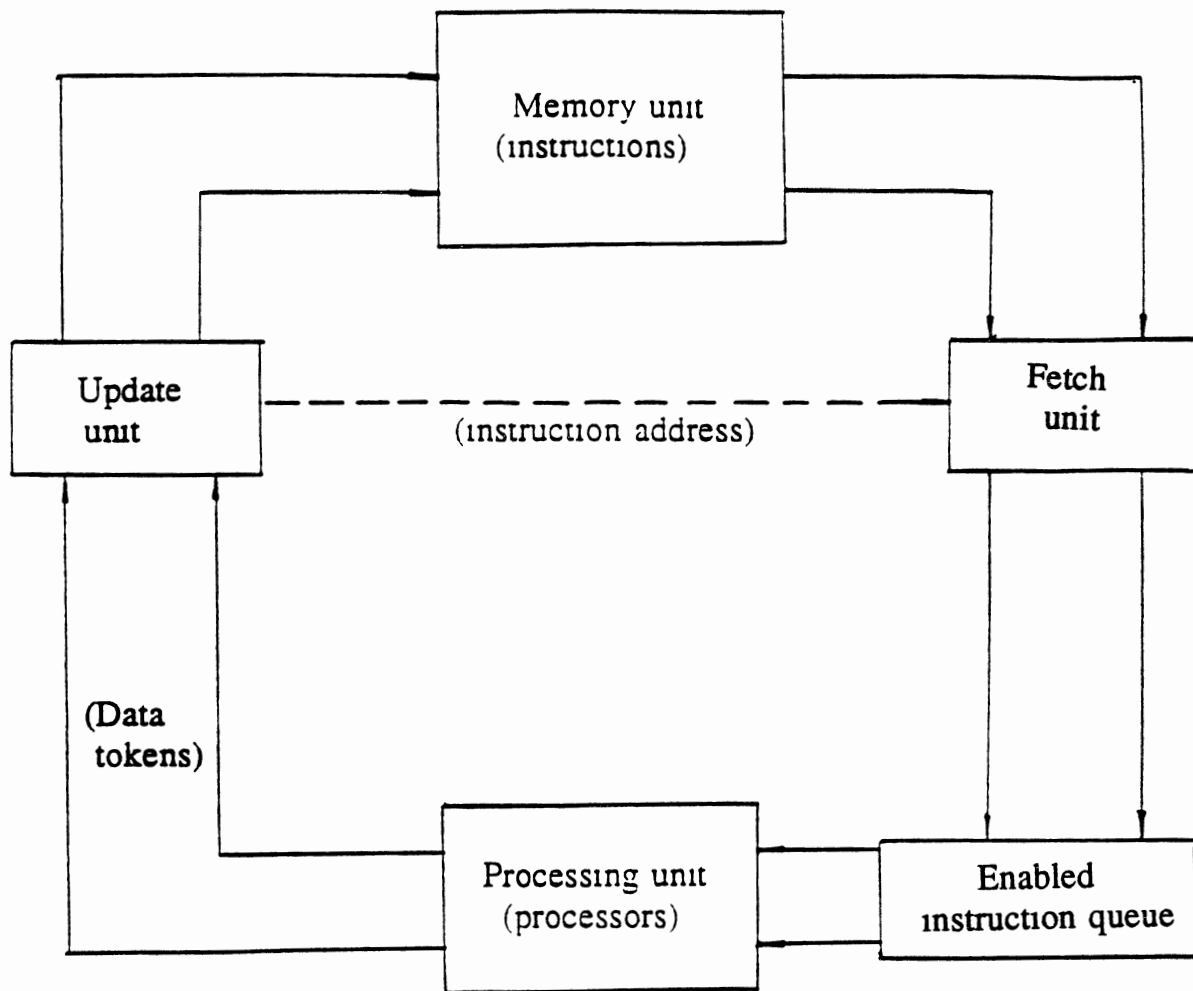


Figure 5. A Static Dataflow Computer Organization

on the arcs. Figure 6 shows a dynamic dataflow computer organization.

In a dynamic dataflow architecture, the tagged tokens are used, so that more than one token can exist on an arc. The tagging is achieved by attaching a label which uniquely identifies the context of a particular token with each token. This dynamically tagged dataflow model suggests that maximum parallelism can be exploited from a program graph.

Sorting Algorithms

The related activities of sorting, searching and merging are central to many computer applications. Sorting alone has been said to account for more than 30% of all computer time spent [22].

Sorting is not only one of the most important problems in computer science but it occurs in every other field of science also.

What Is a Sorting Algorithm?

Sorting algorithms arrange items in a set according to a predefined ordering relation. String information and numerical information are the two most common types of data.

Internal and External Sorting Algorithms

There are numerous algorithms available for sorting: internal sorting algorithms, which arrange data in main memory; and external sorting algorithms, which arrange data in external storage.

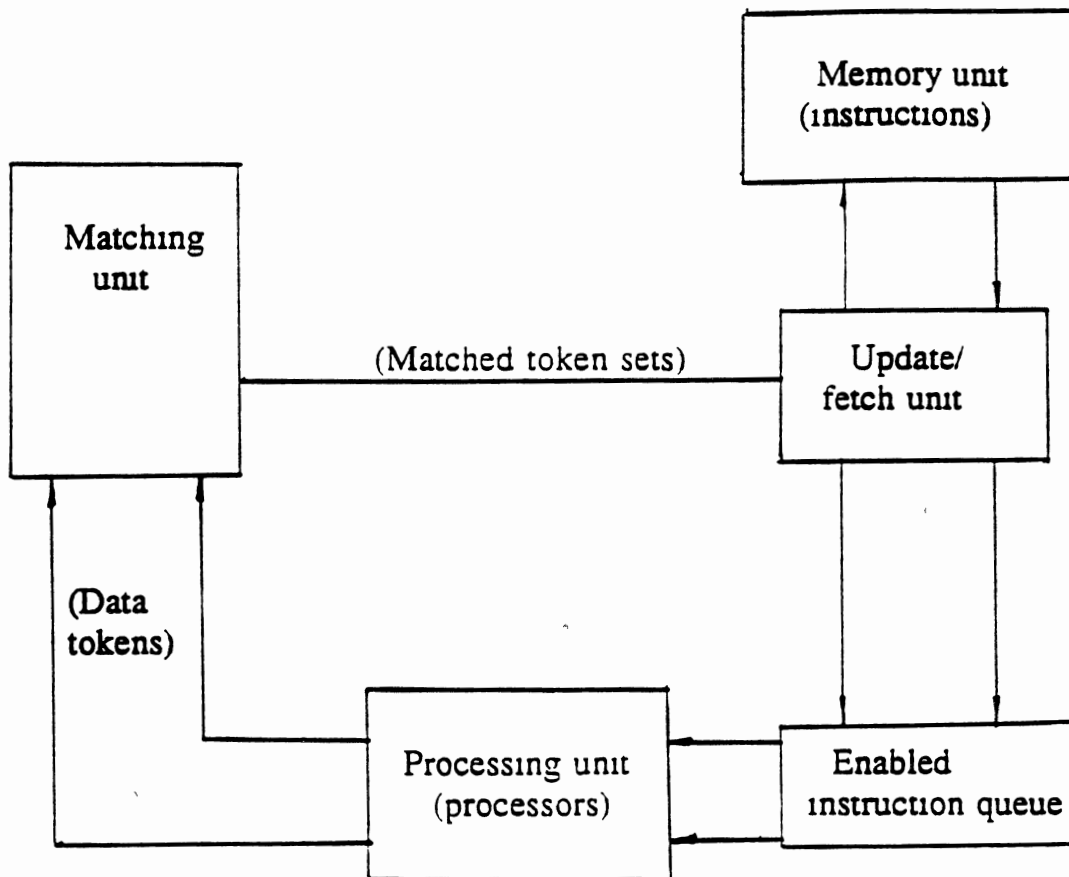


Figure 6 A dynamic dataflow computer organization

There is a variety of the simple and more advanced internal sorting algorithms, such as Shellsort, Quicksort, etc. (see [6] [9] [22]). When the keys to be sorted are general and have no known structure, a lower-bound result [3] states that sequential algorithms will require at least $\Omega (n \log n)$ time to sort a sequence of n keys. Many optimal algorithms like Quicksort and Heapsort, whose run times match this lower bound, can be found in the literature [3].

In principle, any internal sorting algorithm can be used to sort files which are stored in external memory. In practice, internal sorting algorithms are not used to sort external files because it can be too inefficient to process the file other than sequentially, especially if stored data values must be exchanged [34]. Files are ordinarily too large to fit within main memory, so we exclude the possibility of reading an entire file into an array, sorting it internally, then writing out the sorted array. The objective is to sort an external file by reading it sequentially, only one portion at a time, in order to create longer and longer runs (sorting consecutive subsequences). A run is natural if it is maximal: i.e., it is not a subrun of a longer run. A run is artificial if it has a prescribed length. In sorting a file, it is desirable to require as few passes as possible. Most external sorting algorithms are based on the principle of merging [40].

Parallel Sorting

Sorts of extremely large size are becoming more and more common. For instance, banks each night typically sort the checks of the current day into increasing order by account number. Then the

accounting files can be updated in a single linear pass through the sorted file [2]. Many computer scientists believe that the key to obtaining much higher speeds is the abandonment of the conventional von Neumann architecture and the adoption of new designs, in which calculations are performed in parallel rather than in a fixed sequence [37].

With the developing of parallel computer architectures the field of parallel sorting has grown enormously in the past decade [16].

There are many incomparable models of parallel computation being used among computer scientists. Richards collected together the different investigations until 1986 into one bibliography [40]. A very nice survey of parallel sorting is in [16]. The latest publications dealing with parallel sorting algorithms are [4] [5] [14] [15] [18] [23] [24] [27] [38] [39] [43] [55].

