NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

Department of Computer Engineering

GRADUATION PROJECT COM400

Parallel and Distributed Systems

Supervisor: Besime Erin
Submitted by: Meltem Asilsoylu
(960216)

JUNE,2000

CONTENTS

CHAPTER 1

CHAPTER 2

ABSTARCT

This project that I explain to parallel and distributed systems, horus,distributed computing system(DCE) ,mathematically applications of parallel and distributed system(Example branch and bound algorithm,load balancing,combinational obtimization).

There is a fundemental difference between them DCE,Horus and Mathematically parallel distributed systems.

This document summarize the main features of parallel and distributed systems,horus,DCE and mathematically applications of parallel and distributed system.What we feel is most important difference between them,discusses differences between individual capabilities and the maturity of both specifications and products,and cunculudes with view of how an organization should select technology most appropriate to its parallel and distributed systems.

I practice all graphic of parallel and distributed systems.

INTRODUCTION

The department applications of parallel and distributed systems has its traditional field of work in the complex areas of database and information systems.A multitude of system and application projects has been carried out,constantly exploring new subject areas.

There are several types of distributed processing systems in which the componnents are hooked together by telecommunications.List the main reasons for function distribution.Reasons for using hierarchical systems distribution.An important group of reasons on some configurations is related to data-where it is kept and how it is maintained.

Applications areas is computers can be employed for a wide variety of purpose but computers are particularly suited to certain kinds of work.It may be possible to use a computer for a particular application if certain criteria are met.Whether or not a computer is used will depend on other factors.Considers for criteria and factors and then deals with a variety of particular applications which illustrate the general princhiples.

We describe the main characteristics of distributed systems,their classification and programming techniques.Example demonstrate the application areas of distributed systems.

Horus a flexible group communication systems.

Networks of workstations runing under a multiuser multitasking operating system like unix are an increasingly commonplace personel computing environment.Due to their use as personal computing these workstations are typically underutilized most of the time.Branch and bound algorithms for combinatorial optimization on a cluster of workstations.Branch and bound algorithm can yield satisfactory speed-up on a cluster of workstations.The of describe parallezations of the brach-and-bound algorithm for multicomputers.We use as our problem domain the travelling salesperson described in all the parallel algorithms described in this section use the branching and bounding heuristics developed by little et al.The final part of this section discusses anomalies in parallel branc-and-bound algorithm.

General information about the Department

Applications of Parallel and Distributed Systems

Area of Responsibility

The department Applications of Parallel and Distributed Systems has its "traditional" field of work in the complex areas of database and information systems. A multitude of system and application projects has been carried out, constantly exploring new subject areas. The emphasis of the department's research are the following:

Area of Research

Databases and Transaction Management for Open Systems

In this research area we are determining how database management systems and their functionality can be used in open systems that do not comprise central administration components. The WWW (or the Intra-/Internet in general) might be seen as a typical example of such systems, but similar approaches including mobile access and processing facilities are also considered (e.g. OMG and OSF proposals). Evaluation of concepts is done by specific prototypes and related example applications like earth observations systems (EOS) or product data management (PDM).

Parallel Database Technology

New application areas with more demanding requirements or databases within open networks of information systems lead to a rapid increase of data volumes and query complexity. Well-known examples are the areas OLAP and data mining, profiling services or publish/subscribe services. The necessity to process large data sets very efficiently and to reduce response times requires parallel database technology. However, whenever possible, this should be transparent to the user. Parallelization steps which are considered in department projects are above all input/output, operators of the database engine, and measures to parallelize query execution plans.

Extensible Database Technology

The first commercial object-relational DBMS are available since some years. This technology promises to manage complex data structures with an efficient support for complex queries at the same time. Furthermore, DBMS vendors enable third party vendors to extend the base systems by program packages which are specific for an application area. These packages contain new data structures together with corresponding operators, support for more efficient query processing and meta data. An example is a package that adds support for geographic information systems by providing base functionality like geometric data structures, systems of coordinates, and spatial queries, for example. Further application areas like data mining and profiling are examined within this technology framework.

Query Processing in Database Systems

Today`s database systems are used in many diverse application areas (e.g. business and administration, engineering or knowledge-based applications) for efficient data management. The requirements for database processing and especially for query processing are mainly determined by prevailing development trends like e.g. extension of data models, use of multi-processor or multi-computer architectures for database servers or workstation/server environments. In this project area we develop conceptual and implementory foundations for advanced query processing. Furthermore we examine technologies that form a reusable and extensible base to built adapted query processors. These concepts allow to customize query processing engines to specific application scenarios. Obvious advantages of this approach are a substantially reduced development time, a flexible adaptivity as well as a high reuse of technology, implementation concepts and existing software.

Product Data Management and Exchange

This project deals with the management and exchange of product data, covering the design stage as well as efficient and long-term storage of data during the whole product lifecycle. Though relational and object-oriented DBS already offer some basic support, most applications in this area have been developed using other solutions. Special emphasis has been put on the International Standard for the Exchange of Product Data (STEP), which defines the object-oriented data definition language EXPRESS as well as the navigational access interface SDAI (Standard Data Access Interface).

Decision Support and Data Mining

Large databases generally contain important strategic information, which cannot be accessed by normal OLTP-applications. Thus special knowledge discovery methods are required, which are both CPU and I/O-intensive. Research in this field focuses on the development of strategies using parallelism as well as exploiting appropriate query optimization techniques.

Workflow Management

During the last two years, research in the field of workflow management has become more and more important. Since 1987, the research group Applications of Parallel and Distributed Systems is involved in projects about the management of long-lived activities and their combination with traditional, transaction-oriented methods. In 1990, the concept of ConTracts was created. Since then, this concept has been systematically extended to a platform for efficient, fault tolerant workflow management systems.

Flow Control in Design Applications

In integrated computer supported design environments the designflow manager has to keep any cooperation and interaction of the design activities consistent. Those activities comprise all designing interactions and CAD tools. There are different layers of activity abstraction to be found, which build in a natural and system integrated way the typical structure of flow management in a tool oriented design environment. A typical design flow can be divided into interacting sub design flows, which can be well defined as flow protocols on the different activity layers. This project aims at developing corresponding design concepts to support flexible and layer specific control flow management, which compose an overall control flow management. In doing so, already existing concepts in the areas of transaction processing, workflow systems, and group work should be considered.

Geographic Information Systems

With the growing interest in a computer supported presentation, administration and analysis of spatially referenced data the importance of geographic information systems (GIS) has increased significantly. The central component of GIS is a geographic or a spatial database system. These systems differ from the traditional data base management systems developed primarily for business and administration needs mainly by the fact that they offer special concepts and technologies especially for spatial data models and languages and efficient data structures for storage of and access to spatial data. In this project area we address questions concerning the system architecture of GIS, the design of geographical query languages, and especially the integration of geographic database systems based on object-relational technology.

Component Technology and Middleware

Modern information systems are characterized by a high degree of heterogeneity and distribution of software and hardware. This is especially true for most projects of the department "AS" (e.g. earth observation systems, geographical information systems, product data management systems, design control, etc). Moreover, these systems mostly consist of autonomous components (like e.g. database applications, TP monitors or user interaction). In order to cope with the complexity of such systems, it is necessary to model the entire system as a combination of separate components that interact via appropriate middleware. In this sense, the term middleware comprises file and database management systems, TP monitors, network services and e-mail systems, workflow systems as well as more complex frameworks like CORBA or DCOM. In this project area we will deal with the design and efficiency of such component-based systems. Particular emphasis will be put on efficient data access and data shipping techniques. The main goal is to design, implement and evaluate concepts for distributed (heterogeneous) data management supporting STEP-based applications, e.g. CAD tools or bill-of-material processing. Due to heterogeneity and distribution of components, modern concepts which support Intranet and Internet (like CORBA and Java) will be used and evaluated as well.

Models and Tools for Parallel Programming

Based on computationally intensive problems, as they can be found in typical engineering applications, we develop approaches which do not - as is usually the case - take the algorithmical structure of a given problem for granted, but which modify some base mechanisms taken from the database field in such a way that they can be used for parallel programming environments in a variety of applications. One of the main goals is to transfer the potential of automatic parallelism - which has proven successful for descriptive database queries - to the programming of different applications. A very interesting idea is to use a two-tier programming model; at one level the numerical algorithms are coded in conventional sequential style, at the second level the topology of the problem is specified in a way that allows for automatic detection and control of parallelism.

TYPES OF DISTRIBUTED SYSTEMS

There are several types of distributed processing systems in which the components are hooked together by telecommunications. This chapter categorizes them and gives examples.

HORIZONTAL VS. VERTICAL DISTRIBUTION

First we shall distinguish between horizontal and vertical distribution.By vertical distribution we mean that there is a hierarchy of processors, as in Fig.1. The transaction may enter and leave the computer system at the lowest level. The lowest level may be able to process the transaction or may execute certain functions and pass it up to the next level. Some, or all, transactions eventually reach the highest level, which will probably have access to on-line files or data bases. The machine at the top of a hierarchy might be a computer system in its own right, performing its own type of processing on its own transactions. The data it uses is, however, passed to it from lower-level systems.

The machine at the top might be a head-office system which receives data from factory, branch, warehouse, and other systems.

By horizontial distribution we imply that the distributed processors do not differ in rank. They are of equal status-peers-and we refer to them as peer-coupled systems. A transaction may use only one processor, although there are multiple processors available. On some peer-coupled systems a transaction may pass from one system to another, causing different sets of files to be updated.

Figure.2 illustrates horizontal distribution. The top diagram shows multiple processors connected to a bus or wideband short-distance channel. The second diagram shows multiple processors connected to a loop, perhaps spanning several buildings in a factory complex, university campus, or shopping center, but in some systems being comprised of long-carrier connections.The
third and fourth diagrams show horizantal computer networks in which a user may access one of many machines.

Types of Distributed Systems

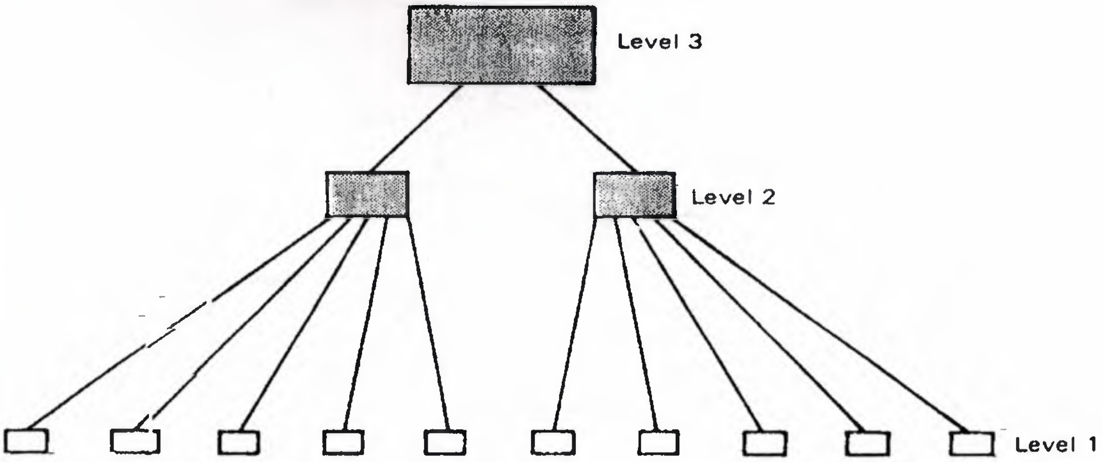


Figure 1 Vertical distribution.

COOPERATIVE OPERATION

In some networks the user has a choice of computer systems available to him, but he normally employs only one computer at a time. The computers are programmed independently, and each computer performs its own functions. In other networks the computers are programmed to cooperate with one another to solve a common set of problems. This is often the case in a vertical system (Fig.1). The lower-level machines are programmed to pass work to the higher-level ma- chines. This is sometimes true also in a horizontal system. The processing of one transaction may begin on one machine and pass to another. The different computers perform different functions or maintain and update different files. The machines may be minicomputers in the same location or computers scattered across the world on a network.

FUNCTION DISTRIBUTION VS. SYSTEM DISTRIBUTION:

In some distributed systems, usually vertical systems, functions are distributed, but not the capability to fully process entire transactions. The lower-level machines in Fig.1 may be intelligent terminals or intelligent controllers in which processors are used for functions such as message editing, screen formatting, data collection dialogue with terminal operators, security, or message compaction or concentration. They do not complete the processing of entire transactions.

Werefer to this distribution as function distribution and contrast it with system such as message editing, screen formatting, data collection dialogue with terminal operators, security, or message compaction or concentration. They do not complete the processing of entire transactions. We refer to this distribution as function distribution and contrast it with system distribution in which the lower-level machines are system in their own right, processing their own transactions ,and occasionally passing transactions or data up the hierarchy to higher level machines

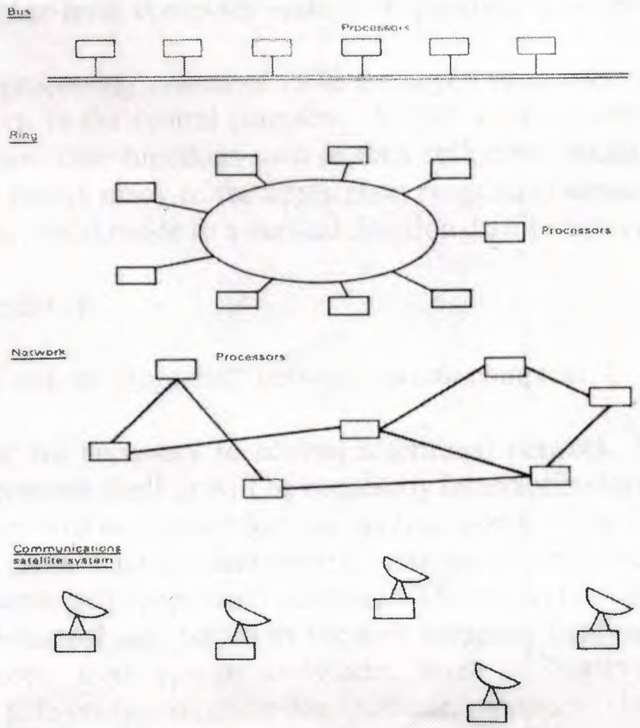
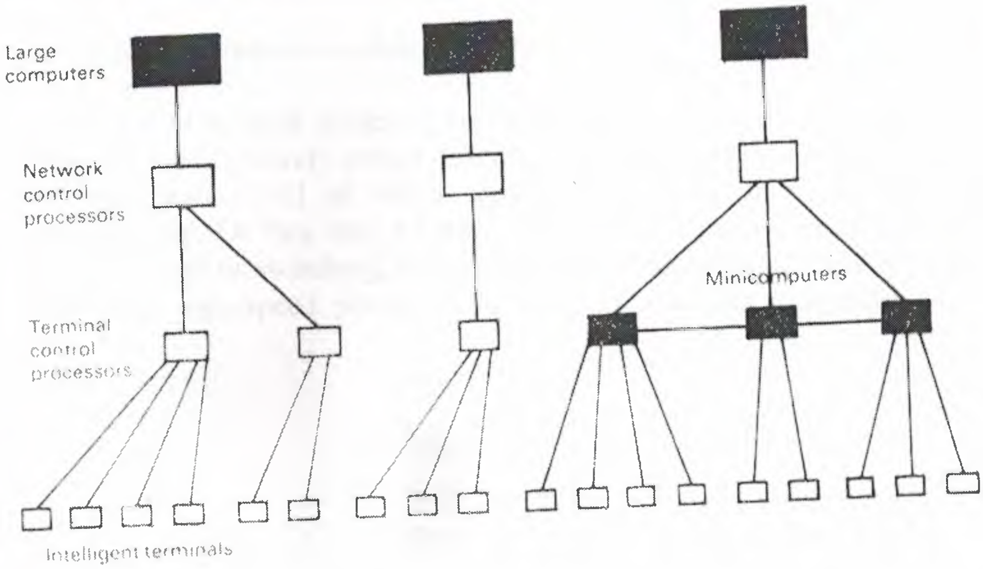


Figure 3.2 Horizontal distribution.

In a systems distribution environment the lower machines may be entirely different from, and incompatible with, the higher machines. In a function distribution environment, close cooperation between the lower-level and higher-level machines is vital. Overal system standards are necessary to govern what functions are distributed and exactly how the lower and higher machines form part of a common system architecture with appropriately integrated control mechanisms and software.



FUNCTION DISTRUBUTION

When the peripheral nodes are not self-sufficient systems but perform a function subservient to a higher-level distant computer, we speak of intelligent terminals, intelligent terminal cluster controllers, or intelligent concentrators. These terms imply a vertical distribution of function in which all or most transactions have to be transmitted, possibly in a modified form, to a higher-level computer system, or possibly to a network of higher-level computer systems.