The theoretical basis of much of the work on comparison-based sorting can be traced either to the study of sorting networks or to the study of parallel decision trees. The premier result in parallel sorting is the existence of an $O(\log N)$ time sorting network as shown by Ajtai, et al [40]. In the parallel decision tree model there is no penalty assigned for scheduling and allocating work to the processing elements and there is no cost for routing data. The width of a computation is the maximum number of simultaneous comparison and the depth is the number of parallel comparison steps in the worst case.

External sorting is used to solve sorting problems in which the amount of data dwarfs the number of processors. There are "merge-

and-split" operations based on standard merging approaches as well as the substitution of normal comparison steps.

CHAPTER III

THE RELATIONSHIPS AMONG DATAFLOW ARCHITECTURES AND SORTING ALGORITHMS

There are two fundamental questions in the research on dataflow architectures: “Can the advantages of dataflow architectures be used for efficient sorting?” and “How do dataflow architectures improve sorting?” In this chapter, the first question is discussed.

The research on using dataflow architectures to improve sorting is based on:

- (1) basic concepts;
- (2) organization of computation;
- (3) program organization;
- (4) machine organization.

In order to give a comparative survey, control flow, is discussed with dataflow and reduction. The discussions in this chapter are based on the concepts presented by Treleaven et al [48].

Basic Concepts

Chapter II describes the basic concepts of dataflow architectures. High parallelism is a natural consequence of the dataflow concept. The basic operation of sorting is comparison

between two values. Parallel sorting is possible and can improve the efficiency. The sorting algorithms and dataflow architectures mesh well together. The concepts of dataflow architectures support the improvement of sorting algorithms.

Computation Organization

Computation organization within a dataflow architecture can be classified by considering computation as a continuous repetition of three phases: selecting, examining and executing.

In the selecting phase, a set of instructions is chosen by a computation rule for possible execution. Only instructions chosen in the selecting phase may be executed, but selection does not guarantee execution. Imperative, innermost and outermost rules are three types of computational rules. The expression to get the minimum number from the set (a, b, c, d, e, f, g, h ...) using parallel tree-sort algorithm is shown in Figure 7.

In the examining phase, each of the instructions previously chosen in the selecting phase is examined by a firing rule to see if it is executable. If an instruction is executable, it is passed on to the next phase for execution; otherwise, the examining phase may take some action, such as delaying the instruction.

In the executing phase which is broadly similar in all computation organizations, instructions are actually executed. As a result of execution the state of the computer is changed. Results are available to other parts of the program. Execution may produce globally perceived changes (such as changing the state of a globally

The expression to get the minimum number from
the set (a, b, c, d, e, f, g, h ...) using
parallel tree-sort algorithm

Imperative: Instruction selected depending on the value of
program counter (PC).

(...(min (min (min a b) (min c d)) , (min (min e f) (min g h)))...)

↑
PC

Innermost: Instructions selected are the most deeply nested

(...(min (min (min a b) (min c d)) , (min (min e f) (min g h)))...)

↑ ↑ ↑ ↑

Outermost: Instructions selected are most outer.

(...(min (min (min a b) (min c d)) , (min (min e f) (min g h))) .)

↑

Figure 7. The Computation Rules Applied to an Expression

shared memory) or produce localized changes (For example, an expression is replaced by its value).

The fetch part of the fetch-execute control cycle is the selecting phase of control flow computation. In the selecting phase, the instructions to be used are chosen by the program counter. Once chosen by selecting, instructions are not checked by an examining phase, but automatically passed on to execution. The executing phase of control flow instructions is allowed to change any part of the state of computation. Control flow uses a shared memory to communicate results. The state of computation is represented by the contents of this shared memory and the program counter register(s). A program counter is updated at the end of each cycle either implicitly or explicitly in the case of GOTOs.

The control flow refers to the computation organizations in which instructions are executed as soon as they are selected. For all computation organizations in control flow, the examining phase is redundant and instruction sequencing is independent of program structure.

In dataflow, instructions are executed as soon as all their arguments are available. So the selecting phase of dataflow computation may be viewed as logically allocating a computing element to every instruction. The examining phase implements the dataflow fire rule, which requires all data to be available before execution can take place. If the values are not available, the instruction will not be executed and remain dormant during the execution phase. In dataflow, the execution phase changes a local

state consisting of the executing instructions. The instruction consumes data values and places a result value in each destination.

The dataflow refers to the computation organizations wherein instructions passively wait for some combination of their data values to be available. This implies a selecting phase, which logically allocates computing elements to all instructions, and an examining phase, which suspends nonexecutable instructions. The key governing execution is the availability of data.

Reduction computers have different rules in their selecting phase. The choice of the computation rule is a design choice for a particular reduction computer. Innermost and outermost rules are the commonest rules used in reduction. The computation rule in a reduction computer determines the allocation of computing elements at the beginning of each computation cycle. In the examining phase the arguments are examined to see whether execution is possible. If possible, the instruction is executed. Otherwise, the arguments are suspended until all input values are available for execution. The instruction set of a reduction computer may contain many different firing rules; each instruction has the rule most suited to it. The execution phase in a reduction machine involves rewriting an instruction. The instruction is replaced by its result.

Reduction refers to the computation organization where instructions are selected only when the value they produce is needed by another already selected instruction. All outermost reduction architectures are demand-driven computation organization. In reduction computers with an innermost computation rule, instructions never are chosen by selecting until their arguments are

available. This restriction means that all arguments reaching the examining stage are preevaluated, exactly as occurs in dataflow. Thus innermost computation organizations are data-driven.

Control-flow computers have a control-driven computation organization. Instructions are selected by program counter and once selected they are immediately executed. But in the selecting phase, the instructions to be used are sequentially chosen by a program counter. Data-flow computers have a data-driven computation organization. In the selecting phase, the computing elements are locating to the instructions if they are available. The dataflow fire rule of examining phase requires all data to be available before execution. In the three phases of dataflow, computation organization has high parallelism. Some reduction computers are demand driven and some are data driven. As in simple dataflow architectures, all instructions execute only when their arguments become available.

From the point of computation organization, control flow is suited only to the sequential data input and sequential sorting algorithms. Data flow and reduction can get greater parallelism for sorting if all the data is available simultaneously.

Program Organization

The program organization shows the way of machine code programs which are represented and executed in a computer architecture. The data mechanism and the control mechanism are two basic computation mechanisms of program organization for these three groups of computers. The Figure 8 is a summary of the

Computation Mechanisms		Data Mechanisms	
		By Value	By Reference
Control Mechanisms	sequential		Control Flow
	parallel	Dataflow	
	recursive	String Reduction	Graph Reduction

Figure 8. Computation Mechanisms of Control Flow, Dataflow, and Reduction Program Organizations

relationship of these computational mechanisms to the three groups of computers.

The control mechanism of control flow is based on "sequential". The basic data mechanism of control flow is "by reference". The effects of changing the contents of a memory cell are immediately available to other users.

Iteration may cause a potential problem for parallel control flow. Program fragments with loops may lead to logically cyclic graphs in which each successive iteration of a loop could execute concurrently, giving the possibility of multiple-data items being stored in the same memory. So special precautions need to be taken in a representation of program to ensure that the natural asynchronous execution does not lead to unwanted indeterminacy, such as by using two schemes of feedback and recursion.

Dataflow is based on a "parallel" control mechanism and a "by value" data mechanism. Flows of data and control are identical in dataflow. A copy of a partial result is passed directly by a data token from the producer to the consumer instruction. There are independent copies of shared data.

The basic format [7] [50] of a reference is in Figure 9. The process field P is used for separating instances of an instruction N that may be executing in parallel, either within a single program or distinct programs. The instruction field N is used for identifying the consuming instruction to which the data token is being passed. The argument field A is used for identifying in which argument position of the instruction N the token is to be stored. In the machine code of a dataflow computer, the values of the N and the A field are usually

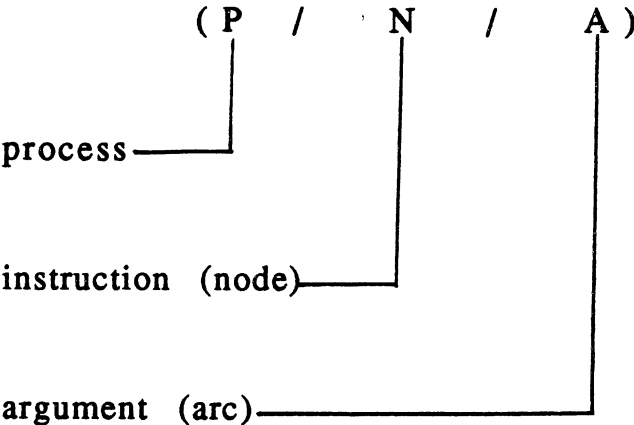


Figure 9. The Basic Format of a Reference in Dataflow

statically embedded in the code at compile time. But the value of P is dynamically generated at run time by the system.

Dataflow must take similar precautions as in parallel control flow. Same as control flow, by using feedback and recursion can avoid unwanted indeterminacy. A third scheme for supporting iteration in dataflow computers is based on an additional iteration number field [7] [50] in each reference, for example, P/N/A/I. This iteration number field distinguishes individual data tokens, logically flowing on a particular arc by giving each token a unique I value, for example, 1, 2, 3.

Reduction is based on recursive control mechanism and either a by-value or a by-reference data mechanism. Reduction is inherently recursive. Because a by-value data mechanism is used, in string reduction separate copies of actual arguments are generated for each formal parameter occurrence. String manipulation is best suited to innermost computation rules where functions are applied only to previously evaluated arguments. Graph reduction is suited to outermost computation rules [48].

Control-flow program organizations are less efficient than dataflow because they have a separation of flows of control from flows of data. For example, in control flow, passing the partial result of a subexpression to the enclosing subexpression requires three operations: storing the result, sending the control flow, and loading the result. However, in dataflow there is only one operation: sending the data token. The data token scheme combines both the by-value data mechanism and the data-driven control mechanism.

The simplicity and the parallel nature of program organization are the major advantages of dataflow.

Transferring data is another basic operation in the sorting algorithms. So making communication quick can improve the speed of sorting. Using data token scheme to transfer data would be much faster than control flow. In the von Neumann machine the processor issues a memory request and waits for the result to be produced. The memory cycle time is invariably greater than the processor cycle time. This problem is much more severe in a multiprocessor because the time to process a memory request is generally much greater than in a single processor and is unpredictable. The dataflow architecture is an extreme solution to the memory latency problem: the processor never waits for responses from memory; it continues processing other instructions. Instructions are scheduled based on the availability of data. Otherwise, because communication is quick, the processes can be made very small, about the size of a single instruction in a conventional computer. This makes segmentation trivial and improves scalability.

Machine Organization

Machine Organization is the configuration of a machine's resources; they are allocated to support a program organization. There are three basic classes of machine organization: centralized, packet communication, and expression manipulation.

A centralized machine is shown in Figure 10. There are a single processor, communications, and a memory resource.

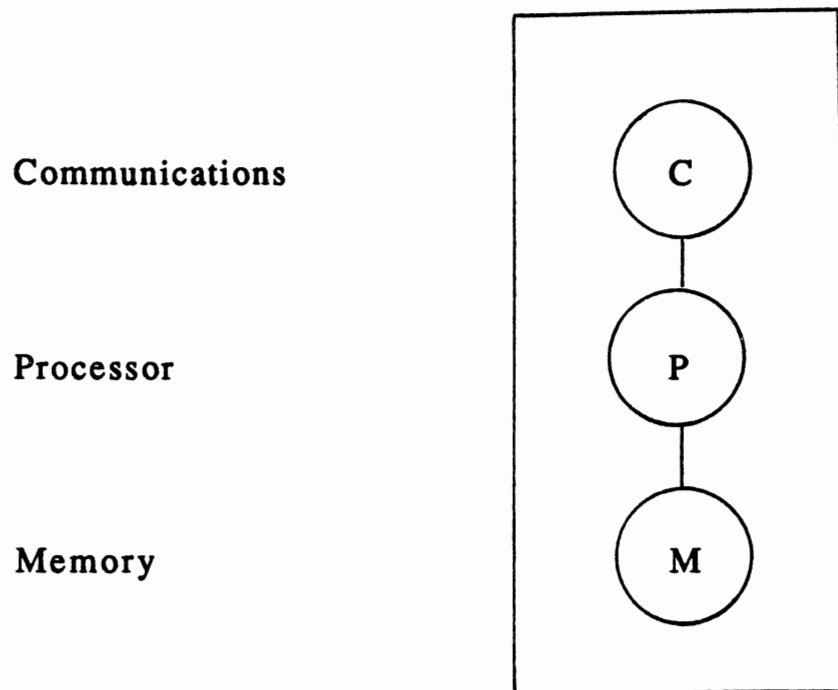


Figure 10. Centralized Machine Organization

A packet communication machine organization is shown in Figure 11. There are a circular instruction execution pipeline of resources in which processors, communications and memories are interspersed with "pools of work." Parallelism is obtained either by having a number of identical resources between pools or by replicating the circular pipelines and connecting them by the communications.

Expression manipulation machine organization consists of identical resources usually organized into a regular structure such as a vector or tree, as shown in Figure 12. Each resource contains a processor, communication and memory capability.

A centralized organization is most suited to sequential control flow. The simplicity, both for resource allocation and implementation, is its advantage. The lack of parallelism is its disadvantage. A packet communication organization for control flow can achieve parallelism, but it lacks the concept of an implicit next instruction. An expression manipulation machine organization for control flow is suited to a parallel FORK-JOIN style of control flow, but incurs additional FORK and JOIN style control operators.

It is hard to imagine a centralized machine organization for dataflow because there are a large number of potentially executable instructions. Packet communication provides two alternative organizations to support dataflow. Figure 13 shows the first scheme which is based on storing data tokens into an instruction and executing the instruction while it is complete. Figure 14 shows the second scheme which is based on matching data tokens. Dataflow

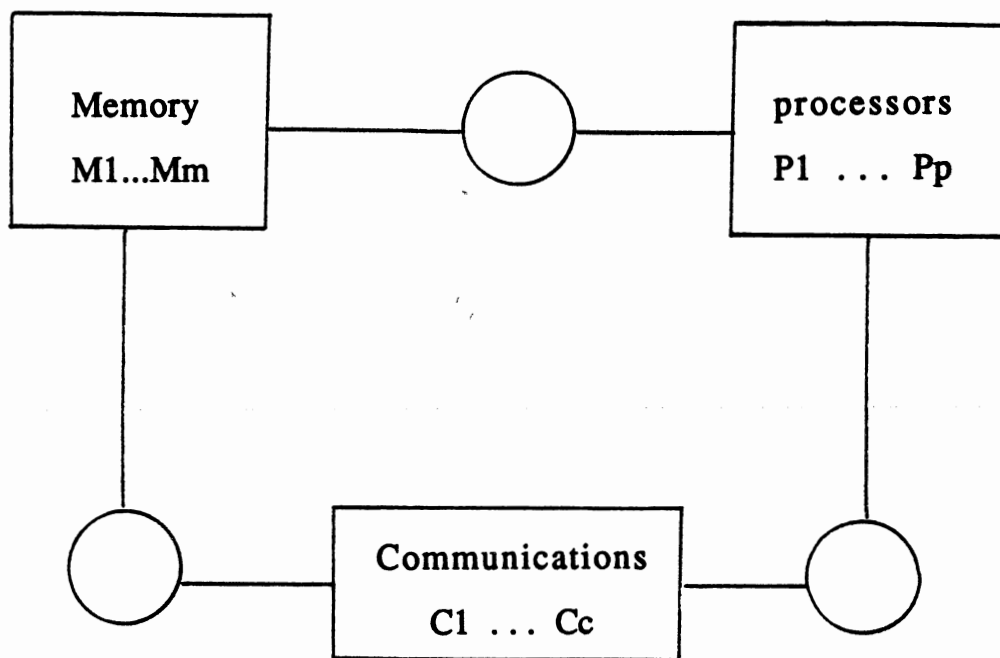
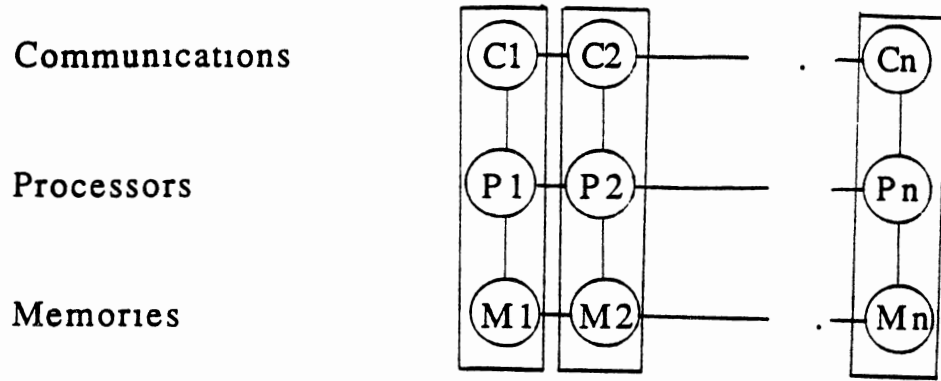
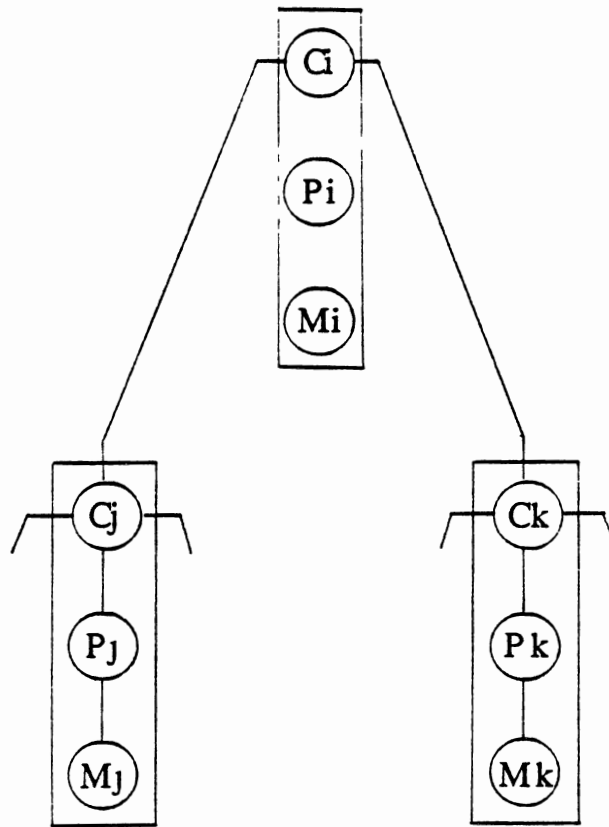