The centralized teleprocessing system of 1970 employed simple terminals and carried out almost all of its functions in the central computer. At first system control and housekeeping functions were moved out, then functions such as data collection, editing, and dialogue with terminal operators, and finally many of the application programs themselves. Figure 4 shows places where intelligence could reside in a vertical function distribution system:

1. In the host computer, B

2. In a line control unit or "front-end" network control computer, C

Many function are necessary to control a terminal network .If the host computer performs all the operations itself ,it will be constantly interrupting its main processing, and many machine cycles will be needed for line control. Some of the line control functions may be performed by a separate line control unit. In some systems, all of them are performed by a separate and specialized computer. The proportion of functions which are performed by a line control unit, which by the host computer hard- ware and which by its software, varies widely from system to system. Some application functions could be performed by the subsystem computer-for example, accuracy checking and message logging.

A major advantage of using a front-end network-control computer is that when the host computer has a software crash or brief failure, the network can remain functionally operational. Restart and recovery of the network without errors or lost transactions is a tedious and often time-consuming operation, and if it happens often it can be very frustrating to the end users.

3. In the mid-network nodes, D and E

The mid-network nodes or concentrators may take a variety of different forms. They may be relatively simple machines with unchangeable logic. They may have wired-in logic, part or all of which can be changed by an engineer. They may be micro programmed. Or they may be stored-program computers, sometimes designed solely for concentration or switching, but sometimes also capable of other operations and equipped with files, high-speed printers, and other input-output equipment.

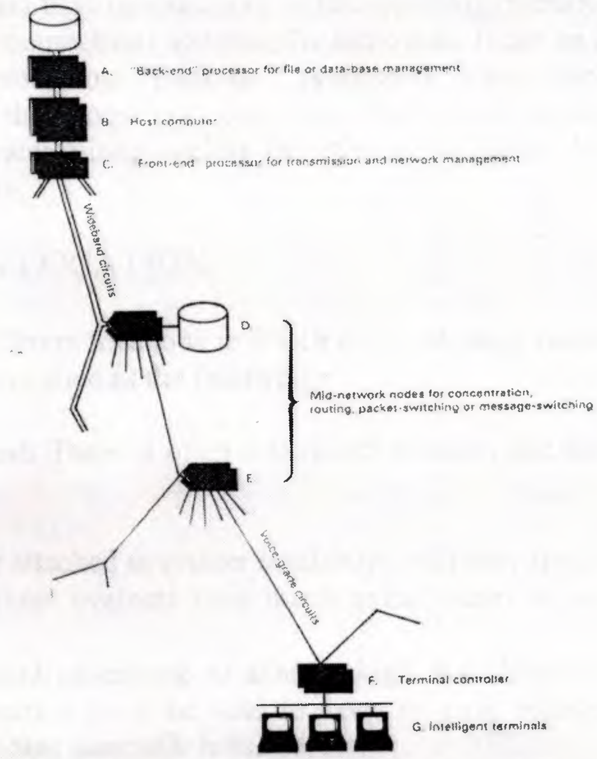

A. "Back-end" processor for file or data-base management
B. Host computer
C. "Front-end" processor for transmission and network management
Wideband circuits
D
Mid-network nodes for concentration, routing, packet-switching or message-switching
E
Voice-grade circuits
F. Terminal controller
G. Intelligent terminals

Figure 3.4 Places where intelligence can reside in a distributed-intelligence network.

4. In the terminal control unit, F

Terminal control units also differ widely in their complexity, ranging from simple hardwired devices to stored-program computers with much software. Increasingly they are computers with storage units and there is a trend towards greater power and larger storage. They may control one terminal or many. They may be programmed to interact with the terminal operator to provide a psychologically effective dialogue in which only an essential kernel is transmitted to or from the host computer. They may generate diagrams on a graphics terminal or interact with the operator's use of a light pen. They are often the main component in carrying out the assortment of distributed functions which this chapter will list.

5. In the terminal, G

"Intelligent terminals" are becoming more intelligent. Their processing functions range from single operations such as accumulating totals in a system which handles financial transactions, to dialogues with operators involving much programming. Some intelligent terminals do substantial editing of input and output data. Some terminals perform important security functions.

Where several terminals share a control unit, F, such functions are probably better performed in the control unit, leaving the terminal s simple inexpensive mechanism in which the main design concern may be tailoring the keyboard and other operator mechanisms to the applications in question.

6. In a "back-end" file or data base management processor, A

File or data base operations may be handled by a "back-end" processor. This can carry out the specialized functions of data base management or file searching operations. It can prevent interference between separate transactions updating the same data. It can be designed to give a high level of data security protection. "Back-end" processors, where they exist today, are normally cable-connected to their local host computer. They could, especially when high-bandwidth networks or communications satellite facilities are available, be remote from the host computers which use them.

CHOICE OF FUNCTION LOCATION

The designer, faced with different locations in which he could place functions, may choose his configuration with objectives such as the following:

1. **Minimum total system cost.** There is often a trade-off between distributed function cost and telecommunications cost.

2. **High reliability.** The value attached to system availability will vary from one system to another. The systems analyst must evaluate how much extra money is worth spending on duplexing ,
alternate routing and distributed processing to achieve high availability. On some systems reliability is vital. A supermarket must be able to keep its cash registers going when a communication line or distant host computer fails.

3. **Security .** In some systems function distribution is vital for system security (as we discuss later).

4. **Psychologically effective dialogue with terminal users.** Function distribution is used to make the dialogue fast, effective and error-free.

5. **Complexity.** Excessive complexity should be avoided. The problems multiply roughly as the square of the complexity.

6. **Software cost.** Some types of function distribution occurring throughout a network incur a high programming expenditure. The use of stored- program peripheral machines may inflate cost.

7. Flexibility and expandability. It is necessary to choose hardware and software techniques that can easily be changed and expanded later especially because telecommunications and net-working technology are changing so fast. Some approaches make this step difficult.

REASONS FOR FUNCTION

Lists the main reasons for function distribution. They fall into three categories:

1.Reasons associated with the host

Many machine instructions are needed to handle all of the telecommunications functions. The load on a central machine could be too great if it had to handle all of these functions. A single computer operates in a largely serial fashion executing one instruction at a time. It seems generally desirable to introduce parallelism into computing so that the circuits execute many operations simultaneously. This is the case when machine functions are distributed to many small machines.

2. Reasons associated with the network

There are many possible mechanisms which can be used to make the network function efficiently. We will discuss them later in the book, These mechanisms are, used to lower the overall cost of transmission and increase its reliability. The network configuration is likely to change substantially on most systems, both because of application development and increasing traffic, and because of changes in networking technology which are now coming a fast and furious rate. Function-distribution may be used to isolate the changing network from parts of the system so that the other parts do not have to be modified as the network changes. The term network transparency is used to imply that changes which occur in the network be not evident to and not affect the users.

REASONS FOR FUNCTION DISTRIBUTION

1. **Psychologically Effective Dialogues**

- **Local interaction.** Much of the dialogue interaction takes place locally rather than being transmitted, and hence can be designed without concern for transmission constraints.

- **Local panel storage.** Panels or graphics displayed as part of the dialogue can be stored locally.

- **Speed**

Local responses are fast. Time delays which are so frustrating in many terminal dialogues can be largely avoided. The delays that do occur when host response is needed can be absorbed into the dialogue structures.

2. **Reduction of Telecommunications Costs**

- **Reduction of number of messages.**
  In many dialogues the number of messages transmitted to and fro can be reduced by an order of magnitude because dialogue is carried on within the terminal or local controller.

- **Reduction of message size.**
  Messages for some applications can be much shortened because repetitive information is transmitted.

- **Reduction of number of line turnarounds.** Because the number of messages is reduced; and because a terminal cluster controller or concentrator can combine many small messages into one block for trans- mission.

- **Bulk transmission.**
  Nontime-critical items can be collected and stored for later batch transmission over a switched connection.

- **Data compaction.**
  There are various ways of compressing data so that fewer bits have to be transmitted. This effectively increases the transmission speed.

- **Minimum cost routing.** The machine establishing a link could attempt first to set up a minimum-cost connection, e.g., a corporate tie-line network. If these are busy it could try pro- gressively more expensive connections (e.g., WATS, direct distance dialing).

- **Controlled network access.**
  Terminal users may be prevented from making expensive unauthorized calls.

3.Reliability

- **Local autonomy.**
  A local operation can continue, possibly in a fallback mode (using a minimal set of functions), when the location is cut off from the host computer by a circuit, network, or host failure. On certain systems this is vital

- **Automatic dial backup.**
  A machine may be able to dial a connection if a leased circuit fails
- **Automatic alternate routing.**
  A machine may be able to use an alternate leased circuit or network path when a network failure occurs.
- **Control procedures.**
  Control procedures can be used to recover from errors, or failures and to ensure that no messages are lost or double-processed.
- **Automatic load balancing.**
  A machine may be able to dial an extra circuit or use a different computer to handle high traffic peaks.

4. Less Load on Host

- **The Parallel operations.**

The parallel operation of many small processors relieves the host computer of much of its work load, and lessens the degree of multiprogramming. In some systems this is vital because the host is overburdened with data base operations.

- **Permits large numbers of terminals.**

Some systems require too many terminals for it to be possible to connect them directly to a host computer. Distributed control and operations make the system possible.

5. Fast Response Times

- **Process mechanisms.**

Local controllers can read instruments rapidly and give a rapid response to process control mechanisms when necessary.

- **Human mechanisms.**

Fast reaction is possible to human actions such as the use of a plastic card or the drawing of a curve with a light pen.

- **Dialogue response items**.

Dialogues requiring fast response times (such as multiple menu selection) can be handled by local controllers.

6. Data Collection

- **Data entry terminals.**

Many inexpensive data entry terminals (for example, on a factory shop floor) can be connected to a local controller which gathers data for later transmission.

- **Local error checking.**

Local checks can be made on the accuracy or syntax of terminal entries. An attempt is made to correct the entries before transmitting them to the host.

- **Instrumentation.**

Local controllers scan or control instruments, gathering the result for transmission to a host computer.

7. More Attractive Output

- **Local editing.**

Editing of output received at terminals can lay out the data attractively for printers or screen displays. Repetitive headings, lines, or text, and page numbers can be added locally. Multiple editing formats can be stored locally.

8. Peaks

- Interactive and real-time systems often have peaks of traffic which are difficult or expensive to accommodate without function distribution. Storage at the periphery allows the peak transactions to be buffered or filed until they can be transmitted and processed economically.

9. Security

- **Cryptography.**

Cryptography on some systems gives a high measure of protection from wire- tapping, tampering with magnetic-stripe plastic cards, etc. Cryptography is vital on certain electronic fund transfer systems.

- **Access control.**

Security controls can prevent calls from unauthorized sources from being accepted, and prevent terminals from contacting unauthorized machines.

10. Network Independence

- **Network transparency.**

Programmers of machines using networks should not be concerned with details of how the network functions. They should simply pass messages to the network interface and receive messages from it.

- **Network evolution.**

As networks grow and evolve, and as different networks are merged, programs in machines using the networks should not have to be rewritten.

- **New networks.**

Network technology is changing fast. As applications are switched to new types of networks (e.g:, DDS, value-added networks, Datadial, satellite networks); the programs in the using machines should not have to be rewritten.

11. Terminal Independence

- **New terminals.**

Terminal design is changing fast. If a new terminal is substituted, the old pro grams should not have to be rewritten. Software in terminal controllers may make the new terminals appear like the old.

- **Virtual terminal features.**

Application programs may be written without a detailed knowledge of the terminal that they will use. For example, the screen size or print-line size may not be known. The programmers use specified constraints on output, and the distributed- intelligence mechanisms map their output to the device in question.

Mechanisms relating to the network may reside in any of the locations indicated in Fig. 4. A terminal or a controller for a cluster of terminals may have mechanisms intended to minimize the transmission cost. A .front-end communications processor may relieve the host of all network functions, and maintain network operations without loss of data if the host or its software fails. Intelligence may also reside in midnetwork nodes such as packet-switching devices, concentrators, intelligent ex- changes, or telephone company equipment in systems such as AT&T's ACS. The phrase "intelligent network" is increasingly used to imply that the network itself uses , computers to share transmission links or other resources in an efficient, dependable manner.

3. Reasons associated with the end user

Probably the most important of the three categories is that associated with the end user. On many systems built prior to the era of function distribution, the dialogue that takes place between the terminal and its operator is technically crude. It is often difficult for the user to learn, and clumsy and frustrating in operation. The user is forced to learn mnemonics and to remember specific sequences in which items must be entered. The response times are often inappropriate. The majority of the users who should be employing terminals are unable to make the machines work, and generally discount the possibility of ever using them because they perceive them as being difficult-designed for technicians, programmers, or a specially trained and dedicated staff. One psychologist describes many of these user-terminal interfaces as "unfit for human consumption."

In the past there has been good reason for the crudity of terminal dialogues. The terminals had no intelligence. Every character typed and displayed had to be transmitted over the network. The network often used leased voice lines serving many terminals, and to minimize the network cost, the number of characters transmitted was kept low. The response times were often higher than psychologically appropriate because of the queries on the lines.

With intelligent terminals or controllers the dialogue processing can take place in the local machine. Most of the characters are not transmitted over the telephone lines. The only characters transmitted are those which take essential information to the central computer and carry back essential information to the terminal. These characters will often be only a small fraction of the total characters typed and displayed in a psychologically effective dialogue.

Much of the future growth of the computer industry is dependent on making the machines easy to use and understand for the masses of people in all walks of life who will employ them, and distributed intelligence can play a vital part in this.

HIERARCHICAL DISTRIBUTED PROCESSING

So far this chapter has discussed function distribution in which the peripheral machines are not self-sufficient when isolated from their host by a telecommunications or other failure. Now let us expand the discussion to processing distribution in which the peripheral processors keep their own data and can be self- sufficient, but which are connected to higher-level systems.

There is not necessarily a sharp boundary line between function distribution and system distribution. In some cases there has tended to be growth from function distribution to system distribution, with more and more power being demanded in peripheral machines. In other cases the peripheral machines started as standalone minicomputers and became linked into a higher-level system.

The application programming steps for most (but not all) commercial transactions do not require a large computer. Small, inexpensive, mass-produced processors such as those discussed in

the previous chapter could usually handle the whole transaction.They would handle it with a much smaller software path length than a large computer .The difference in software path length greatly reinforces the arguments about there no longer being economies of scale.Some large mainframes with complex data base management systems use more than 100,000 software instructions per transaction and only a few thousand application instructions per transaction.

In some cases there are good reasons for storing the data which a transaction requires centrally

In other cases the data also can be kept in storage attached to the local machine.

As we commented earlier, criteria for determining whether a transaction is transmitted could be:

1. **It needs the power of a large computer**

2. **It needs data which are stored centrally**

If one of these criteria does not apply, then the transaction is processed locally. Most commercial transactions and many scientific calculations do not need the power of a large computer. There are exceptions such as simulations and complex models. Many of these exceptions would not use the teleprocessing anyway. But the second criterion-centralized data-is important to some, but not all, data. Consequently data base and data communications techniques are closely related, and computer manufacturers produce data base, data communications (DBDC) software.

EXAMPLES OF HIERARCHICAL CONFIGURATIONS

Some examples of hierarchical configurations are as follows:

1.Insurance

The branches of an insurance company each have their own processor with a printer and terminals. This processor handles most of the computing requirements of the branch. Details of the insurance contracts made are sent to a head office computer for risk analysis and actuarial calculations. The head-office management has up-to-the- minute information on the company's financial position and exposure, and can adjust the quotations given by the salesmen accordingly.

2. A chain store

Each store in a chain has a minicomputer which records sales and handles inventory control and accounts receivable. It prints sales slips (receipts) for customers at the time of sale. Salesmen and office personnel can use the terminals to display pricing inventory and accounts receivable information, and customer statements. The store management can display salesman
performance information and goods aging and other analysis reports.

The store systems transmit inventory and sales information to the head office system.At night they receive inventory change information.The fast receipt of inventory and sales information enables the head office system to keep the inventory of the entire organization to a minimum.

The store systems run unattended.Any program changes are transmitted to the systems from the head office computer.

3. Production control

Various different production departments in a factory complex each have a mini-computer. Work station terminals on the shop floor are connected to the minicomputer and the workers enter details of the operations they perform. The task of scheduling the operations so as to make the best utilization of men and machines is done by the minicomputer.The shop foreman displays these operations schedules and often makes changes to them because of local problems and priorities.He frequently makes a change and instructs the machine to reperform itsscheduling program.

Details of the work to be done are made up by a higher-level computer which receives information about sales and delivery dates,and performs a gross and net breakdown of the parts that must be manufactured to fill the orders.The central computer passes its job reqirements to the shop floor minicmputers, and receives status reports from them.

PROCESS CONTROL

Hierarchies of processors were common in process control applications before they were used in commercial data processing. Many instruments taking readings in an industrial or chemical process are connected to a small reliable computer which scans the readings looking for exceptions or analyzing trends. The same computer may automatically control part of the operation, setting switches, operating relays, regulating temperatures, adjusting values, and so on.