Figure 11. Packet Communication Machine Organization



(a) Vector



(b) Tree

Figure 12. Expression Manipulation Machine Organization

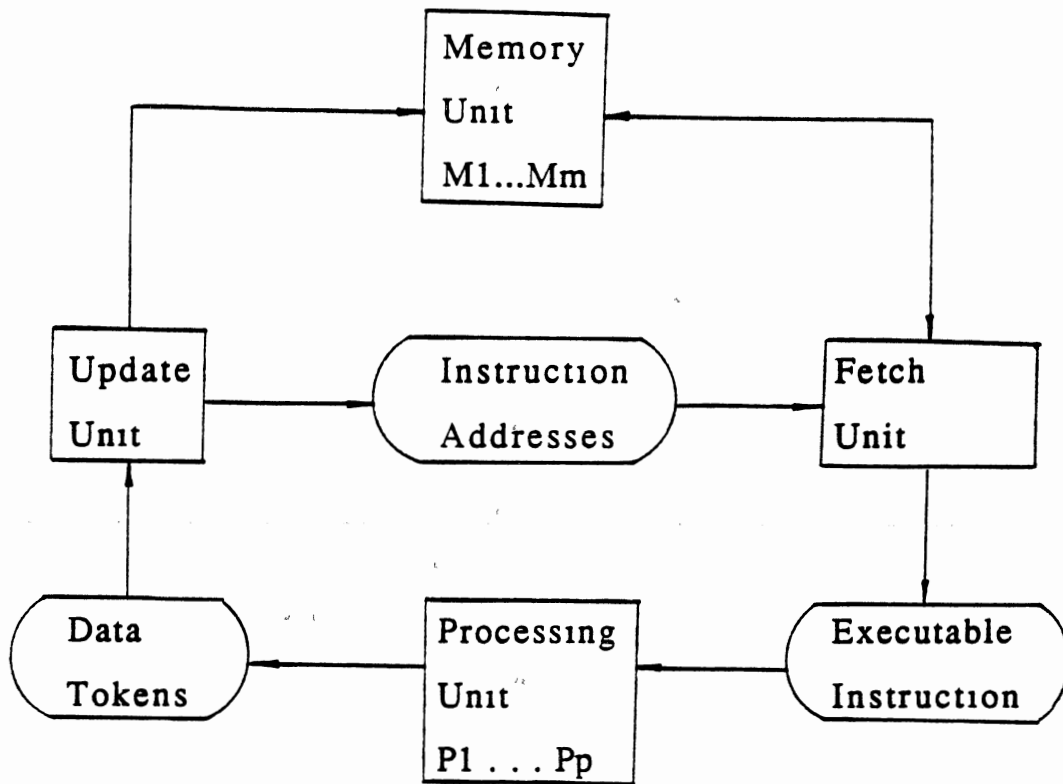


Figure 13. Dataflow Packet Communication with Token Storage

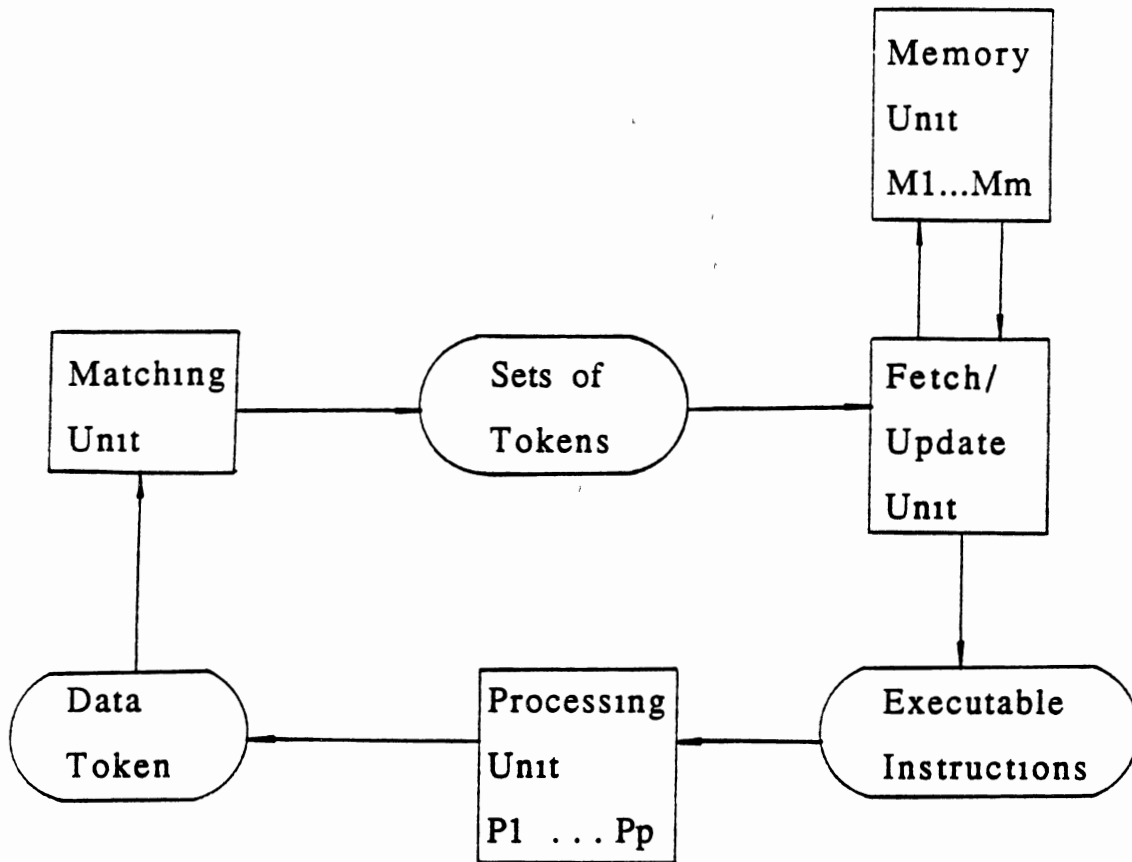


Figure 14. Dataflow Packet Communication with Token Matching

architectures in which the processors are connected by dataflow graphs are built by expression manipulation machine organization.

Reduction can be supported efficiently by any of the three machine organizations because of the various computational rules for it. A centralized organization is best suited to a sequential form of reduction. A packet communication and expression manipulation organization are suited to a parallel computational rule.

Comparison between two values and data transfer are basic operations in sorting. Sorting algorithms can have a high degree of parallelism. The packet communication and expression manipulation organizations for dataflow are two of the best machine organizations for sorting algorithms. A packet communication organization is most suited for a general-purpose sorting machine. An expression manipulation organization is most suited for a special-purpose sorting machine.

Dataflow architectures and sorting algorithms are suited to each other in terms of the basic concept, computation organization, program organization and machine organization. Dataflow architectures strongly support improvement of sorting algorithms.

CHAPTER IV

DATAFLOW SORTING MACHINES

This chapter is concerned with the architecture of dataflow machines for improving parallel sorting algorithms. Special-purpose dataflow sorting machines, general-purpose dataflow sorting machines, and dataflow database machines are discussed.

Special-Purpose Sorting Machines

Finding a good mapping of a parallel sorting algorithm onto a hardware architecture is a feasible way to build a special-purpose sorting machine. To fix the processing elements as a tree or a vector is not a good way to design a general-purpose dataflow machine, but it is a good idea for a special-purpose sorting machine. A binary tree data structure with $(2n-1)$ nodes is used to sort n numbers in a serial tree selection sorting algorithm. Sorting by selection seems to be the best method of sorting which allows avoidance cumbersome exchanging, insertions, and other operations performed on a file records [32]. A special-purpose sorting machine in which the processors are fixed as a binary tree is proposed below. It is adapted from a parallel tree-sort algorithm reviewed in the paper by Bitton, et al [16]. The algorithm is introduced first.

A Parallel Tree-Sort Algorithm

The binary tree, which has $2n-1$ nodes, has n leaves, and initially one number is stored in each leaf. Sorting is performed by selecting the minimum of n numbers first, then the minimum of the remaining $(n-1)$ numbers, etc.

The binary tree structure is used to find the minimum number by iteratively comparing the numbers in two sibling nodes, and moving the smaller number to the parent node. By simultaneously performing comparisons throughout the binary tree, a parallel tree sort is obtained [11]. A simple example is shown in Figure 15.

Consider a set of n processors interconnected to form a binary tree with one processor both at every pair of leaf nodes and at pairs two interior nodes of the tree (see Figure 16). By starting with one number at each leaf processor, the minimum can be transferred to the root in $\log(n)$ parallel comparison and transfer steps. At each step, a parent processor receives one or two element from each of its two children processors, performs a comparison, retains the larger element, and transfers the smaller one to its parent, Then empty side receives another element from the child processor, The sorting is completed in time $O(n)$. It will be a binary parallel merging tree if a set of ordered data is the input to each leaf (see Figure 17 for a simple example).

Parallel Tree Sorting-Merging Processor

The dataflow graph of a tree sorting-merging processor is shown in Figure 18. There are two kinds of operators in the dataflow

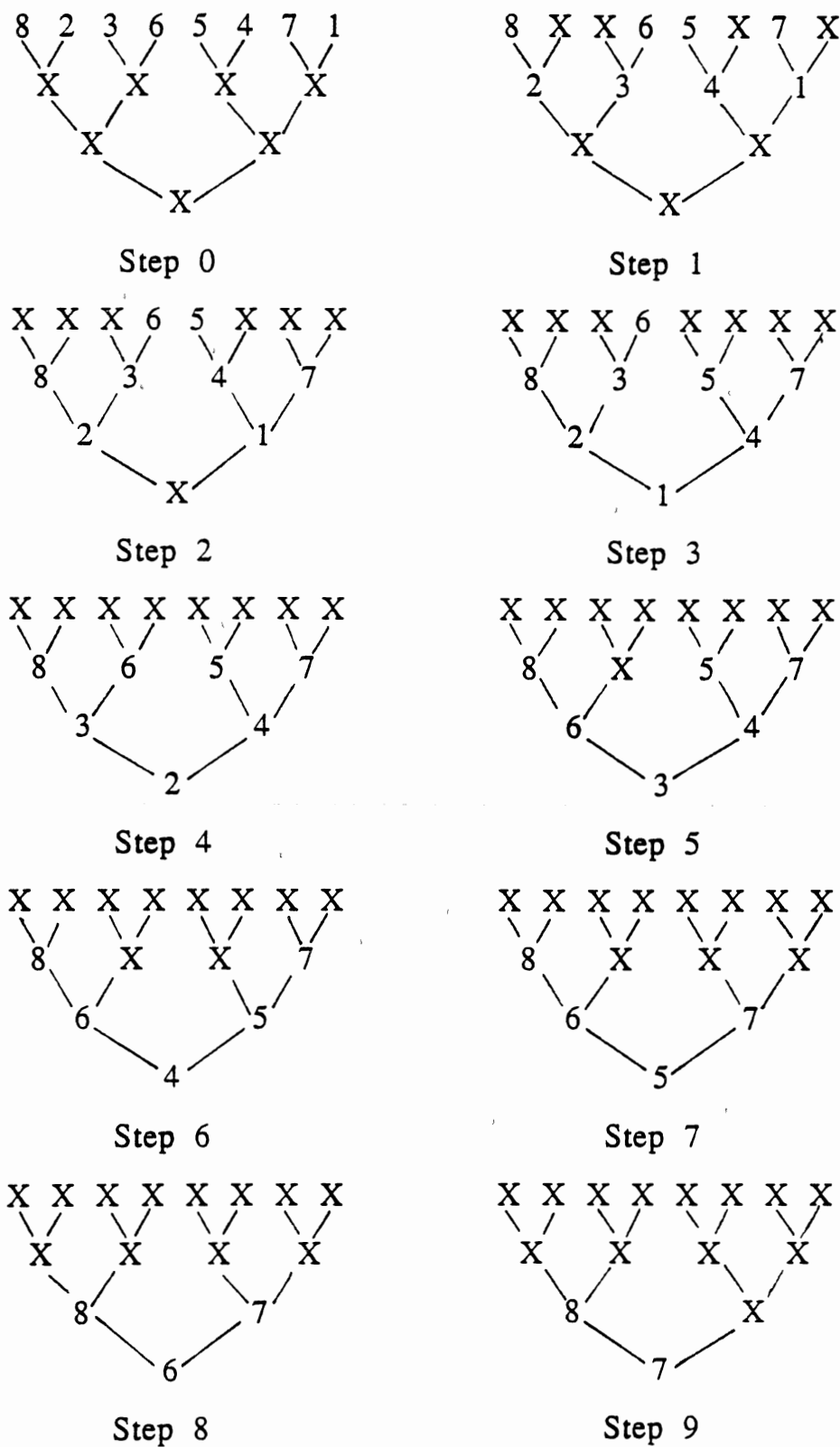


Figure 15. An Example Using Parallel Tree Selection Sort

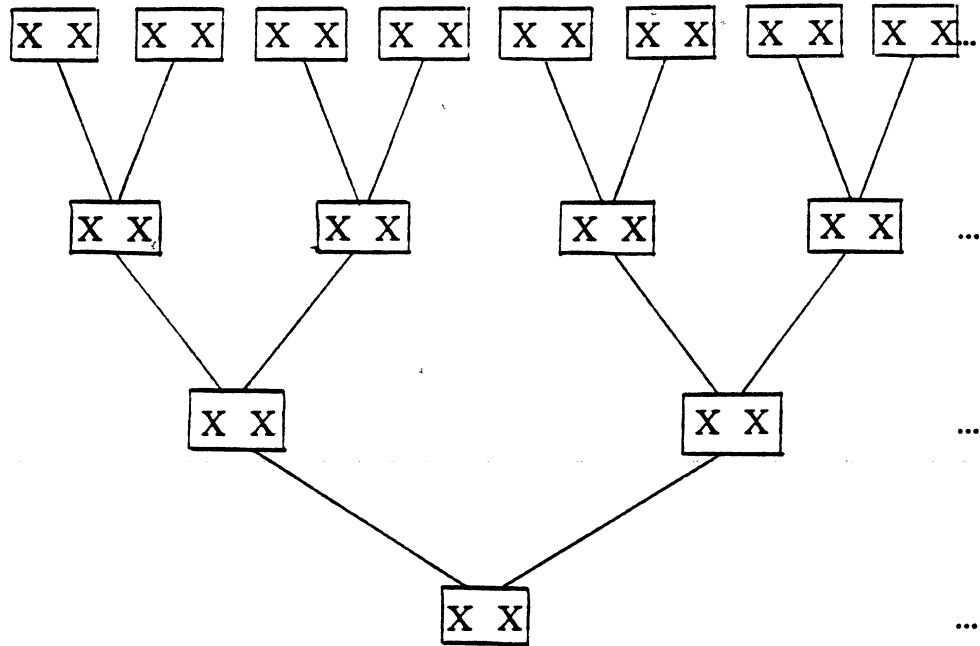


Figure 16. The Sorting-Merging Processors in a Parallel Tree

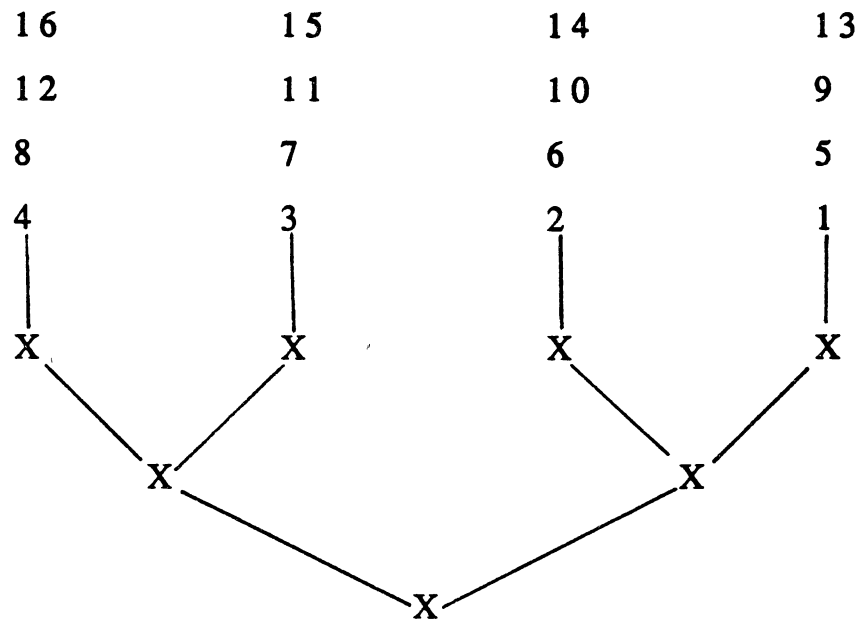


Figure 17. An Example of Parallel Binary Merging Sort

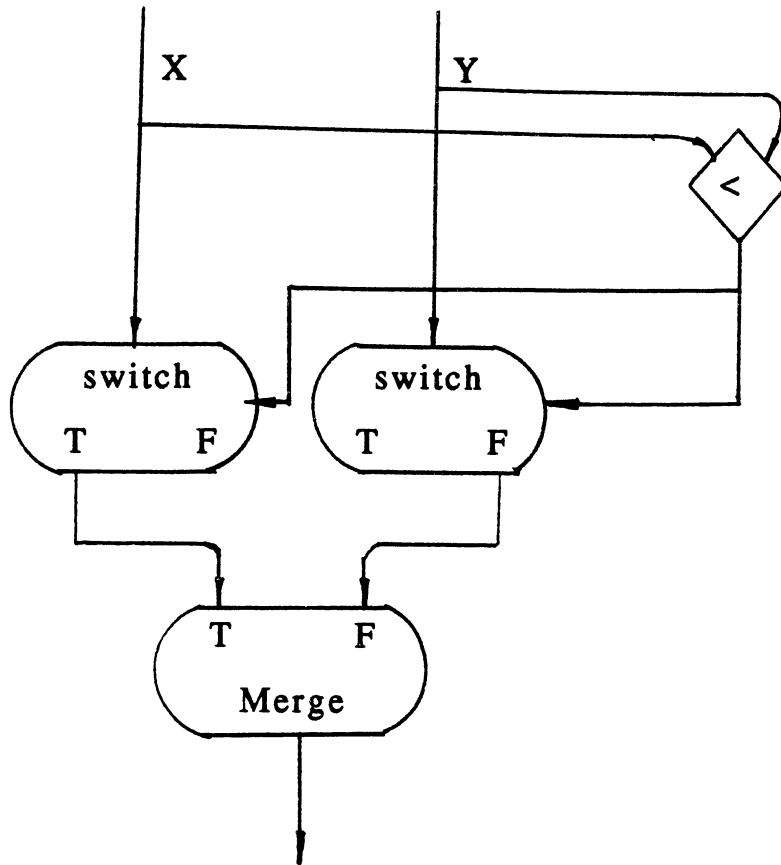


Figure 18. The Dataflow Graph of a Sorting-Merging Processor

graph: function operators and control operators. The comparison-function operators, such as $>$, $<$ and $=$, are often using for sorting algorithms. Two control operators, switch and merge, are used often for sorting algorithms also. The advantages of a tree-sorting processor are simple: no program is necessary and it is easy to make such a processor on a VLSI chip. The whole tree can be made by either connecting several chips or only by one chip.