Response time must be fast on some process control applications. A local mini-computer is used to ensure fast response. Increasingly today, tiny cheap microprocessors are being employed in instruments and control mechanisms. Many such devices may be attached to a minicomputer which stores data relating to the process being controlled. A higher-level computer may be concerned with planning the operations, optimization, providing information for management control, or general data processing. Figure 6 shows a configuration in a steel mill, with different processors each having its own two-level process-control system, with these systems being linked to a higher production planning system.

In hospitals, the elaborate patient instrumentation used in intensive-care wards is monitored and controlled by small, local and highly reliable computers. These in turn are linked to higher-level machines which can perform complex analyses, provide in- formation to stations, record patient histories, and so on.

CAUSALLY COUPLED

In some configurations the design of the peripheral systems is largely independent of the design of the higher- level systems. In others the periphery and the center are so closely related that they are really separate components of the same system.

An example of a causally coupled configuration is a corporate head-office information system which derives its data from separate systems, separately installed in different corporate departments. These systems transmit data at the end of the day to the control system where it is edited, reformatted, and filed in a different manner to that in the peripheral systems, to serve a different purpose. The installers of the peripheral systems designed them for their own needs and were largely unaware of the needs of the central system. An example of a closely coupled design is a banking system in which all customer data is stored by a central computer. (This does not apply to all banks. Some have loosely distributed systems.) A small computer in each branch, or group of branches, serves the processing needs of that branch, providing the tellers and the officer with the information they need at the terminals. Customer data is also stored in the branch com- largely in case of a failure of the central system or the telecommunications link to it. The peripheral files are strictly subsets of the central file. The programs developed for the peripheral computers are compiled on the central computer, and loaded from it into the peripheral computers. Changes in the peripheral programs are made centrally and transmitted. Account balancing requires tight cooperation of the peripheral and central machines.

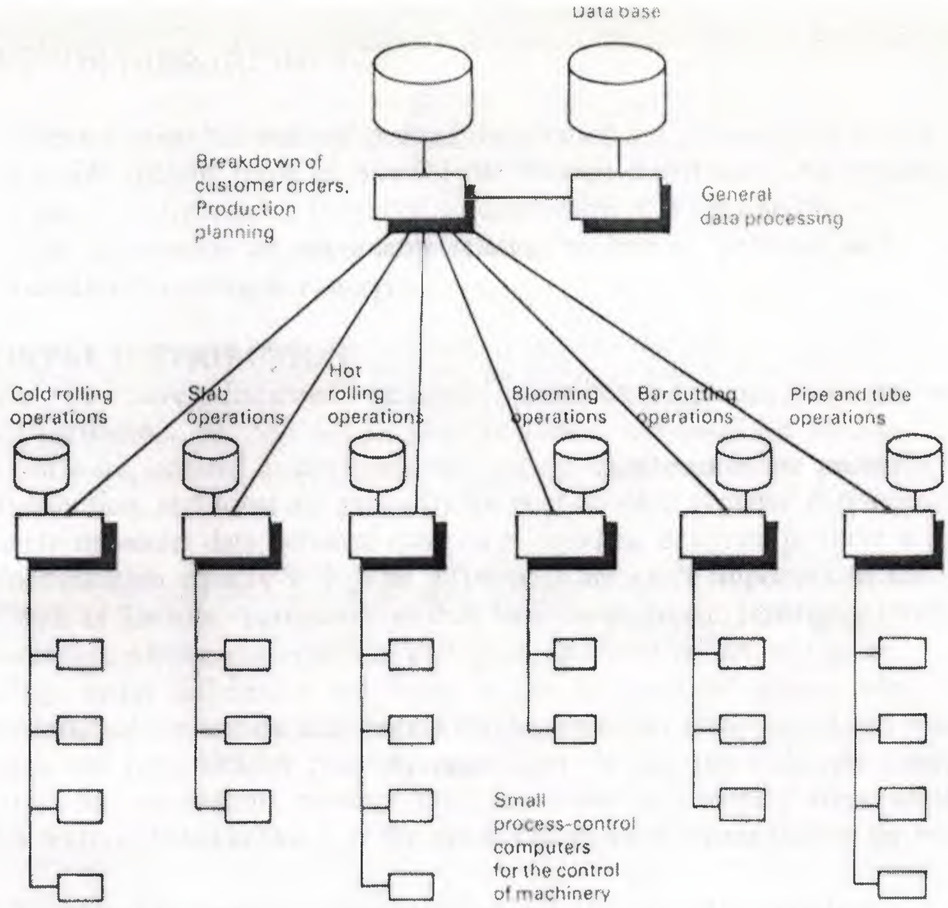


Figure 6 a hierarchy of computers in a steel mill which integrates the process control in several plant areas, and production planning. The system gives higher productivity of plant operations and permits immediate response to customer orders.

MULTIPLE LEVELS

Vertically distributed configurations may contain more than two levels of processor. In some there may be as many as four levels .

The lowest level may consist of intelligent terminals for data entry, or microprocessors in a factory, scanning instruments.

The next level may be a computer in a sales region assembling and storing data that relates to that region, or a computer in a factory assembling the data from the microprocessors and being used for production planning.

The third level is a conventional large computer system in the divisional head office, performing many types of data processing and maintaining large data bases for routine operations. This computer center receives data from the lower systems and sends instructions to them.

The highest level is a corporate management information system, with data structured differently from that in the systems used for routine operations. This system may be designed to assist various types of high-management decision making. It may run complex corporate financial models or elaborate programs to assist in optimizing certain corporate operations, for example, scheduling a tanker fleet. It receives summary data from other, lower systems.

REASONS FOR HIERARCHIES

Reasons for using hierarchical systems distribution are, summarized in Box 3.2. The set of reasons should include those in Box 3.1 on function distribution. An important group of reasons on some configurations is related to data-where it is kept and how it is maintained. Also of great importance are arguments relating to human, political, and organizational reasons, in addition to technical reasons .

HORIZONTAL DISTRIBUTION

So far we have discussed vertically distributed systems. Now we will consider horizontal distribution.

Some software, control mechanisms and system architectures are primarily oriented to vertical distribution, and some are primarily for peer-coupled systems. A transport subsystem which merely transmits data between computers could be designed to serve a horizontal or vertical configuration equally well. The differences are more important in the higher-level activities such as file man- agreement, or data base management, intelligent terminal control, data compression, editing, man-machine dialogues, recovery, restart, and so on.

In reality, major differences are found in the transport subsystems also. A transport subsystem designed for vertical distribution can have simpler flow control and routing control mechanisms, and have simpler recovery procedures. It may use elaborate concentrators or other devices for maximizing network utilization, and may employ some of the function distribution features listed in Box 3.1. We discuss these mechanisms later in the book.

BOX 3.2 Technical reasons for using hierarchical distributed processing

(Note there are also human, political, and organizational reasons which are often more important than these technical reasons.)

- **Cost.**

Total system cost may be lower. There is less data transmission and many functions are moved from the host machine.

- **Capacity.**

The host may not be able to handle the workload without distribution. Distribution permits many functions to be performed in parallel.

- **Availability.**

Fault tolerant design can be used. Critical applications continue when there has been a host or telecommunications failure. The small peripheral processors may be substitutable. In some systems high reliability is vital; e.g., a supermarket system, or hospital patient monitoring.

- **Response time.**

Local responses to critical functions can be fast; no telecommunications delay; no scheduling problems; instruments are scanned and controlled by a local device.

- **User interface.**

A better user interface can be employed, e.g., better terminal dialogue, when the user interacts with a local machine; also better graphics or screen design; more responses, faster response time.

· **Simplicity.**
Separation of the peripheral functions can give a simpler, more modular system design.

· **More function.**
More system functions are often found because of ease of implementing them on the peripheral machines. Salary savings often result from increased peripheral functions.

· **Separate data organizations.**
The data on the higher-level system may be differently organized from those on the peripheral systems (e.g., corporate management information organized for spontaneous searching versus local detailed operational data tightly organized for one application).

Reasons for horizontal computer networks

· **Resource sharing.**
Expensive or unique resources can be shared by a large community of users, as on ARPANET.

· **Diversity.**
Users have access to many different computers, programs, and data banks.

.Transaction interchange
Transactions are passed from one system to another or from one corporation to another: e:g., financial transactions passed between banks on SWIFT; airline reservations or messages passed between computers in separate airlines, as on SITA.

· **Separate systems linked.**
Separate previously existing systems are linked so that one can use another's data or programs, or to permit users to access all of them.

· **Local autonomy.**
Local autonomous minicomputer systems are favored, with their own files, and some transactions need data which reside on the file of a separate system.

· **Functional separation.**
Instead of one computer center performing all types of work, separate centers specialize in different types. For example, one does large-scale scientific computation. One does information retrieval. One has a data base for certain classes of application. One does mass printing and mailing.

· **Transmission cost.**
Separate systems share a common network designed to minimize the combined data (and possibly voice) transmission cost.

· **Reliability and security.**
When one system fails, others can process transactions. If one system is destroyed, its files can be reconstructed on another.

· **Load sharing.**
Unpredictable peaks of work on one machine can be off-loaded to other machines.

· **Encouragement of development.**

A corporate network can permit small data processing groups to develop applications.

PATTERNS OF WORK

Because of the mechanisms built into software or systems architecture, designers sometimes try to make all configurations vertical, or all configurations horizontal. This can result in excessive aver- head, system inflexibility, or clumsy control. Whether or not a configuration should be vertical, or horizontal, or both, depends upon the patterns of work the configuration must accomplish and the patterns of data usage.

In designing a distributed system we are concerned with such questions as:

· Where are the units of processing work required?
· How large are these units? What size of processing machine do they need?
· Are the units independent, or does one depend on the results of another?
· What stored data do the work units employ?
· Do they share common or independent data?
·What transactions must pass between one unit and another? What are the patterns of transaction flow?
· Must transactions pass between the units of work immediately, or is a delay acceptable? What is the cost of delay?

The answers to these questions differ from one organization to another. The patterns of work are different. The patterns of information flow between work units are different. Different types of corporations tend, therefore, to have their own natural shapes for distributed processing. What is best for an airline is not necessarily best for an insurance company.

The nature of the work units may be such that they can be independent of one another and have no need to know what each of the others is doing. They may be standalone units having no communication with any other unit-possibly standalone minicomputers. On the other hand, they may need to share common data which resides centrally. In this case there are vertical links to a common data store. There may be multiple such data stores which themselves pass information to a higher system. Alternatively the work units at one level may be such that they need to pass information to other units at the same level. This situation may lead naturally to hori- zontal communication; but it could also, if necessary, be handled vertically with a centralized processor relaying transactions between the units.

EXAMPLES

1. An airline reservation system requires a common pool of data on seat availability. Geographically scattered work units use, and may update, the data in this pool. Each of them needs data which is up-to-date second by second. This data needs to be kept centrally. The bulkiest data are those relating to passengers. A passenger may telephone the airline in cities far apart; when he does so the agent to whom he talks must be able to access needed data. In order to rind the data it is easier to keep it centrally also.

2. A car rental firm may permit its customers to pick up a car at one location and leave it at another. When the car is picked up a computer terminal prepares the contract. When the car is left a terminal is used to check the contract and calculate the bill. If a minicomputer at each location performed these functions, horizontal communication would be needed between the destination location and the location where the car was picked up. However, some centralized work is also needed because it is necessary to keep track of the company's cars and ensure that they are distributed appropriately for each day's crop of customers. Credit and other details about regular customers may also be kept centrally. The shape of the work therefore indicates both vertical and horizontal dis- tribution. However, because the centralized (vertical) links are needed, the customer con- tracts may also be kept centrally and the same links used to access them. The rental offices may then use intelligent terminals rather than complete minicomputers.

3. Insurance companies have offices in different locations. They keep details about customers and their policies. An office does not normally need to share these data with another office or pass transactions to it. The offices could therefore use standalone machines. Customers in different locations may have different requirements. In the U.S. different states have different insurance regulations and tax laws. The different machines may therefore be programmed some what differently. The insurance company's head office, however, needs to know enough details of all customers policies to enable it to evaluate the company's cash flow, and risks, and to perform actuarial calculations which enable it to control the company's financial exposure. Enough data for this purpose is therefore passed upwards to the head office. This vertical communication does not need to be real-time, as in the case of an airline reservation system. It can be transmitted in periodic batches.

Although the pattern of the work in an insurance company is appropriate for a decentralized system, that does not necessarily mean that a decentralized system will be the cheapest or best. There are various arguments for centralization, among them economies of scale, centralized control of programming, and use of data base software. A function-distribution rather than a processing-distribution configuration is used in some insurance companies.

4. In a group of banks, each handles its own customers with its own data processing system. A customer in one bank, however, can make monetary transfers to customers in other banks. A network is set up by the banks to perform such transfers electronically. The money is moved very rapidly and hence is available for use or interest-gathering by banks for a longer period. The use of this ' `float" more than pays for the network. In this example we have a peer-coupled configuration with need for a horizontal transfer between the work units.

DEGREE OF HOMOGENEITY :

We may classify horizontal configurations according to the degree of homogeneity of the systems which communicate. This affects the design, the choice of software and network techniques and, often, the overall management. At one extreme we have identical machines running the same application pro- grams in the same corporation. In other words the processing load has been split between several identical computers. At the other extreme we have incompatible ma- chines running entirely different programs in different organizations, but nevertheless interconnected by a network. One of the best known examples of this is ARPANET, interconnecting university and research centers.

NONCOOPERATIVE SYSTEMS

We may subdivide configurations into those composed of cooperative and non cooperative systems. A non cooperative configuration consists of computer systems installed independently by different authorities with no common agency controlling their design, but linked by a common shared network.

When the networking capability becomes accepted and understood by the various system development groups, there may be slightly less non cooperation. Developers know that a certain data base exists on another system. They may learn to think in terms of interchanging. data, sharing resources, and establishing compatible transaction formats.

Because the cost and ease of networking will improve greatly in the future, some corporations have attempted to impose certain standards on their diverse systems groups, which will eventually make interconnection of the systems more practical or more valuable. Among the types of standards imposed or attempted have been the following:

1. Standardization of transaction formats.
2. Standardization of line control discipline.
3. Use of compatible computers (one large corporation decreed that all minicomputers should be DEC machines, possibly anticipating future use of DEC's network architecture).
4. Standardization of data field formats and use of an organization-wide data dictionary.
5. Standardization of record or segment formats.
6. Use of a common data description language (e.g., CODASYL DDL, or IBM's DLJI)
7. Use of a common data base management software.
8. Use of a common networking architecture.

COOPERATING SYSTEMS

Cooperating systems are designed to achieve a common purpose, serve a single organization, or interchange data in an agreed-upon manner. We can subdivide cooperating systems into those in which the separate systems are used by the same organization and those in which separate corporations are interlinked.

Networks which interlink separate corporations are found today in certain industries. In the future they may become common in most industries to bypass the laborintensive steps of mailing, sorting, and key-entering orders, invoices, and other documents which pass from a computer in one organization to a computer in another.

Industries with inter corporate computer networks today include banking and air- lines. Most major airlines have reservation systems in which terminals over a wide geographic area are connected to a central computer. Worldwide airlines have worldwide networks. Many booking requests cannot be fulfilled completely by the airline to which they were made. The airline might have no seats available, or the journey may necessitate flights on more than one carrier.

Booking messages therefore have to be passed from the computer in one airline to the computer in another, and often the response is passed back swiftly enough to inform the booking agent who initiated the request at his terminal. In order to achieve this linking of separate systems all partici- pating airlines must agree to a rigorously defined format for the messages passing between the airlines. This format is standardized by an industry association, ATA in the United States and IATA internationally. To operate the interlinking network, the air- lines set up independent nonprofit organizations.

ARINC (Aeronautical Radio Incorporated) in the U.S., and STTA internationally (Societe International de Telecommunications Aeronautique). The separate airlines must send ATA- or IATA-format messages using the ARINC or SITA protocols. These networks began as networks for sending low-speed off line telprinter messages. As the need arose they were upgraded to handle fast-response messages between computers as well as conventional teleprinter traffic.The computer-to-computer network of SITA (including future proposed links).

Networks have also been designed to connect bank computers for moving money .

The SITA network's present and proposed trunks. Many smaller, lower-level centers are connected to those shown.
and messages almost instantaneously between banks. As with the case of airlines, the bank computers are differently programmed, incompatible machines, set up by widely different corporations in different countries. Like the airlines the banks must send rigorously formatted messages and observe precise network protocols. In this case a very high level of security must be built into the cooperative procedures because sums exceeding a million dollars are transmitted between computers.

SYSTEMS UNDER ONE MANAGEMENT

Much of the use of distributed computing is within one corporation under one management. This could result in a compatible configuration using a common networking architecture. Often, however, the systems to be linked were installed separately in separate locations without any thought about eventual interconnection. The files or data bases are incompatible; the same data field is formatted differently in different systems; programs cannot be moved from one computer to another without rewriting; where teleprocessing is used the terminals are incompatible; and even the line control procedures are different so the terminals cannot be changed without a major upheaval in the systems they are connected to. In this environment a major reprogramming and redesign effort is needed before networking becomes of much value, and often this effort is .too expensive.