Figure 19 is an outline of a proposed special-purpose dataflow parallel tree sorting machine. Figure 20 shows the DPTSM1 (Dataflow Parallel Tree Sorting Machine 1) machine. There are 64 binary sorting-merging trees (each has 1024 leaves) and one merging tree which has 64 leaves. So the maximum number for sorting once is 85736. Both the input pool and output pool can be connected to the secondary storage.

General-Purpose Sorting Machines

The special-purpose machine can get high efficiency when the input data number is near a certain number, such as 85736 for the DPTSM1 machine. This kind of machine improves sorting for special work. Its disadvantage is its lack of flexibility. When the number of data input is not close to the given number, then many sort-merge processors will be wasted.

A general-purpose sorting machine has more flexibility than special-purpose sorting machine. This kind of sorting machine is more convenient for a research center, but it is more complex than the special-purpose sorting machine. Figure 21 is an outline of a

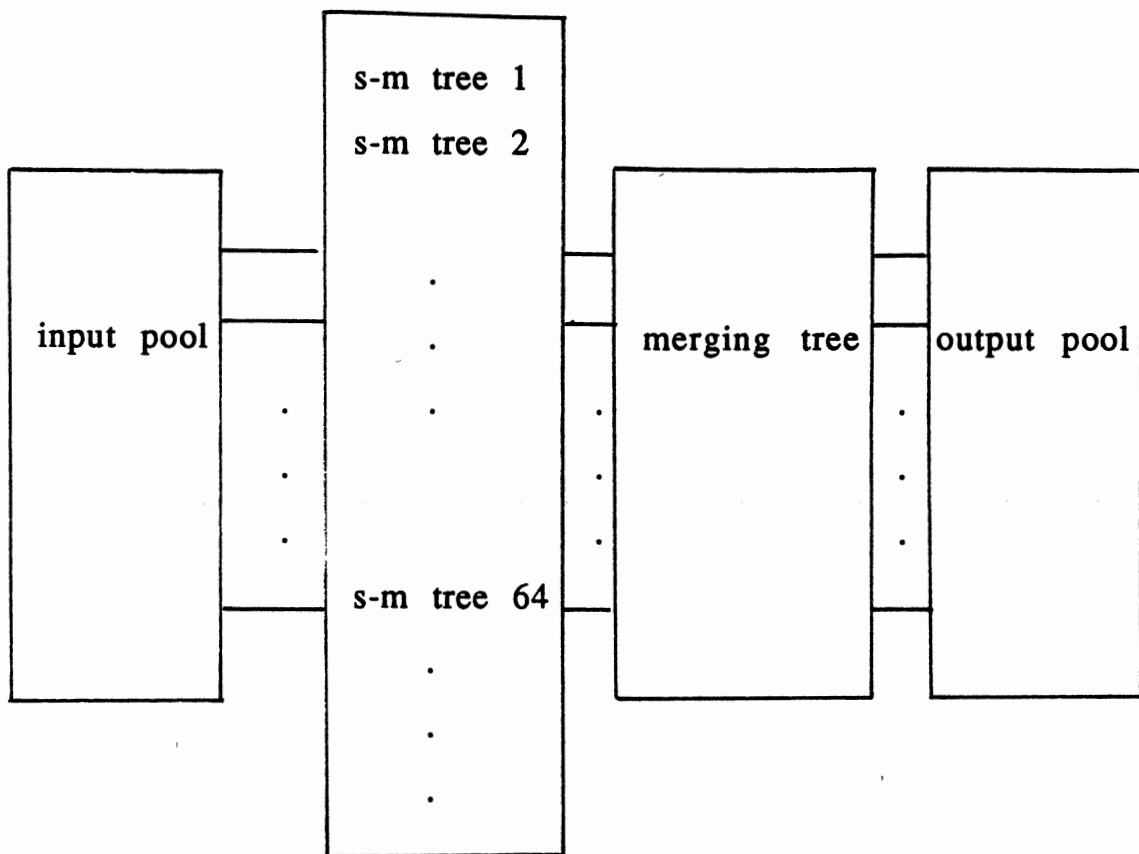


Figure 19. The Outline of a Special-Purpose Dataflow Parallel Tree Sorting Machine

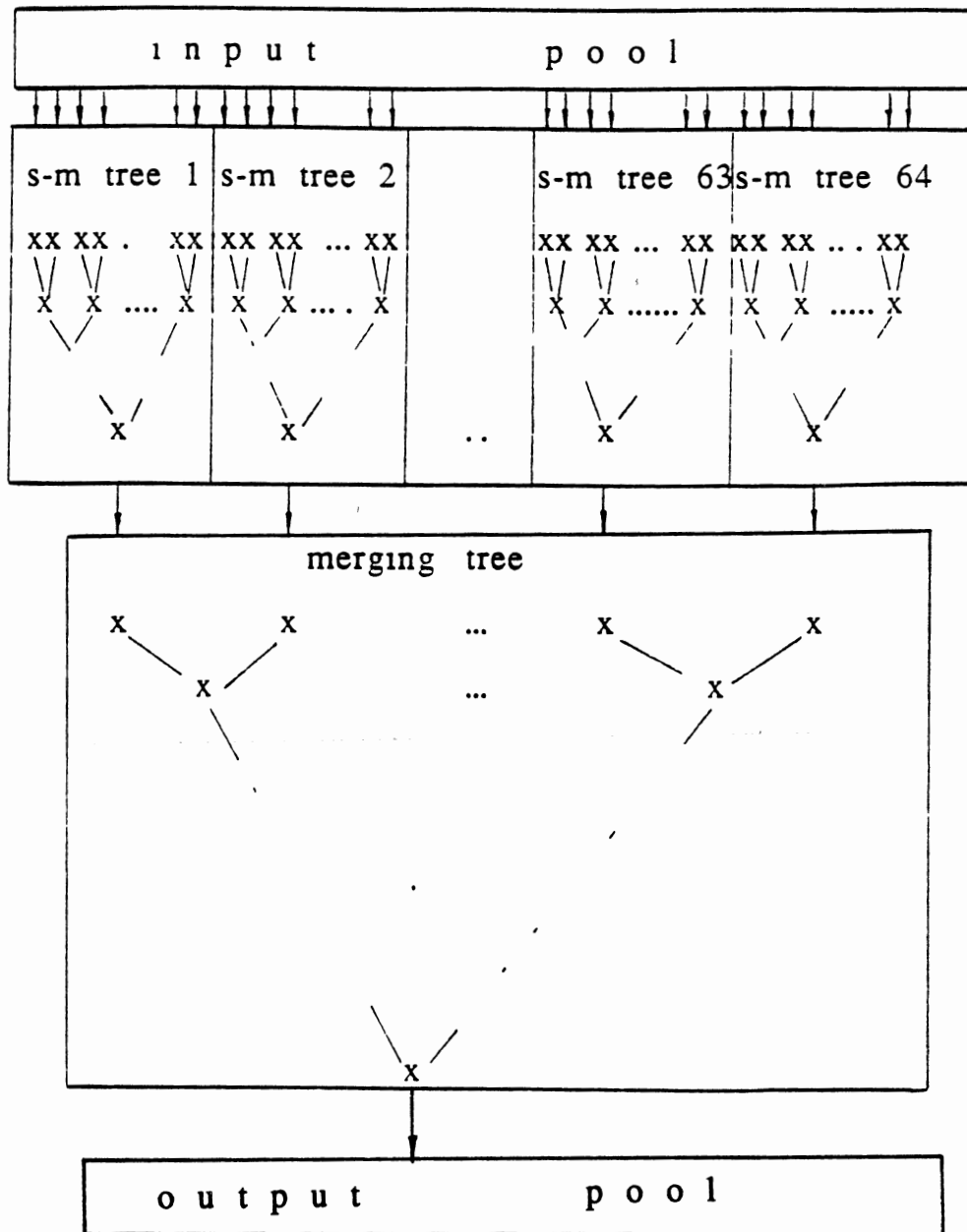


Figure 20. DPTSM1 Machine

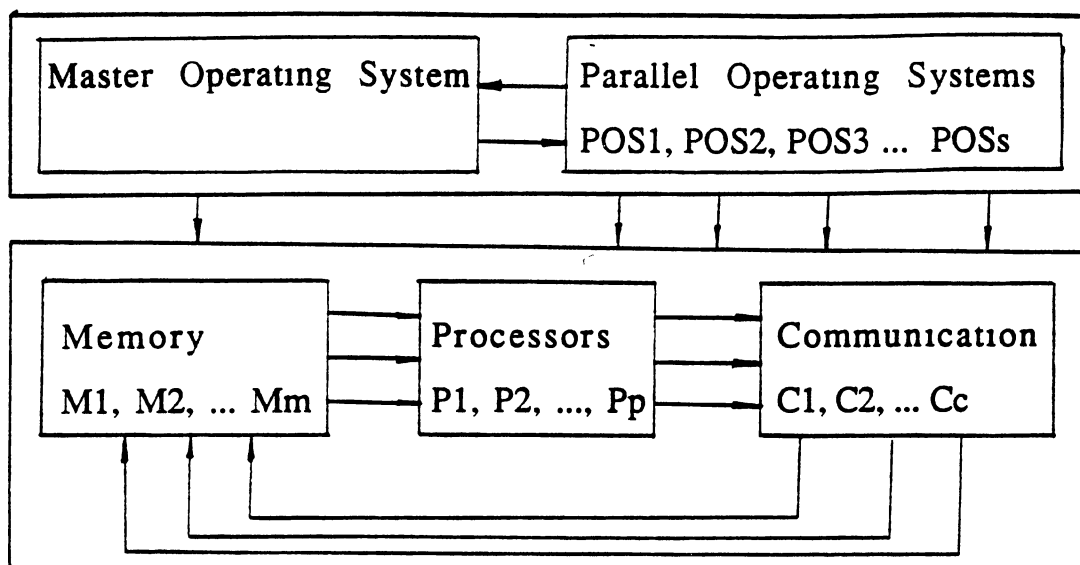


Figure 21. The Outline of a General-Purpose Dataflow Sorting Machine

proposed general-purpose dataflow sorting machine. Figure 22 shows the GPDSM1 (General-purpose Dataflow Sorting Machine 1) machine. Many terminals can use GPDSM1 at the same time. In order to use all processors efficiently, there are a master operating system and several slave parallel operating systems to manage the work in the machine. The master operating system manages global resources and communications among the slave parallel operating systems. In case of need, the master operating system can help a parallel operating system borrow some resources from others. The master operating system can synchronize the the slave parallel operating systems to manage resources. The operating systems can be implemented by either hardware or software. This is a packet communication machine organization. There is a processor for each kind of operation, such as addition, comparison, and logical operation. Comparison processors are the most used. The processors are connected according to the algorithms of the program. The connecting of processors is very flexible. The numbers t , m , p of Term t , Mm , and Pp in Figure 22 are determined by the requirements before final design.