It is necessary that systems in different functional areas of a corporation be developed by different groups. Corporate data processing is much too complex for one group to develop more than a portion of it. The current trend to decentralization is resulting in more and more autonomous groups carrying out application development. This a valuable trend because it results in more people being involved in application development, and the development being done locally where the application problems are understood.

INTERFACES

In order to make computer networking of value, it is desirable that the interfaces between the separately developed systems be rigorously defined and adhered to. If the interfaces are preserved, each development group can work autonomously.

There are several levels of interface:

1. Interface to the transport subsystem which permits blocks of data to be moved between distant machines. This interface can be defined independently of the application or the firms which use the network.

2. Interfaces for the software services which are external to the transport subsystem but not part of the application programs; for example software for remote file access, compaction, con- version, cryptography, setting up sessions, editing messages, and so on.

3. Applications interfaces defining what transaction types are interchanged between different application systems. These can be defined independently of the choice of networking software or hardware.

Interface 1, above, is provided by some common carrier systems for computer networking (the CCITT X.25 standard, for example). Interfaces 1 and 2 are provided by some of the manufacturers protocols for computer networks and distributed processing (for example IBM's and DEC's architectures for networks). Interface 3, above, is usually up to the systems analysts.Gives an illustration of computers serving six functional areas in a corporation, and shows the transaction types flowing between them. A typical transaction would be given a rigorously defined format. When they are transmitted between machines, data would be in the format with additional headers and a trailer prescribed by interfaces 1 and 2.

As changing costs take the computer industry increasingly toward distributed processing, one highly desirable characteristic is portability of programs. Programs should be capable of being moved from one processor to another and gaining access to distributed data instead of centralized data. There are arguments for, and against, distributed processing, and there are many possible distributed configurations. It is advantageous for a manufacturer's product lines to possess the flexibility to change system configurations without the need to rewrite programs.

The interfaces and protocols that are desirable for distributed processing make the software complex, as we shall see. Furthermore there are so many different configurations, functions, machines, operating systems, access methods and data base management systems that need to be supported that it will be years before the software for distributed systems can do everything that is theoretically desirable. New machines, operating systems, and other software will increasingly be designed to plug into the rigorously defined architectures for distributed systems.

Computer networks and distributed processing are a vitally important and fundamental step in the growth of the computing and telecommunications industries. There is a long road ahead, and the journey will take years to come.

Applications Areas

INTRODUCTION

1. Computers can be employed for a wide variety of purposes but computers are particularly suited to certain kinds of work. It may be possible to use a computer for a particular application if certain criteria are met. Whether or not a computer is used will depend on other factors. This chapter considers these criteria and factors and then deals with a variety of particular applications which illustrate the general principles.

CRITERIA FOR USING COMPUTERS

2.The following are the criteria by which to judge an application's possible suitability to the use of computers:

a.Volume. The computer is particularly suited to handling large amounts of data.

b.Accuracy. The need for a high degree of accuracy is satisifed by the computer and its consistency can be relied upon.

c.Repetitiveness. Processing cycles that repeat themselves over and over again are ideally suited to computers. Once programmed the computer happily goes on and on automatically performing as many cycles as required.

d.Complexity. The computer can perform the most complex calculations. As long as the application can be programmed then the computer can provide the answers required.

e.Speed. Computers work at phenomenal speeds. This combined with their ability to communicate with other systems, even those at remote locations, enables them to respond very quickly to given situations.

f.Common data. One item of data on a computer system may be involved in several different procedures, or accessed by a variety of users.

It can be updated and inspected by a number of different users. In manual systems data is often accessible to a limited number of people for particular purposes. This can hinder the work of others who need access to the data.

3.It is usually the combination of two or more of the criteria listed which will indicate the suitability of an application to computer use. The criteria that have been described will be used by those who carry out a Preliminary Survey in order to judge the suitability of applications for computerisation.

OTHER FACTORS

4.If the general criteria for using a computer suggest that a particular application may be suitable for computerisation, then there are a number of questions which will require satisfactory answers before any decision to computerise is taken. The main questions will be;-

a. Is the use of a computer for this application technically feasible? ie. can it be done with the computer technology currently available?
b. Would the use of a computer be cost effective? ie. would the computer pay for itself in terms of the benefits it would provide?
c. Would the use of a computer be socially acceptable? ie. would the impact of the computer on people's work, jobs or general lifestyle be acceptable?

5.The answers to questions such as those just mentioned, change with changing circumstances. For example many computer applications which were mere science fiction a few years ago, are now technically feasible. eg. the use of simple robots. Developments in microelectronics have reduced prices so that applications which have been technically feasible for twenty years or more, are only now becoming cost effective. Peoples willingness to accept computers depends on previous experience, general attitudes, and on how well or badly they have been informed.

LEVELS OF COMPUTERISATION

6.The extent to which an application may be computerised will be determined by the nature of the work involved. Three basic levels of computerisation may be identified: basic levels of computerisation may be identified

a. **Complete computerisation.** Simple well defined and repetitive tasks can often be completely computerised eg. basic clerical functions or control of simple machines.

b. **Partial computerisation.** Computer is can often be applied to applications which require the control of operations under some agreed plan or strategy. The computer may take oven routine control bat may be monitored by humans, who will also deal with exceptional cases eg. the day to day operation of a stock control system or a computerised production line.

c. Computer aided applications. Computers may be used in many applications to aid management or decision making, by the Sorely provision of accurate results or information eg. the computer can be used to analyse problems or simulate systems in order to aid designing or planning.

MAIN **AREAS OF APPLICATION**

7. **Two** main areas of computer application may be identified

a. **Commercial applications. This covers the use of** computers for clerical, administrative and business uses, in private and public organisations, ie. the emphasis is on data pr000ssiog. (in. Collecting, maintaining and manipulating volumes of data to produce information).

b. Scientific, Engineering and Research Applications. This covers the use of computers for complex calculations, the design, analysis and control of physical systems, and the analysis of experimental data or results, in. the emphasis on Scientific Processing (in. the rapid processing of data relating to complex problems.)

There are other to minor areas which do not fall into either or the two main categories eg. Personal Computing in computing done as a hobby. One could argue that it fulls into wiser category.

8. Many organisations use computers for a variety of applications. For example a manufacturer may use computers for data processing, scientific research and engineering development work.

COMMON APPLICATIONS

9.**Payroll.** This is a well established computer application normally handled by batch processing. The production of the weekly wages or monthly salary payments of employees is a regular repetitive clerical task on sizeable volumes of data and ideal for computerisation.

10.**Office Automation.** In contrast with payroll which is a long standing computer application, office automation is a relatively new area of computerisation.

In automated offices many of the routine clerical and secretarial tasks are taken over by computer based equipment which exploits developments in microelectronics.

11. Elements of such systems include:
a. Modern computer systems.
b. Word processing systems (ie. Computers used for document preparation).
c. Modern methods of displaying and copying data electronically
d. Modern communication links able to interconnect all elements in the system to one another and to other systems eg. by networks or electronic mail (details later).

12.Stock Control. This application will be discussed . The control of stock is important in both public and private orgnaisation.

13.Production and labour control. The success of an organisation depends on how well it manages its resources. People, machines, materials, money and buildings all need careful management. Computers are used to control production and labour, just as they are used to control stock.

14. Accounting. There are many routine clerical tasks associated with recording details of financial transactions made by an organisation. This has given rise to the frequent use of computer for such accounting functions, particularly in larger organisations

EXTENDING AND INTEGRATING APPLICATIONS

15. Many basic applications can be extended to give useful information for management purposes eg. using a stock control system to provide reports to management. Further benefits can be obtained by integrating different 16. applications eg. linking the payroll system to the labour or production control system.

16. Further benefits can be obtained by integrating different applications eg. linking the payroll system to the labour or production control system.

PARTICULAR APPLICATIONS AREAS

17. In this section a number of particular applications areas are described because of their importance and interest.

18. Applications exploiting the full computational power of computers. Many of these applications have a scientific bias. They include:-

a. **Weather forecasting systems.** Reliable weather forecasting a demands vast computational powers. This is an area for the super computers (ie. computers with exceptionally fast processors).
b. **Mathematical and Statistical Analysis.** This includes large calculations and the solution of mathematical problems. The applications requiring this include research in physics, chemistry, geology, archaeology, medicine astronomy etc. Some commercial problems also have a mathematical bias eg. those ,that require mathematical analysis to determine the optimum use of resources.
c. **Design work.** Computers can be exploited as design tool in engineering and other disciplines. CAD (Computer Aided Design) is growing in importance in

Electronic, Electrical, Mechanical and Aeronautical Engineering and Architecture.This application often also exploits in computer graphics.

19.Analog Computing. Most computers in everyday use today are digital computers. That is, they are computers which carry out operations on distinct data values in discrete steps. Analog computers, in contrast to digital computers carry out operations on data which can vary continuously.

20.Financial Applications. The banks and insurance companies are major users of commercial computer systems. Here is an indication of some of the ways in which banks use computers:

a. **Automatic cheque clearing.** Since the late 1950s banks in Britain have used a computerised system for handling cheques which ensures that payments by cheque are cleared within three days.

b. **Standing Orders and Direct Debit.** Regular payments may be made automatically by banks as part of a computerised system called BACS (Bankers' Automatic Clearing Services Ltd). Magnetic tape is used to store details of the transactions for a particular day.

c. **General uses.** Bank customer accounts are largely computerised and some details, eg. current balance, may be available on-line.

d. **Newer uses.** Within the last ten years or so a number of special purpose cash dispensing machines have been introduced for use outside banks even when the banks are not open. This is a computerised service.

21.The cashless society. The success of computers in banking and in supporting credit card systems such as BARCLAYCARD or ACCESS has led many people to predict that we will eventually have a "cashless society" in which credit cards and special tills will cater for all "money" transactions. At present over 90% of all payments are in cash.

22.Retailing. The use of computers in the retail trades is now widespread. There have been numerous developments particularly in the area of data capture.

23. Some large supermarket chains have large and sophisticated stock control systems in which tills, using laser scanners, provide on-line data capture, and also have warehouses which are fully computerised.

24. Medical Applications. There are numerous. applications of computers in medicine. Here are some exam research

a. Computers can be used as an aid to medical research the by analysing data produced from experiments e trial of drugs.

b. Computers can be used to aid diagnosis. The computer acts as a large bank of data about known medical conditions. Once the computer system has been set up by medical experts an ordinary doctor can be taken through a question and answer session by the computer until a correct diagnosis is made.

c. Computers can be used to hold details of dentists' or GPs' patients. Small computer systems have been used for years in increasing numbers since the late 70 s.

d. Computerised children's health records for immunisation have been used by local health authorities for a number of . These records are used by medical officers, health years visitors etc.

25. Education. Computers of specialist study in Computer Studies, they are used as an extremely versatile way of aiding the understanding of a user wide variety of other subjects. The computer can guide a course of instruction at a VDU. The computer can through provide instructions and ask questions of the user CAI of activity is called CAL (Computer Aided Learn g) (Computer Aided Instruction).

26. Computers are also used for a number of other applications the marking of multiple choice examination in Education eg examination results for many papers and processing examinations boards.

27. Manufacturing. Some aspects of computer use in manufacturing stock a ring facturing have already been covered egineering design.The design control, and engineering computerised, and testing processes are design, hence the terms CAD and CADMAT Computer (Computer Aided Manufacture Aided Design Manufacture and Testing).

In some countries, particularly in Scandinavia, there are public mittees charged with the task of approving the design of systems that are to contain personal data. To make their task easier, design guidelines are being developed and these will give a passive users some influence , although quite indirect on the design of systems in which they may be included.

APPLICATION AREAS

The use of CBISs has spread from the research applications in technical d natural sciences , government statistical services and military systems for which they ere initially used in the mid-1950s to ' t ally all research areas, government and business administration, health services the study and development of hence and music scores, teaching from pre-school to post-graduate, customer services computing, and other areas. Today it is difficult to find an area which has not been affected to some degree by automation. And indeed, new application areas s em limited only by our imagination.

Perhaps not all applications are good in the sense that automation has improved some facet of the work or leisure activity, or has have been a economically justifiable. However as the costs f computer power continue to fall and the aveila6le computer power and memory space continue to increase, the economics of applications will continue to improve .

In the following discussion we will list a number of traditional application areas and note a few of the more unusual application . The idea is to stimulate the reader's imagination not to list good application

Business Applications

The best-established business CBISs are at the operational level of the organization . These s est ms capture operating data product customer , suppliers, and employees. They then produce r port inventory status, customer billing, material orders, payrolls, budgets, and accounts. Traditional CBIS applications are found in:

- production management: production control, engineering support (CAD/CAM)project management, inventory management internal accounting.
- marketing: market research, sales support, customer service supplier systems.
- personnel management: payroll, project assignment , personnel development (education, job histories, recruitment).

USING COMPUTER-BASED INFORMATION SYSTEMS

- planning: decision support systems, operational analysis, project scheduling, forecasting, simulations.
- finance: budget planning, portfolio management, accounts receivable or payable, general ledger.
- office management: word processing for reports and letters, scheduling of meetings, message/memo systems, archives, information retrieval systems.

Within business applications, current CBIS development efforts are directed toward providing office management and office activities with automated tools and toward making the organization's data and data processing facilities more readily available to both tactical- and strategic- level management. Under the name of decision support systems (DSS), management decision aids are being provided for use in forecasting, statistical analysis, econometric analysis, and simulation of decision alternatives.

Improvements in terminal facilities and user interface languages are allowing non-EDP personnel and infrequent users to use DP services. The advent of microcomputers has allowed data processing services to move into the office and into small businesses, such as medical offices, dental offices, or local groceries.

Government and Public Service Applications

Many of the types of CBISs applicable in business administration are also applicable in government administration at all levels, from national to local, and in all sections, including health and social services, communications, justice, military, and others. Examples include personnel administration, finance, customer/client services, and project management.

In addition, specific government and public service operations use a number of CBIS applications. These include:

- law enforcement: criminal and crime registers, court scheduling, law research.
- health services: patient journals, hospital service scheduling; intensive care monitoring, outpatient registers, billing and payment, national health services (Medicare/Medicaid), medical research.
- social services: participant registration (such as Social Security membership), recipient payments, registration of social services and pro- grams, school administration, Social Security administration.
- internal revenue services: personal and business tax management, budgeting. . statistical services: census, labour statistics, business, export and import.

APPLICATION AREAS

- military applications: research, space exploration, strategic planning.
- weather forecasting.
- library and information services: recording and making available information about books, articles, and laws; medical information; consumer reports; statistical reports.
- graphic services: constructing maps of states, counties, or cities; presentation of data in graphic form; locating special items, such as fire hydrants, candy shops, or hospitals. Also, in government and public service areas, as well as in the private sector, organizations are developing CBISs aimed at providing information-processing services to high-level management, office staff, and the general public.

Particularly interesting is the development of public information services where "customers," that is, anyone with a terminal, telephone, or TV, will be able to reach a news/information service and receive information covering any of a number of areas, from weather to sports to foreign and local news. Customers will also be able to order anything from airline tickets to food.

Research Applications

Almost all possible areas of research have made or can make use of CBISs. For example:

- Statistical methods are useful for determining relationships between observed phenomena in areas such as weather, stars, public opinions, epidemics, word usage in texts, and many others.
- Computer simulations are useful for analyzing models of functions of people (medicine), animals (biology), organizations and technical systems, and many others.
- Information retrieval techniques are useful for organizing, and selecting individual items from; voluminous quantities of texts, and from reference collections (catalogs) for manuscripts, music scores, descriptions of pictures, museum objects, specialist directories, and so on.
- Graphic techniques can be used for studying almost all physical items from molecules and cells, towns and populations, to the solar system.

The special area of artificial intelligence uses the computer heavily to mimic human behavior in order to better understand how we function, the point being to be able to construct useful robotic helpers. Also, the computer and CBISs are vital to the further development of computers and CBISs.

Educational Applications

There are two important CBIS application areas in the educational sector, one in school administration and the other in support of teaching activities. Both these areas have made use of CBISs and are continuing to expand their use. A primary impetus to the use of automated tools is the advent of microcomputers, providing inexpensive computational power.

School administration, for one school or for a nation's or district's school system; is similar in many ways to business and government ad- ministration, including financial and personnel management; student (= customer/client) administration, and course (= product/service) administration. An interesting special application is the problem of scheduling students, teachers, courses, rooms, and time.

Computer-assisted instruction (CAI) programs for drills in spelling and math; simulated experiments in chemistry and physics; language development in English, Spanish, arid German; and many other areas are available. CAI programs are also available for a host of teach-your- self courses from programming and typing to history. CAI systems have been developed for the traditional school system, grades 1 through 12, for pre-school learning, for adult education, and for special job training. An example of the last would be the flight simulators used to train pilots.

Home Applications

With the advent of microcomputers, computing has become financially accessible to many homes and families. There are literally thousands of programs available for the microcomputer owner and his or her spouse and children. These include business systems to support home economics, such as check managers, word processors for letter writing, information retrieval systems for recipe collections, address list pro- grams for Christmas card lists, CAI programs for homework and new learning, and, of course, games.

The home market is becoming increasingly interesting to the information-service industry, which is offering home users such services as news and weather reports, stock market services, home banking, mail order services, airline reservations, and access to extensive data bases.

Computers are already used in our cars and stoves. They can also be used to regulate lighting and heating, and as burglar alarms.

Introduction to Distributed Systems and Distributed Software

We describe the main characteristics of distributed systems, their classification end programming techniques. Examples demonstrate the application areas of distributed systems.

Changes in Computing Technology

During the last two decades the principal application of the information processing and computing technology has been to central computers. Up to several thousand terminals have been connected to such centralised systems. Users sitting in front of their terminals have shared the central processor, the attached equipment for data storage, and the available programs. Each user is granted time slices of the central processor, so that the computer is shared between the various users. The execution of the users' programs is interleaved with several users sharing the execution time of the central processor. This type of system is called a time sharing system; it can be regarded as the parallel or concurrent execution of several programs i.e. their execution on several processors. If the same program is executed by several users each execution is in a different state, i.e. each user has his own set of data and the program instruction to be executed next is specific to the user. The execution of a program for a particular user is called a process (A more detailed discussion of the term process is given in section 1.3). Each process is given individual time slices of the central processor.

Several processes can have access to the same data base concurrently. This can lead to a situation in which the database is in an inconsistent state. This happens if for example one process wants to write to the database but access to the central processor is given to another process so that only parts of the new data have been written to the data base. As such other processes can have altered the data base before the first process can again access the central processor. To keep the data in a consistent state, access to shared data must be synchronised. In order to do this several synchronisation concepts have been developed. Synchronisation allows the execution of processes to be controlled, e.g. a process is stopped until another process has reached a certain state. Most programs have involved access to data bases. The major task of data base applications has been the retrieval. processing updating and replacing of data in data in. The programs process one or more data bases. The aim has been to develop a data model which covers all data aspects of an organisation in order to avoid that the same data is contained in more than one data bases.

This has been done by designing a single closed data base or by merging several existing data bases. The application programs try to; meet the requirements of organisational units but they cannot cover the requirements of small groups or individuals. The computer has to be used for the most important tasks of as many users as possible in order to recoup its high cost. There has been no room for individual wishes and requirements. Before starting the development of a new program or changing an existing program the requirements of all the users have to be ascertained. This is a complicated and cumbersome process. The development or alteration of a program has taken a long time, especially if new data which is not available in the existing data bases or was not even foreseen in the data model has been required.

Since the eightes a new trend emerged, personal computers or PC's. These allow the environment of each user to be individually configured. All the computing capability, storage capacity, data, and programs belong to the owner of the PC. This has made it possible to implement programs which meet the reqirements of a single user much better. The exclusive ownership of computer capability has allowed the development of very convenient user interfaces and individually tailored solutions. This has been supported by window oriented operating systems, text processing systems, table calculation programs, graphics programs, etc. PC's have provided a very individual and convenient form of computer access. In order to provide access to data which is shared by several users PC's have been connected to host computers. PC's, workstations (which can be considered as more powerful PC's) and host computers have been combined in client server systems. The PC's are considered as clients which can use the services provided by specialised servers. The main part of the application program runs on the client but some parts run on specialised servers. For example a data base application together, with a convenient user interface runs on a PC separate and apart from the database system which runs on a server. Via special communication services, clients can access the facilities provided by the servers. Another technique has been to connect computers with each other directly. These loosely coupled computer systems can exchange data directly instead of sharing data stored on a connected server. Instead of shared data, messages are used. These different types of computer networks support different types of applications and programs.

Characteristics of Distributed Systems

A precise and general definition of a distributed system is not very easy and as far as we know, a definition does not exist which has been generally accepted. Therefore we do what nearly all authors who have written books about different aspects of distributed systems do: we try to give the reader a good understanding of the nature of distributed systems by describing their major characteristics. Experience has shown that this is an adequate substitute for a definition.

Distributed computer environments are based on distributed computer systems which consist of a set of processing components connected by a communication network. The software systems running on the various processing components ex- change data through the communication network. This type of system is also called loosely coupled distributed system. Processing nodes can be composed of several processors which share memory. This shared memory is used to exchange information by the software executed on such a node. This type of system is called a tightly coupled distributed system. The advantages of distributed systems are outlined in nearly all books and papers related to the topic e.g. /SHWA89/, /C0D088/. Below we mention the most important ones /SHWA89/:

- Increased Performance

Performance is generally defined in terms of average response time and through- put. If processing capability can be located where it is required the response time can be highly reduced. Data can be processed locally before it is sent to other nodes for further processing. This increases throughput.

- Increased reliability

Normally nodes in a distributed system can take over the tasks of other nodes which are currently out of order. This means that a distributed system continues its work with reduced performance but with little or no reduction of functionality

- Increased flexibility

Additional functionality can be added to a distributed system or the number of users can be permanently increased. A distributed system allows this system growth by simply adding more processing nodes.

Parallel or Concurrent Programs

Before we consider the characteristics of distributed software in more detail, we have to consider the concepts of parallel processes and programs. Parallel or concurrent programs are characterised by a set of statements inter- related by multiple control threads. Each sequence of statements executed by one or more control threads is called a process object /NEHM8Ba/ /Z0H088/ (The term `process' shall be used instead of `process object' when it is clear from the context that we mean a process object). The relationship between processes or threads and process objects is shown in the following figure.

Concurrent or parallel programs are either interleaved, distributed, or both. For a programmer it is not necessary to know whether multitasking or a distributed system is used to run his program.

Normally the processes of a concurrent program share the resources such as processor, memory, disk, and databases, and if they cooperate in order to reach a common goal they exchange information and synchronise their activities.

Their are two reasons to structure a program in parallel executable process objects:

1. Fine grain parallelism is mainly used to accelerate large numerical computations. This type of parallelism is often achieved by using vector processors and the pipelining of operations. It is mainly implemented by hardware.
2. Structural parallelism is used if the structure of the task to be performed is fundamentally parallel. The process objects are a very important concept for structuring programs in certain application areas, e.g. operating systems, real time systems, and communication systems. Especially in real time systems which must react to external events, processes (objects) are used to achieve separation of the tasks /FAPA88/. Each process handles a related set of events and cooperates with other processes to achieve a common purpose. In order to cooperate, processes exchange information either via shared data or via messages.

When considering the software running on a distributed system we can distinguish between networked computing systems and cooperative computing systems /SHWA89/. In the following sections the major aspects and applications of these two distributed software types are discussed.

NETWORKED COMPUTING

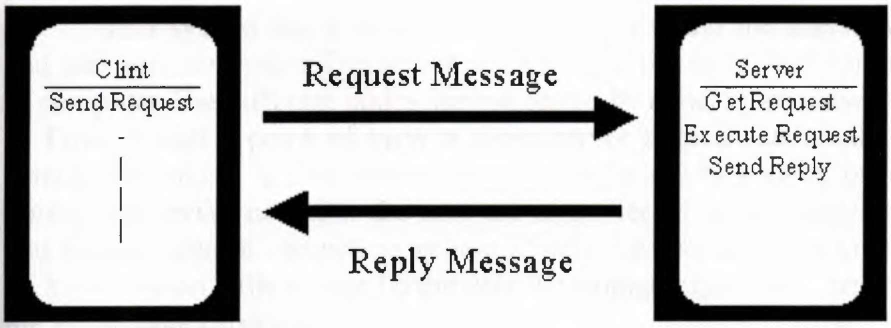
Network Structure and the Remote Procedure Call Concept

Networked computing is characterized by several sequences of jobs which arrive independently at various nodes. The jobs are designed and implemented more or less independently of each other and are only loosely coupled. The distributed sys- tem serves primarily as a resource sharing network.

A very common example of resource sharing is the file server. All files are located on a dedicated node in a distributed system. Software components running on other nodes send their file access requests to the file server software. The file server executes these requests and returns the results (to the clients).

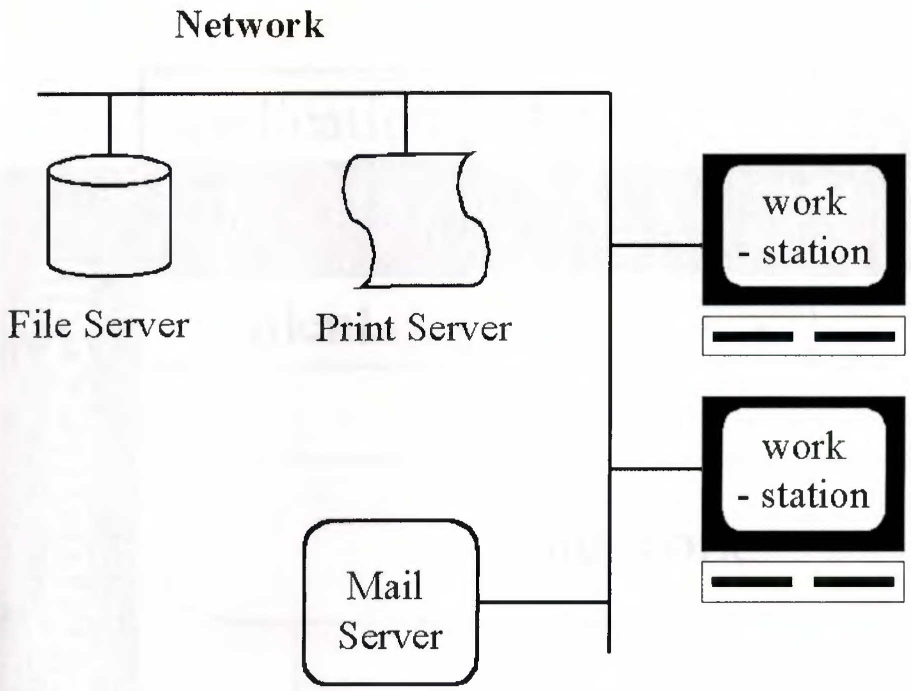
In addition to file servers many other kinds of servers such as print servers, compute servers, data base servers, and mail servers have been implemented As with the file server, clients send their requests to the appropriate server and receive the results for further processing. Servers process the requests from the various clients more or less independently of each other. The programs running on the clients can be viewed as being designed and developed independently of each other.

The following figure shows the concept of client server systems.



In client server system, the clients represent the users of a distributed system and servers represent different operating system functions or a commonly used application.

The following figure shows a simple example of a client server system.

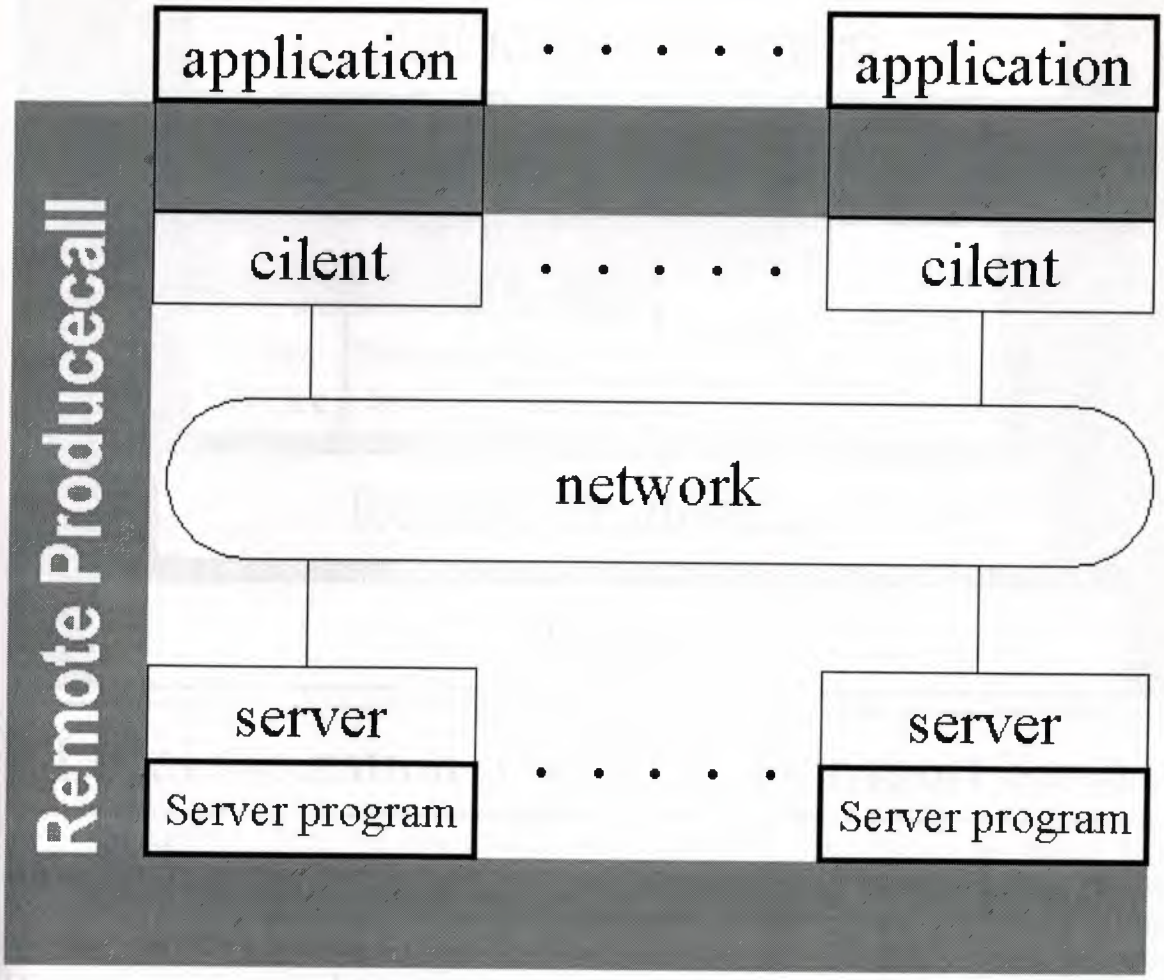


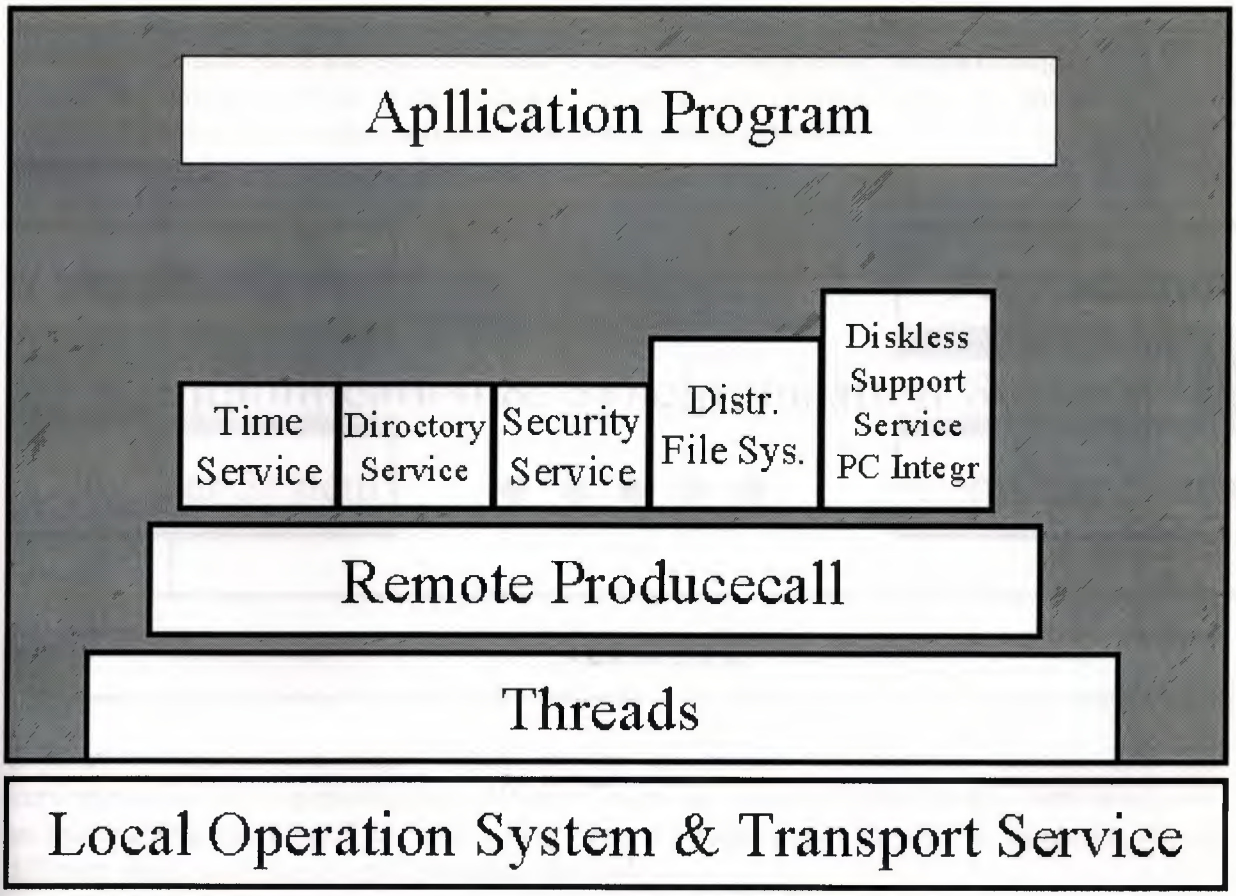
This system has a print server,a file server and the users which run on workstations and personel computers.The server software and the client software can run on the same type of computer.The different nodes are connected by a local area network.