Dataflow Database Machines

Both general-purpose and special-purpose machines can improve the time required for sorting a large list of data. For inserting a few data in the ordered list, the whole list should be sorted again with the new data.

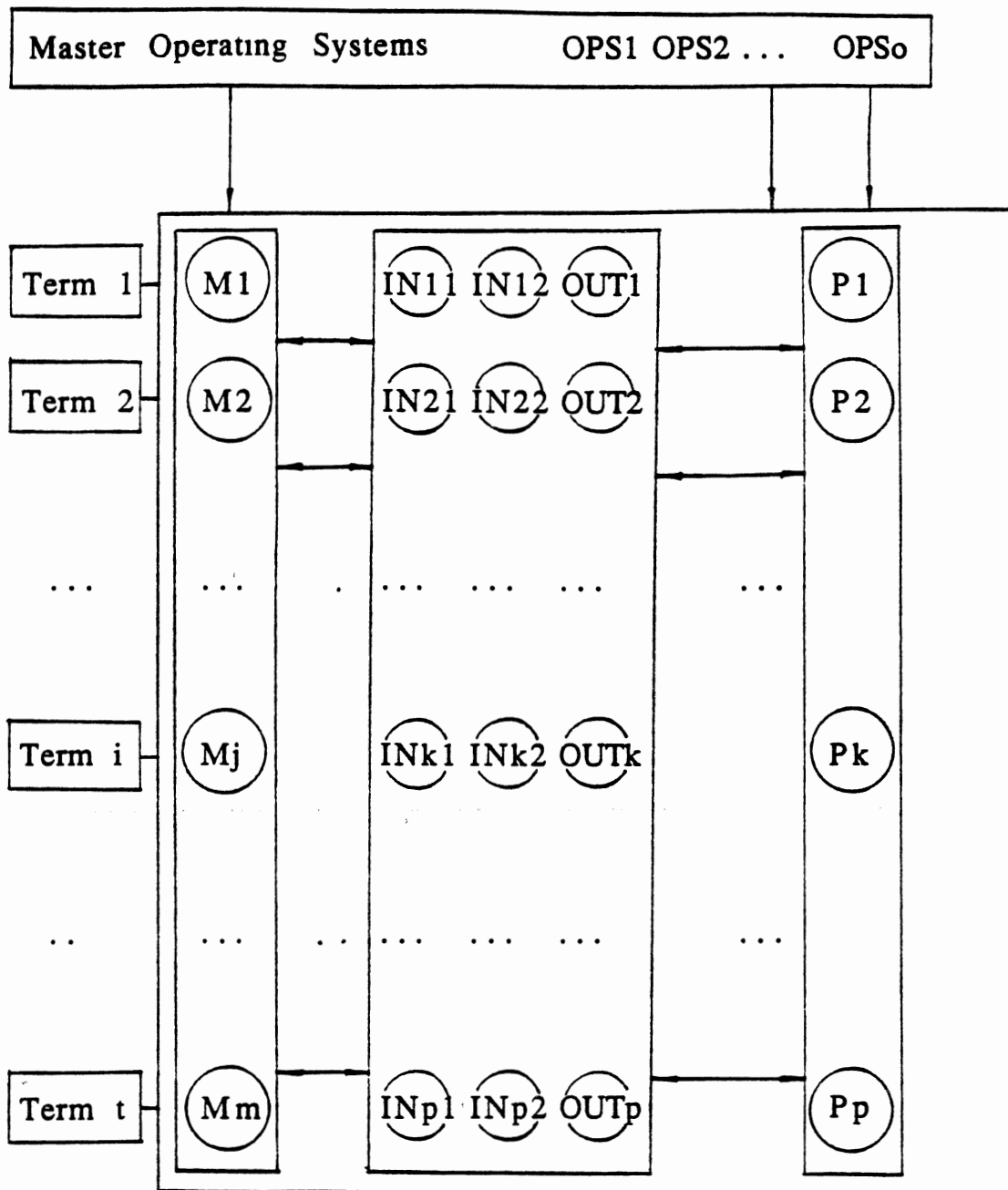


Figure 22. The Structure of GPDSM1

A dataflow database machines has significant potential parallelism in the form of searching, sorting, and various kinds of updates [12]. Their architectures are based on the principles of data-driven computation. This kind of machine consists of a large number of disk units, each equipped with a separate processing element. According to this model, the database is represented as a network in which each node is conceptually an independent, asynchronous processing element, capable of communicating with other nodes by exchanging messages along the network arcs. To answer a query, one or more such messages, called tokens, are created and injected into the network. These tokens propagate asynchronously through the network in search of result satisfying the given query. The asynchronous nature of processing permits the model to be mapped onto a computer architecture consisting of large numbers of independent disk units and processing elements.

The database is a dataflow graph, where each node has associated node to be mapped onto possibly a different PE node, resulting in a high degree of parallelism during execution. An integral part of the model is a high-level data manipulation language that permits the user to specify queries and updates by prescribing the flow of tokens through the database sets.

A distributed and deadlock/restart-free dataflow database machine is described in [12]. It is based on the token-tagging scheme used in dataflow systems. The model has been implemented on a simulated computer architecture to obtain some preliminary performance estimates. The result of these simulations may be found in [13] and [25]. A prototype of this machine has been

developed and is currently operational [12]. It incorporates 16 processing elements, each having a 32-bit INMOS transputer and a 200 Mbyte disk. Performance testing of the prototype is currently in progress. A commercial version of this machine is being developed by Hyperstore Systems Inc., of Irvine, California.

CHAPTER V
SUMMARY, CONCLUSIONS, AND SUGGESTED
FUTURE RESEARCH

Summary and Conclusions

In the last few decades, the problem of sorting a given file has become one of the most important problems in computer science. Many new algorithms have been presented in order to solve this problem. Sorting extremely large sets of data is becoming more common. But even the best sorting algorithm consumes the time of $O(n \log n)$ [31]. Since this time is unsatisfactory for many practical sorting tasks, then construction of dedicated sorting machines has occurred [10] [35] [46] [53] [54]. A common property of these machines is that a comparator is their basic component, and comparison of two numbers is the basic operation. At least, $\log(n!)$ comparisons are necessary for sorting a set of n keys. The purpose of new sorting machines is to make as many comparisons as possible in parallel.

Dataflow architectures show promise of making use of both VLSI and parallelism. In a dataflow computer an instruction is ready for execution as soon as all its operands are available. Either intermediate or final results are passed directly as data tokens among instructions. The total amount of time required for data

transfer time in dataflow computers is much less than what is required in control flow computers.

Comparison between two values and data transfer are basic operations in sorting algorithms. Sorting algorithms can have a high degree of parallelism. Improvement of sorting algorithms can be effected computer architectures which have high parallelism and which minimize the total time required for data transfers.

Parallel sorting architecture can be classified into two major categories: sorting that requires special hardware or processors, and sorting that can be implemented either on general purpose computer networks, or on multiprocessor systems [55].

When algorithms and architectures mesh well together, the architecture supports the algorithm [36]. High parallelism is a natural consequence of dataflow architectures. Therefor dataflow architectures strongly support parallel sorting algorithms.

There are three kinds of sorting computers which use dataflow architecture for performance improvement: special-purpose dataflow sorting machines, general-purpose dataflow machines, and dataflow-database machines. The processors of a special-purpose dataflow sorting machine can be structured according to a parallel sorting algorithm. They are fixed-processor dataflow structures. They are efficient for a sorting center which performs sorting for customers. The processors of a general-purpose dataflow sorting machine are structured dynamically by the programs. They are efficient either for a big research center or for a large university which has many kinds of large sorting tasks. A dataflow-database machine consists of a large number of disk units, each equipped with a separate

processing element. It has been designed to exploit the potential parallelism for a database in which there is frequently searching, and sorting, as well as various kinds of updates.

Suggested Future Research

A future research topic in the study of general-purpose sorting machines is how to connect the processors dynamically. This is a communication problem as well. At present, using hardware is one of the ways to solve this problem. A better and more convenient way communications technique is needed in the future.

For special purpose sorting machines, one of the future research problems is how to use the processors efficiently. Partial control of processors can be added to improve the efficiency. Otherwise, making machines more flexible is another important problem in the future study.

The dataflow-database machine is in the practical use period. Combining processors and storage together can be the best way of dataflow-database machine. Then the modifications of sorted file can be done by database itself.