From a user's point of view a client/server system can hardly be distinguished from a central system.e.g. a user cannot see whether a file is located on his local system or on a remote file server node.For the user the client/server system appears to be a very convenient and flexible central computing system.Mostly the user does not know whether a file is stored on his PC or on a file server.To the user the storage capacity of the server appears to be part of the PC storage capacity.

Client/server systems are also very flexible. For a new application a specialised new server can be added e.g. data base systems run on specialised data base servers which have short access times. Data base applications are primarily controlled by the local client; all the data is stored at the data base server and special computations are executed by a compute server (also called number cruncher). The application program running on the client, calls the required functions provided by the servers. This is done mainly by way of remote procedure calls (RPC). An RPC resembles a procedure call except that it is used in distributed systems. The following is a description of how the RPC works. The program running on the client looks like a normal sequential program. The services of a particular server are invoked via a remote procedure call. The caller of a remote procedure is stopped until the invoked remote procedure is finished and the server has provided the results to the calling client in the same way that parameters are returned by a procedure. The servers are used in the same way that library procedures are used. This means that remote procedure calls hide the distribution of the functions of the system even at the program level. The programmer does not need to concern himself with the system distribution.

The figure below shows the basic structure of a client/server system.


Remote Producecall
application
application
cilent
cilent
network
server
server
Server program
Server program



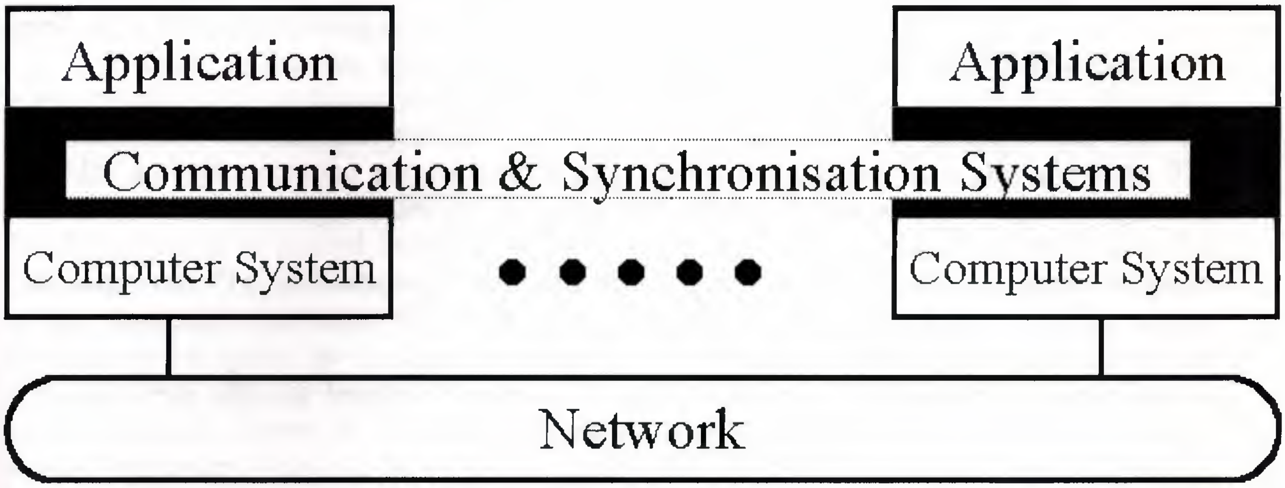
In the DCE client and server programs are executed by threads i.e. processes. Threads use an RPC in order to communicate with each other and binary semaphores and conditional variables for synchronisation. In the DCE remote procedure calls are supported by directory services (DCE Call Directory Service) and security services (DCE Security Service). Directory services map logical names to physical addresses. If a client calls a particular service provided by a server, the directory service is used to find the appropriate server. The DCE security service provides features for secure communication and controlled access to resources. Distribute Time Service provides precise clock synchronisation in a distributed system. This is required for event logging, error recovery, etc. The distributed file service allows the sharing of files across the whole system. Finally the diskless support service allows workstations to use background disk files on file servers as if they were local disks /SCHILL93/, /0SF92/.

Cooperative Computing

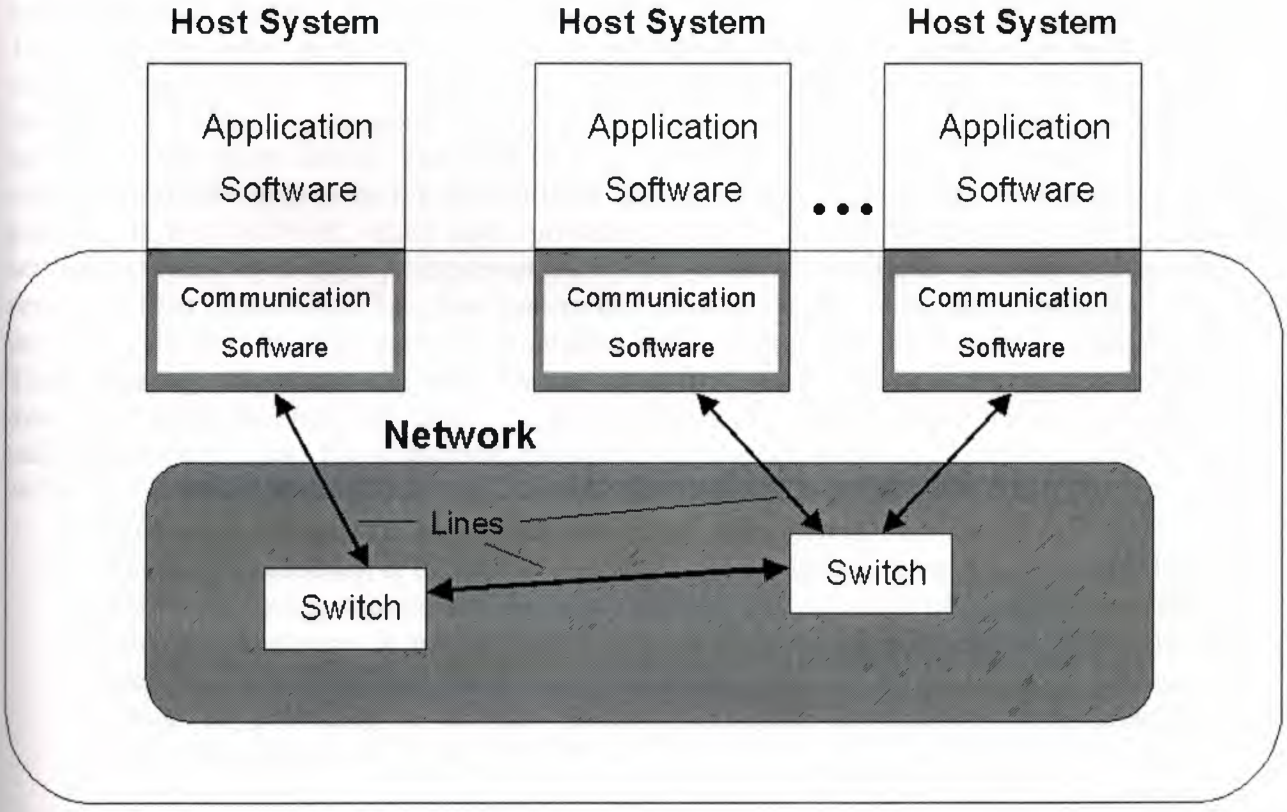
In cooperative computing a set of processes runs on several processing nodes. These processes cooperate to reach a common goal and together they form a distributed program. This is different from the client/server systems described above. In cooperative systems the processes which comprise the distributed program are coupled very closely. This means that the closely coupled processes are executed on a loosely coupled system.

In cooperative systems, the distribution of computing capability is not hidden behind programming concepts. The different program sections running on different computers comprise a single program; but it can be seen at the programming level that the program sections are executed concurrently. These different program sections are also processes. Processes form a very important concept for central systems, client server systems and cooperative systems. If processes have to work together to perform their task, they must exchange data and synchronise their execution. Programming systems for concurrent systems contain communication and synchronisation concepts. Cooperative programming resembles a human organisation which works together to achieve a common goal. Its members must communicate with each other and must synchronise their activities.

The following figure shows the basic structure of cooperative systems.



Coperative systems are mainly used for the automation of technical process and the implementation of communication software.Technical process in the mostly part consist of several parallel activities.This means that several processes which can be implemented in different ways work together to perform their task.



Communication System

Communication Software Systems

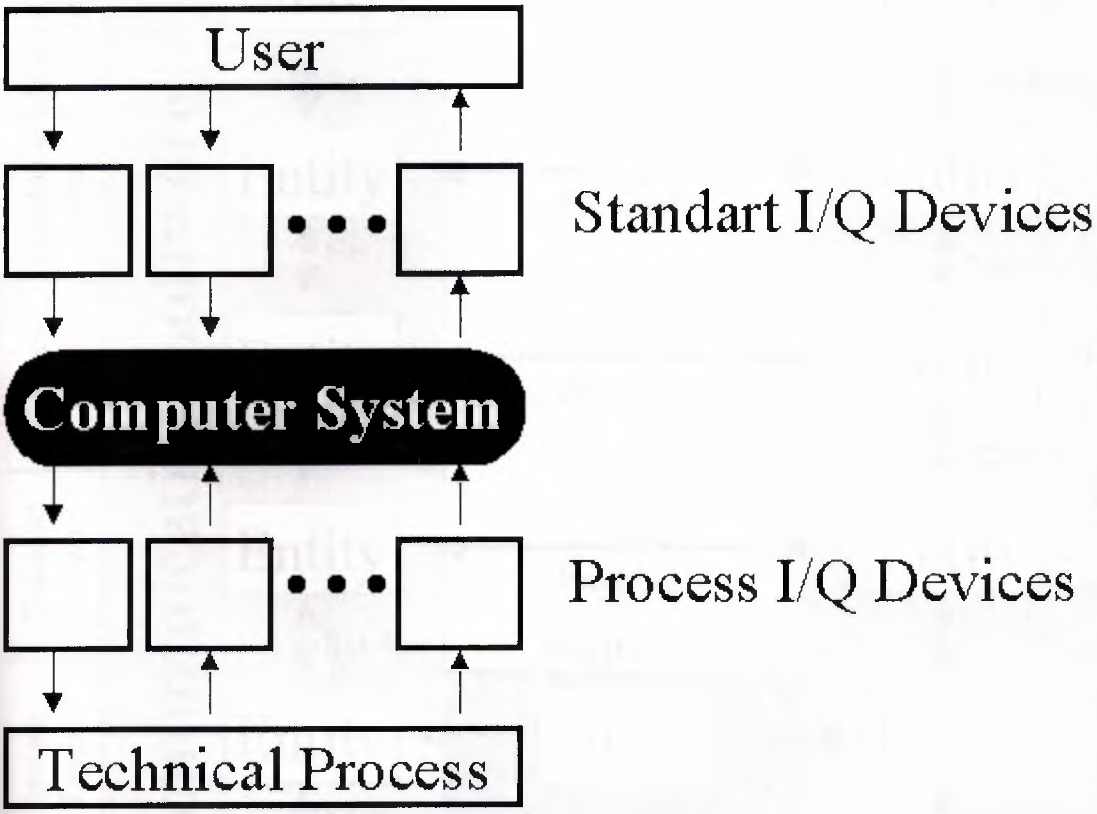
A communication system consist of a communication network and the communication software which runs on the various processing nodes.The communication software provides a more ar less convenient communication service for the application sotware.The application software on each node uses the communication service to excahange messages with the application software running on other nodes.

In order to provide a convenient communication service the commuýýication software systems also exchange messages. This message exchange is based on the sim- pler communication mechanism provided directly by the network. For example the network provides a communication service which only allows the transfer of a sin- gle byte. The communication service provided by the communication software al- lows byte strings of a fixed or even an unlimited length to be sent or received. This can be implemented in the followmg way: The application software of a host system A wants to send a sequence of bytes to the application software of a host system B. The sequence of bytes is given to the communication system by the application system. The communication system on host system A sends a byte with the length of the byte string (the number of bytes) to the communication system on host system B. The communication system on host system B sends back an acknowledgement. This is a byte with a certain value. After the communication software on host system A has received the acknowledgement it starts to transfer the bytes of the byte string. When system B has received the number of bytes indicated in the first byte it again sends an acknowledgement. After sending the acknowledgement, the communication software on host system B gives the received byte string to the application software. This communication sequence which implements the transfer of a byte string is just a simplistic illustration of what communication software can do. As the example above shows, the communication between the communication software systems follows well defined rules. These rules are called protocols. The need to provide convenient communication services for the application software leads to software communication protocols which can be extremely complex and must be organised in layers. Each layer offers an improved communication service to the layer above. The widely used reference model for Open Systems Inter- connection (OSI) defined by the International Standard Organisation (ISO) pro- poses seven protocol layers /IS07498/. Each layer provides a certain service to the layer above. The service provided by a layer is implemented by the protocol specific to its layer and by the services of the layer below. In a host system the services specific to the layer are realized by protocol entities. The layer protocol is defined between protocol entities of the same layer. These exchange information by using the service of the layer below. In each host system there must be at least one entity per layer. The set of entities of different layers in a host system is called a protocol stack. The implementation of these protocol stacks is called communication software. Communication software has the following execution properties /DROB 86/:

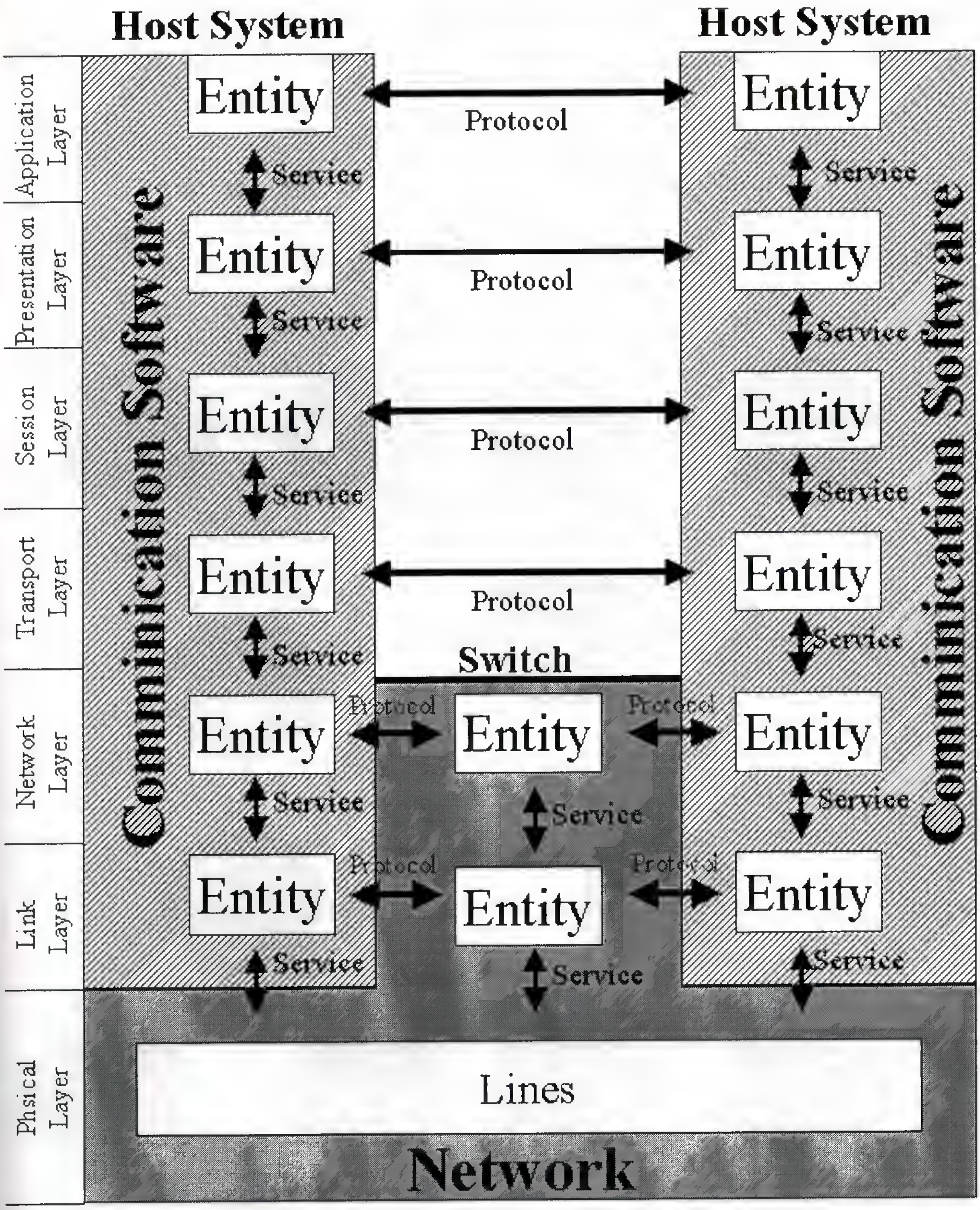
- interleaved execution of several entities on the same system
- distributed execution of entities of the same layer on different systems. Interleaved and distributed computations are usually modeled as systems of parallel processes. Processes executing in parallel normally have to exchange information if they are to cooperate in solving a common task. Entities are modeled by one or more processes. Using or providing a service means exchanging information with processes representing entities of the layer below or above. The figure above shows

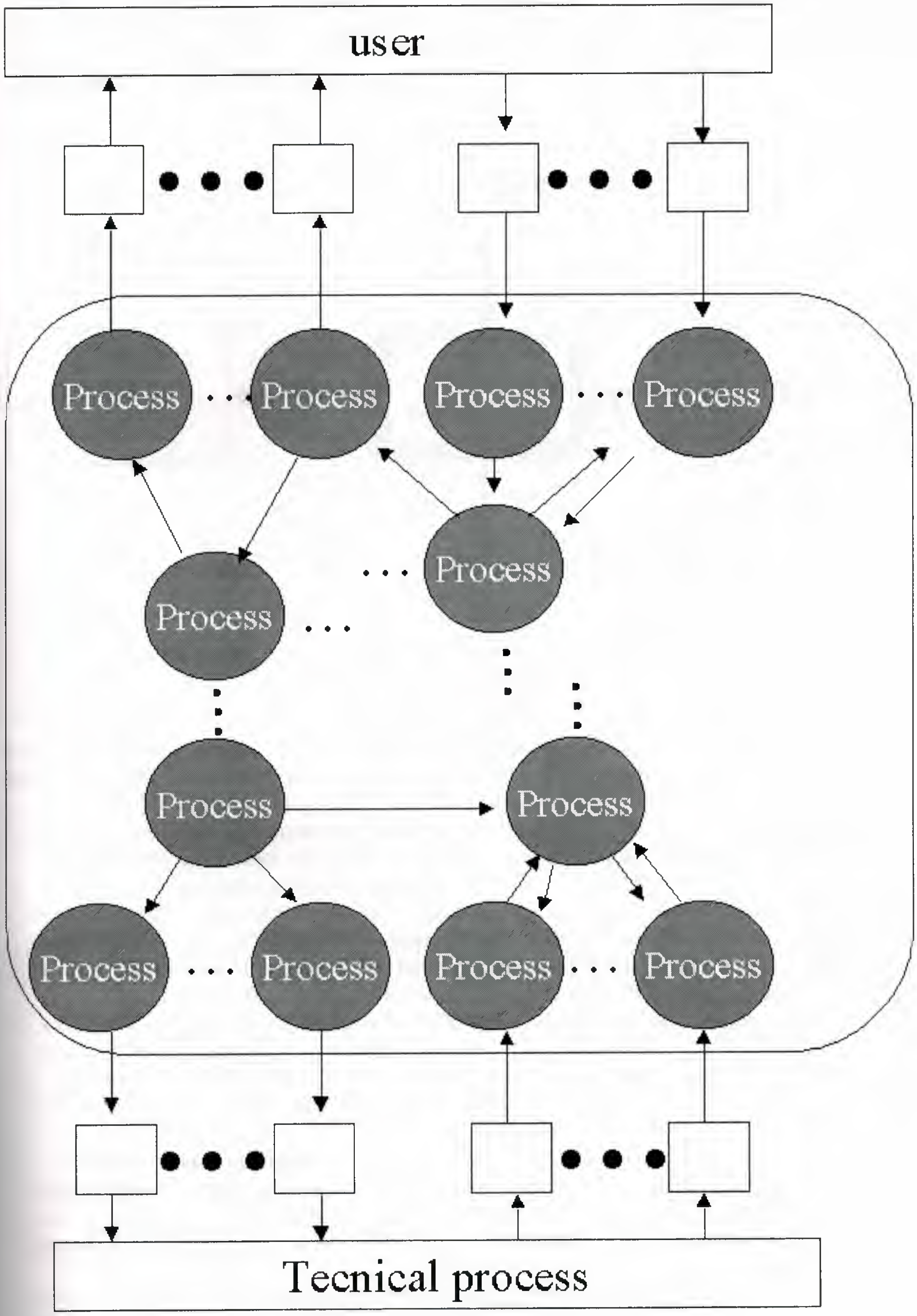
Technıcal Process Control Software Systems

Another important example of cooperative computing is a distributed technical process control system. The basıc structure of technıcal systems controlled systems is shown in the following figure /NEHM84/.



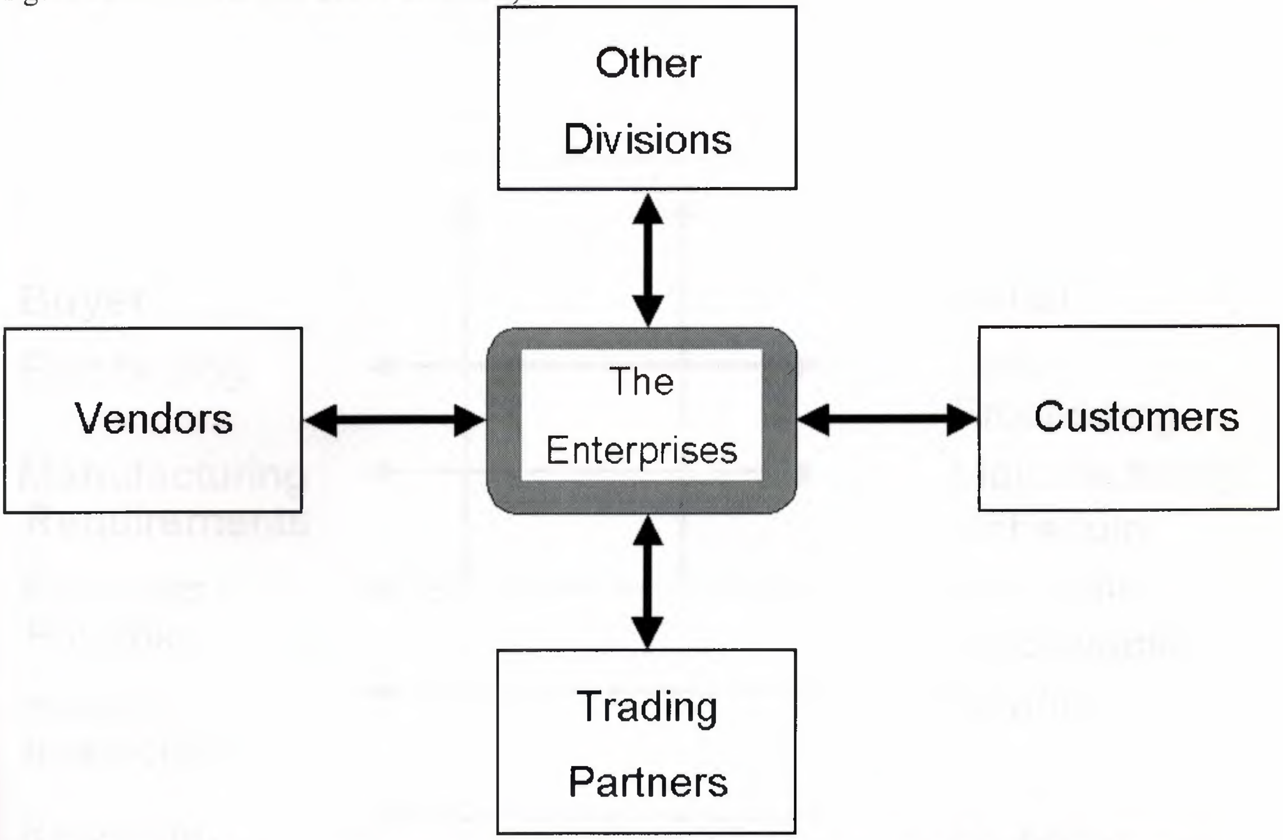
The communication between computer systems and technical systems must meet hard realtime requirements, whereas the communication with the user is more or less dialogue-oriented with less emphasis on time conditions (except in the case emergency signals such as fýre alarms). For the sake of simplicity, we will focus on the relationship between technical systems and real-time computer systems. A technical system consists of several mutually independent functional units which communicate via appropriate interfaces with the computer system. Therefore the real time program must react to several simultanous inputs. This implies the structuring of a process control software system that takes into account a number of processes. Each process handles a certain group of signals. The basic requirement for a process control software system is the capability to follow the changes of the technical system as fast as possible. The infomation in the process control software must be as close as possible to the state of the technical system. The easiest way to achieve this is to design a process for each interface element. This leads to the software system structure shown in the following figure /NEHM84/.


Host System
Host System
Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Link Layer
Phsical Layer
Communication Software
Communication Software
Entity
Service
Protocol
Switch
Lines
Network



1.5.3 Electronic Data Interchange (EDI)

Electronic Data Interchange (EDI) is the computer-to-computer exchange of inter- and intracompany technical and business data, based on the use of standards /DIGIT9O/ (see figure below of the EDI business model).



These data can be structured or unstructured. Exchanging unstructured data follows specific communication standards although the data content is not in a structured format. More important is the exchange of structured data. Examples of structured data exchange are:

- Trade Data Interchange

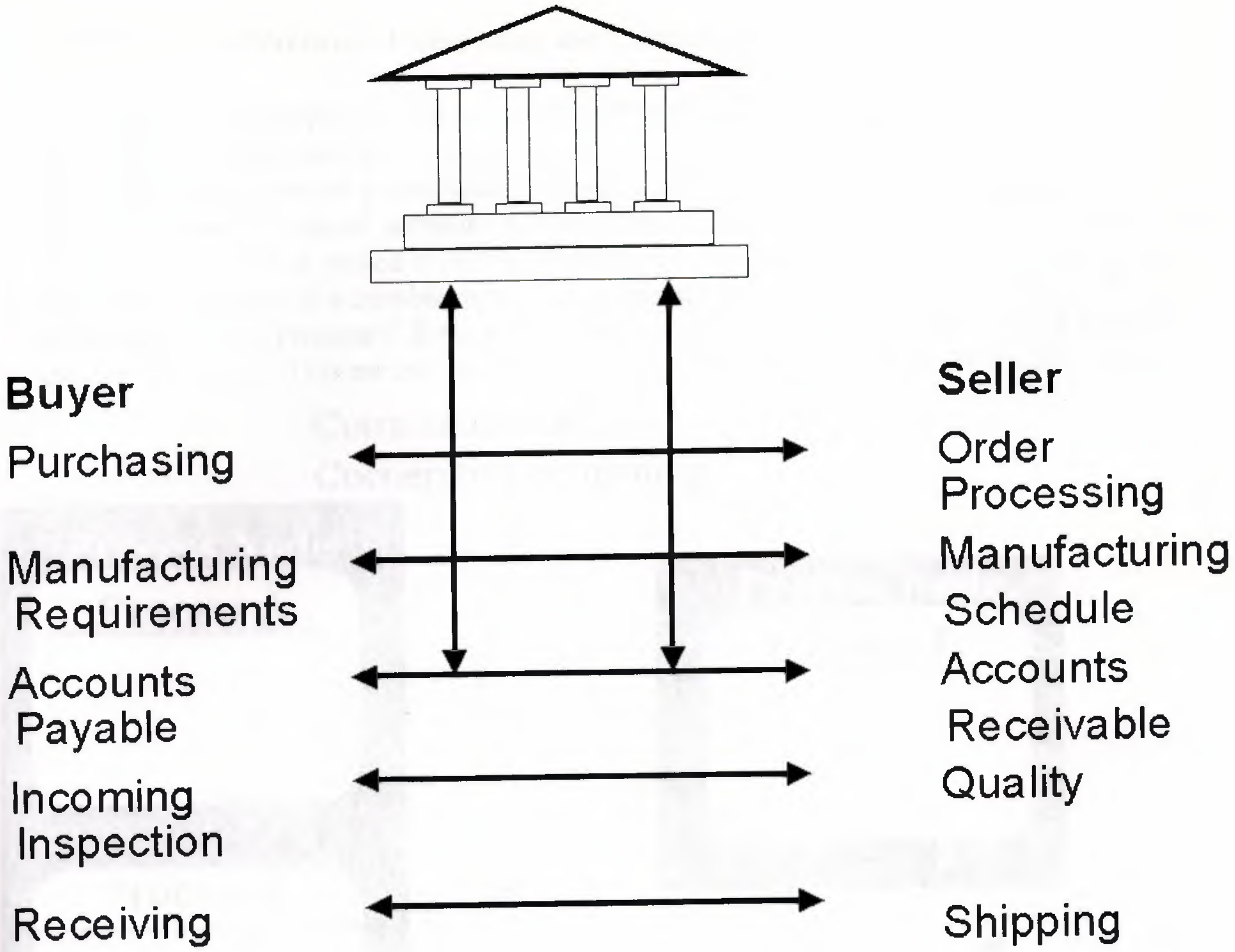
This type of EDI document exchange is mainly used to automate business processes. Examples of trade data interchanges include a request for quotation (RfQ), purchase orders, purchase order acknowledgements, etc. Each company and industry has its own requirements for the structure and contents of these documents. A number of specific industry and national bodies have been formed with the intention of standardising the format and content of messages. For the chemical industry CEFIC is the EDI standard and for the auto industry the related EDI standard is called ODETTE. The standard defined by CCITT is called EDIFACT. In order to exchange EDIFACT documents very often the CCITT E-Mail standard X.400 is recommended /liILL9O/.

- Electronic Funds Transfer Payment against invoices, electronic point of sale (EPOS) and clearing systems are examples of electronic funds transfer.

- Technical Data Interchange

Improvement in technical communication can play a key role in determining the success of a project. There is growing demand from trades for communication between their CAD (computer aided design) workstation and the workstations of important vendors.

The following example shows how the different types of EDI interactions are used to handle a business process.



Groupware

In organisations people work together to reach a common goal. The formal interaction between members of an organisation is described by structures and procedures. Additionally there exist informal interactions which are very important. Both types of interactions can and should be supported by computers. Computer Supported Cooperative Work (CSCW) deals with the study and development of computer systems called groupware, which purpose it is to facilitate these formal and informal interactions /ENGLEH88/.