BIBLIOGRAPHY

- [1] Agerwala, T., "Data Flow Systems," Computer (Feb 1982), 10-12.
- [2] Aggarwal, A. and , J. S., "The Input/Output Complexity of Sorting and Related Problems," Commun. of the ACM, 31 (Sep. 1988), 1116-1127.
- [3] Aho, A, Hopcroft, J. E. and Ullman, J. D., The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.
- [4] Alon, N. and Azar, Y, "The Average Complexity of Deterministic and Rendomized Parallel Comparison-sorting Algorithms," SIAM J. Comput., 17 (Dec 1988), 1187-1192.
- [5] Alon, N. and Azar, Y., "Finding an Approximate Maximum," SIAM J. Comput., 18 (Apr 1989), 216-228.
- [6] Amsterdam, J., "an Analysis of Sorts,"Byte, 10 (Sep. 1985), 104-108.
- [7] Arvind and Gostelow, k. p., "A Computer Capable of Exchanging Processors for Time," Proc. IFIP congress (1977), 849-854.
- [8] Arvind and Culler, David E., "Dataflow Architectures," Annual Reviews in Computer Science, 1 (1986), 225-253.
- [9] Baase, Sara, Computer Algorithms: Introduction to Design and Analysis, Addison-Wesley Publishing Company, 1978.
- [10] Batcher, K. E., "Sorting Networks and Their Applications," Proc. AFIPS 1968 SJCC, 32, 307-314.

- [11] Beatley, J. L. and Knug, H. T., "A Tree Machine for Searching Problems," Proc. of the 1979 International Conference on Parallel Processing (Aug. 1979)
- [12] Bic, L. and Hartmann, R. L., "AGM: A Dataflow Database Machine," ACM Tran. on Database Systems, 14, 1(Mar 1989), 114-146.
- [13] Bic, L. and Hartmann, R. L., "Simulated Performance of a Data-driven Database Machine," J. Paral. Distrib. Comput., 3, 2 (Mar. 1986), 1-22
- [14] Bilardi, G., "Merging and Sorting Networks with Topology of Omega Network," IEEE Trans. on Computers, 38, 10(Oct. 1989), 1396-1403.
- [15] Bilardi, G. and Nicolau, A., "Adaptive Bitonic Sorting: an Optimal Parallel Algorithm for Shared-memory Machines," SIAM J. on Computing, 18 (Apr. 1989), 216-218.
- [16] Bitton, D., DeWITT, D.J., Hsiao, D.K. and Menon, J., "A Taxonomy of Parallel Sorting," Computing Surveys, 16, 3 (Sept. 1984), 287-318.
- [17] Carlson, W.W., "Algorithmic Performance of Dataflow Multiprocessors," Computer, (Dec. 1985), 30-40.
- [18] Cole, Richard, "Parallel Merge Sort," SIAM J. Comput., 17 (Aug. 1988), 770-785.
- [19] Davis, A. L., A Data Flow Evaluation System Based on the Concept of Recursive Locality, In Proceedings of the National Computing Conference, AFIPS Press, Reston, Va., 1979, 1079-1086.
- [20] Dennis, J. B., "Data Flow Supercomputers," Computer (Nov. 1980), 48-56.
- [21] Dennis, J. B., First Version of a Data Flow Procedure Language, In Lecture Notes in Computer Science, Vol. 19, G. Goos and J. Hartmanis, Eds. Springer, New York, 1974, 362-376.

- [22] Dromey, R. G., How to Solve It by Computer, Prentice-hall International, INC., London, 1982.
- [23] Dwork, C., Kanellakis, P. C. and Stockmeyer, L., "Parallel Algorithms for Term Matching," SIAM J. Comput., 17 (Aug 1988), 711-731.
- [24] Faigle, U. and Turan, G., "Sorting and Recognition Problems for Ordered Sets," SIAM J. on Computing, 28, 1 (Jan 1985), 111-129.
- [25] Hartmann, R. L., A Database model and Its Implementation on a Highly Parallel Architecture, Ph. D. Dissertation, Deptment of ICS, University of California, Irvine, 1987.
- [26] Hazra, A., a Description Method and a Classification Scheme for Data Flow Architectures, in Proceedings of the 3rd International Conference on Distributed Computing Systems, IEEE, New York, Oct. 1982, 645-651.
- [27] Heide, F. and Wigderson, A., "The Complexity of Parallel Sorting," SIAM J. Comput., (Feb 1987), 100-107.
- [28] Horguchi, S., Nakada T. and Shigei, Y., "Experimental Evaluation of parallel Sorting on a Multiprocessor System," The Transactions of the IEICE, E 71, 2 (Feb. 1988), 127-131.
- [29] Hwang, Kai, Computer Architecture and Parallel Processing, McGraw-Hill, New York, 1984, 732-738.
- [30] Karp, R. M. and Miller, R. E., "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing," SIAM J. Appl. Math., 14, 6 (Nov 1966), 1390-1411.
- [31] Kapralski, a., "The Maximun and Minimum Selector SELRAM and Its Application for Developing Fast Sorting Machines," IEEE Tran. on Computers, 38, 11(Nov. 1989), 1572-1577.
- [32] Knuth, D. E., The Art of Computer Programming Vol 3 Sorting and Searching, Reading, MA: Addison-Wesley, 1973.
- [33] Komori, Shinji and Shima, Kenji, "The Data-Driven Microprocessor," IEEE MICRO, (June 1989), 45-59.

- [34] Lew, Art, Computer Science: A mathematical Instruction, Prentice-Hall International, INC., London, 1985.
- [35] Nassimi, D. N. and Sahni, S., "Bitonic Sort on a Mesh Connected Parallel Computer," IEEE Tran. Comput., C_27, 1(Jan 1979).
- [36] Paseman, W.G., "Applying Data Flow in the Real World," Byte (May 1985), 201-214.
- [37] Patton, C., "Data-flow Architecture Unclogs the Bottleneck of von Neumann Systems," Electronic Design (Feb 23, 1984), 60-61.
- [38] Pippenger, N., "Sorting and Selecting in Rounds", SIAM J. Comput., 16, (Dec 1987), 1032-1038.
- [39] Rajasekaran, S. and Reif, J. H., "Optimal and Sublogarithmic Time Randomized Parallel Sorting Algorithms,"SIAM J. Comput., (June 1989), 594-607.
- [40] Richards D., "Parallel Sorting - A Bibliography," SIGACT News, 18, 1 (Summer 1986), 28-48.
- [41] Rodriguez, J. E., A Graph Model for Parallel Computation, Tech. Rep. 64, Project MAC, Massachusetts Institute of Technology, Cambridge, Mass, 1969.
- [42] Rumbaugh, J., A Data Flow Multiprocessor, Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, Sagamore, N.Y., 1975, 220-223.
- [43] Schmeck, H, et al, "Systolic S -Way Merge Sort is Optimal," IEEE Trans. on Computers, 38, 7(July 1989), 1052-1056.
- [44] Srini, V. P., "An Architectural Comparison of Dataflow Systems," Computer, 19, 9 (Mar. 1986), 68-88.
- [45] Thakkar, S., Dataflow and Reduction Architectures, Computer Society Press of the IEEE, Washington, D.C.(1987).

- [46] Thompson, C. D. and Hung, H. T., "Sorting on a Mesh Connected Parallel Computer," Commun. ACM, 20(Apr. 1977), 263-271
- [47] Treleaven, P. C., Lima, G., "Future Computers: Logic, Data Flow,..., Control Flow?", Computer (March 1984), 47-55.
- [48] Treleaven, P. C., Brownbridge, D. R., and Hopkins, R. P., "Data-Driven and Demand-Driven Data Flow Computer," ACM Computing Surveys, 14, 1(March, 1982), 93-143.
- [49] Treleaven P. C. and Vanneschi, M., Future Parallel Computers, Springer-Verlag, Berlin, Germany, 1987.
- [50] Treleaven, P. C., "Principle Components of a Data Flow Computer," Proc. 1978 Euromicro Symp. (Oct. 1978), 366-374
- [51] Veen, A. H., "Dataflow Machine Architecture," ACM Comput. Surveys, 18, 4 (Dec, 1986), 365-396.
- [52] Vegdahl, S. R., "A Survey of Proposed Architectures for the Execution of Functional Languages," IEEE Trans. on Computers, C-23, 12 Dec. 1984, 1050-1071.
- [53] Winslow, L.E. and Chow, Y., "The Analysis and Design of Some New Sorting Machines," IEEE Trans. on Computers, c-32, 7 (July 1983), 677-683.
- [54] Wong, F. S. and Ito, M. R., "Parallel Sorting on a Recirculating Systolic Sorter," Comput. J., 27, 4(1984).
- [55] Yang, M., Huang, J. S. and Chow, Y., "Optimal Parallel Sorting Scheme by Order Statistics," SIAM J. Comput., 16 (Dec 1987), 990-1003.

VITA

Renmei Shu

Candidate for the Degree of
Master of Science

Thesis: THE USE OF DATAFLOW ARCHITECTURES FOR IMPROVEMENT
OF SORTING ALGORITHMS

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Wuhan, China, May 17, 1949, the daughter of Song-gui Shu and Zhi-ling Wang, Married

Education: Graduated from the high school attached to Beijing Institute of Technology, August 1966; received Bachelor of Science Degree in Automatic Control from Beijing Institute of Technology, Beijing, China, December, 1975; Completed requirements for the Master of Science Degree at Oklahoma State University, May, 1990

Professional Experience: Research and Teaching Assistant at Beijing Institute of Technology, Beijing, China 1975-79; Software Engineer/System Analysis at Beijing Institute of Technology, Beijing, China 1980-84; Graduate Assistant at Oklahoma State University, Stillwater, Ok, 1987-present.