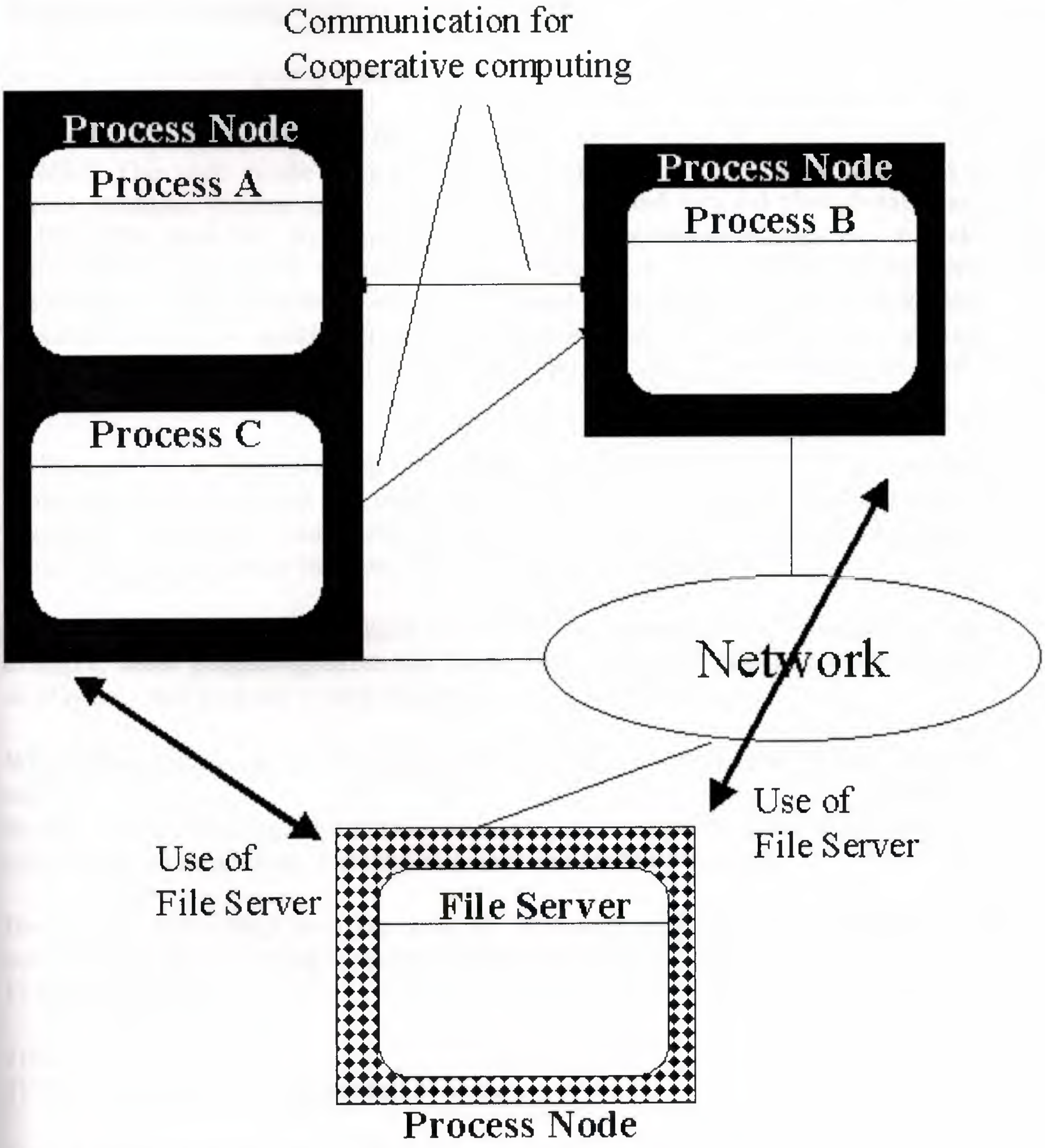
CSCW projects can be classified into four types /ENGLEHB8/ namely:

1. Groups which are not geographically distributed and require common access in realtime **Examples**: presentation software, group decision systems
2. Groups which are geographically distributed and require common access in realtime **Examples**: video conferencing, screen sharing
3. Asynchronous collaboration among people who are geographically distributed. **Examples**; notes conferences, joint editing
4. Asynchronous collaboration among people who are not geographically distributed Examples: project management, personal time schedule management

Groupware requires computers connected by a network. Thus groupware systems are distributed systems. Members of a group share data and exchange messages. Therefore groupware software systems are combinations of network and cooperative computing.

Combination of Network Computing and Cooperative Computing

Cooperative computing can be combined with client server systems. Processes in a distributed system can have access to servers. From the standpoint of a client server system the processes of a cooperative system can be considered as client processes. In a technical process control software system a process can collect data from the technical process. This data is stored in a file located on a file server node. The following figure shows an example of a combination of a cooperative and a client/ server system. Process A, Process B and Process C form a cooperative software system. Process B and Process C use the file server. This means that process B and process C are clients of the file server.



Distributed Computing System
A distributed computing system is not yet a Noema. Many of the components are present but some are still missing or not fully integrated. The network would be the communication

mechanism for the distributed computing Noema supporting message passing, protocols, and asynchronous communication. The languages of communication are the protocols built up with bytes of data. Replication and groups of services could be made available with special name space management services available on the network. Some information may be kept in a data warehouse for analysis. Some information could be locally cached. Some functions could be pre-evaluated and stored in anticipation of usage. Both code and data may have a common representation. Thus programs are to be treated as data in some cases and programs in other cases. Not all data can be interpreted as a program. The distributed computing Noema would need a security system with authentication, authorization, and data privacy. The next chapters define how to build a distributed computing Noema.

Distributed Computing System

In our distributed computing system:

A "Node" is a Network-User* Interface (NUI) that provides network access to the WWW*. This node maybe as simple and economical as a "JavaTerm", which has a decent processor, limited memory/cache, I/O devices and optional pheripherals such as CD ROM, hard disk, an input device which handles portable storage etc.. A node could also be a terminal, such as a UNIX workstation, PC or Mac with network capabilities*. Their processing storage and local applications may differ, but their operations should be mostly dependent on their network bandwidth (which network service providers, such as PacTel, MCI provide) and the pipe of the servers (end-service providers).

A "Server" is a computer that provides services interactively. Services include providing executables (e.g. we may remotely load Word and run it in our network interface), database or search engine (e.g Component library of TI), banks, stock broker firms or any entity that handles and processes requests.

A "Site" is a network destination that provides non-interactive information. For example, most people/organization's home page nowadays which contains visual display only and does not accept/require user input is merely a site.

What differentiates a Server from and a Site is: a server is interactive "active" while a site is "inactive."
Serena (aka wleung) argues that the above two could/should be grouped together and called sites, while another definition of Server should be formulated.

During the last group meeting (10/12), Professor Newton mentioned that there could/should be something between a node and servers. This intermediary could be:
1) State Manager

2)Memory
3) Temporary mirror site (proposed by Susan)

State Manager manages things that doesn't fit into the cache, it could be handled by a central "Service Provider"* which interacts with other servers/sites. However, this would present a major security problem; who's to believe that a "Service Provider" would ensure security of clients' data from internal and external access. (Maybe digital signitures would be required to access and retrieve enscripted data, or maybe enscription could be done at the clients or over the network) There would also be a durability problem. What happens when a State Manager goes down? If we have mirror images, then consistency and security problems arise and this all leads us to the ultimate debate of how distributed systems should be architected. As for network main memory and mirror sites, administration problems immediately come up to my mind. How can they be administered, monitored and by whom? How can data security be provided for this virtual object?

My argument is that none of these intermediate onjects should exist, i.e. nodes should interact directly with servers (present model of WWW). At todays' price and technology curve, pockets-sized DRAM or hard-disk at an acceptable price, performance and capacity (>=500MB) is imminent. One might argue that 500MB is not a lot of stoarge. That's because in today's standards, people store executables in their hard disks, but in the future, all people need is their personal documents (e.g. word-processing files, database, spread-sheets, etc.) that they (regularly) edit as executables will be run off the Net. As for large audio, video files and graphically intense operations such as CAD or games, they should stay at their respective servers where an adequate bandwidth and special transmission mechanisms are provided.

State management in this case is done either on a local storage (cache or hard disk) and/or at the server. Less consistency concerns is achieved at the expense of a higher response time for applications (updates need to go as far as the server instead of an intermediate node).

The Future

Microsoft's dominance of local processing will be displaced by major database and database tools (e.g. Oracle, Informix) companies together with software vendors that develop network-based applications that run at the servers, aimed at providing high throughput, scalability, etc.

.

Hardware vendors, such as Cisco and Bay Networks will be a force as well in helping clients design and implement the appropriate network/WAN strategies.

FootNote *

1) A User may be a human being, processes or other computers.
2) WWW may include or be a part of the Information Superhighway.
3) If "Everything" (from mail to Word, Quicken) is run within a network interface, would CPU processing power and speed be relevant in the future, or this will be a hardware issue that primarily interests "Server" side of the operations. Primary end-user concern would be network bandwidth and display capabilities.
4) "Service Provider" could be network services providers such as PacTel or software vendors such as Oracle.

Chap. 22 CCITT Recommendation X.25 361

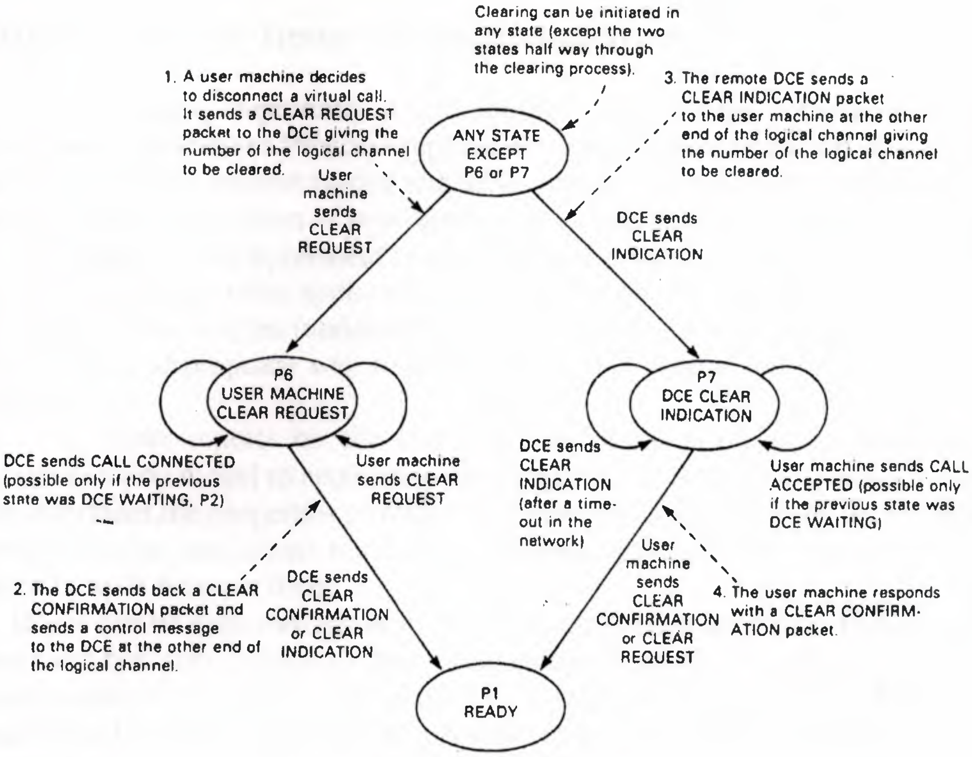

Clearing can be initiated in any state (except the two states half way through the clearing process).
1. A user machine decides to disconnect a virtual call. It sends a CLEAR REQUEST packet to the DCE giving the number of the logical channel to be cleared.
3. The remote DCE sends a CLEAR INDICATION packet to the user machine at the other end of the logical channel giving the number of the logical channel to be cleared.
ANY STATE EXCEPT P6 or P7
User machine sends CLEAR REQUEST
DCE sends CLEAR INDICATION
P6 USER MACHINE CLEAR REQUEST
P7 DCE CLEAR INDICATION
DCE sends CALL CONNECTED (possible only if the previous state was DCE WAITING, P2)
User machine sends CLEAR REQUEST
DCE sends CLEAR INDICATION (after a time-out in the network)
User machine sends CALL ACCEPTED (possible only if the previous state was DCE WAITING)
2. The DCE sends back a CLEAR CONFIRMATION packet and sends a control message to the DCE at the other end of the logical channel.
DCE sends CLEAR CONFIRMATION or CLEAR INDICATION
User machine sends CLEAR CONFIRMATION or CLEAR REQUEST
4. The user machine responds with a CLEAR CONFIRMATION packet.
P1 READY

Figure 22.16 A state diagram for the clearing process. A normal disconnect of a virtual call follows the four numbered steps shown in red.

be permitted either to place calls but not accept calls from other users, or to accept calls but not place them.

Packet Retransmission

A user machine can ask its DCE to retransmit one or several data packets. It does this by sending a REJECT packet to the DCE containing the Receive Sequence Number, R, of a packet received. The DCE retransmits packet R and those following it. The number of packets for retransmission cannot exceed the flow-control window size. This is not an end-to-end mechanism. The request for retransmission of a data packet cannot be relayed to the user which originated that packet.

Flow control parameter selection

A network normally has a given maximum window size and maximum data length. A user machine may optionally operate at less than these because it has limited buffer size or control capability. The window size and maximum data length is referred to as a throughput class and may be indicated in the facilities field of a CALL REQUEST packet. If there is no such indication, the call is connected with the highest attainable values.

HORUS: A Flexible Group Communications System

Computing represents a promissing step towards robustness for mission-critical distributed applications.Proccess replicated for availability or as part of a coherent cache.They can been used to support highly available security domains.And,group mechanisms fit well an emerging geneneration of intelligent network and collaborative work applications.

Yet there is little agreement concerning how process groups should look or behave.The requirements that applications place on a group infrastructure can vary tremendously,and there may be fundamental trdeoffs between semantics and performance.Even the most appropriate way to present the group abstraction to the application depends on the setting.

This paper reports on the Horus system,which provides an unusually flexible group communication model to application-developens.This flexibility extends to system interfaces,the properties provided by a protocol stack,and even the configuration of Horus itself,which can run in user space,in an operating system kernel or microkernel or be split between them.

Horus can be used through any of several application interfaces.These include toolkitstyled interfaces,but also interfaces that hide group functionality behind Unix communication system-calls,the Tk/Tcl programming language,and other distributed computing constructs.The intent is that it be possible to be slide Horus beneath an existing system as transparently as possible,for example to introduce fault-tolerance or security without requiring substantial changes to the system being hardened.

Horus provides efficient support for the virtually synchronous execution model. This model was introduced by the Isis Toolkit,and has been adopted with some changes by such systems as Transis,Psync,Trans/Total ,RMP,and Rampart.The model is based on group membership and communication primitives,and can support a variety of facult-tolerant tools,such as for load-balanced request execution,fault tolerant computation,coherently replicated data and security.

Although often desirable properties like virtual synchrony may sometimes be un-Wanted,introduce unnecessary overheads,or conflict with other objectives such as real-time guarantees.Moreover,the optimal implementation of a desired group communication property sometimes depends on the runtime environment.In an insecure environment ,one might accept the overhead of data encryption,but wish to avoid this cost when running inside a firewall.On a platform like the IBM SP2,which has reliable message transmission,protocols for message retransmission would be superfluous.

Accordingly,Horus provides an architecture whereby the protocol supporting a group can be varied,at runtime, to match the specific requirements of its application and environment.

It does this using a structured framework for protocol composition,which incorporates ideas from systems such as the Unix "streams"framework and the x-kernel,but replaces point-to point communication with group communication as the fundamental abstraction.In horus group communication support is provided by stacking protocol modules that have a regular architectureand in which each module has a separate responsibility.A process group can be optimized by dynamically including or excluding particular modules from its protocol stack.

Horus also innovates by introducing run-time configuration,group communication interfaces full thread-safety,and supporting messages that may span multiple address spaces.Since horus does not provide control operations and has one single address format,protocol layers can be mixed and matched freely.In both streams and the x-kernel,the different protocol modules supply many different control operations,and design their own address format,both severely limiting such configuration flexibility.

1- A LAYERED PROCESS GROUP ARCHITECTURE

We find it useful to think of horus central protocol abstraction as resembling a lego block,the horus"system" is thus like a "box" of lego blocks.Each type of block implements a microprotocol that provides a different communication feature.To promote the combination of these blocks into macroprotocols with desired properties,the blocks have standardized top and bottom interfaces that allows them to stacked on top of each other at run time in a variety of ways.Obviously,not every sort of protocol block makes sense above or below every other sort.But the conceptual value of the architecture is that where it makes sense to create a new protocol by restacking existing blocks in a new way,doing so is staightforwad.

Techically,each horus protocol block is a software module with a set of entry points for down call and upcall procedures.For example there is a downcall to send a messsage and an upcall to receive a message.Each layer is idendified by an ASCII name and registers its upcall and down call handlers at initialization time.There is a strong similarity between horus protocol blocks and object classes in an object-oriented inheritance scheme and readers may wish to think of protocol blocks as members of a class hierarchy.

To see how this works,consider the horus message-send operation.It looks up the message send entry in the topmost block and invokes that fuction.This function may add a header to the message and will then typically invoke message-send again.This time control passes to the message send function in the layer below it.This repeats itself recursively until the bottom most block is reached and invokes a driver to actually send the message.

The specific layers currently supported by horus solve such problems as interfacing the systems to varied communication transport mechanisms overcoming lost packets eneryption and decryption ,maintaining group membership helping a process that joins a group obtain the state of the group merging a group that has partitioned,flow control,etc.Horus also includes tools to assist in the development and debugging of new layers.

Each stack of block is carefully shielded from other stacks.It has its own prioritized threads,and has controlled access to available memory through a mechanism called memory channels.Horus has a memory schedular that dynamically assigns the rate at which each stack can allocate memory depending on availability and priority so that no stack can monopolize the available memory.This is particulary important inside a kernel,or if one of the stacks has soft real-time requirements.

Besides threads and memory channels cach stack deals with three other types of objects:endpoints,groups,and messages.The endpoint object models the communicating entity.Depending on the application it may correspond to a machine ,a process ,a thread ,a socket,a port ,and so forth.An endpoint has an address and can send receive messages.However as we will see later messages are not addressed to endpoint but to groups.The endpoint address is used for membership purposes.

A group object is used to maintain the local protocol state on an endpoint.Associated with each group object is the group address to which messages are sent and a view a list of destination endpoint addresses that are believed to be accessible group members.Since a group object is purely local ,horus technically allows different views of the same group.An endpoint may have multiple group objects allowing it to communicate with different groups and views.A user can install new views when processes crash or recover and can use one of several membership protocols to reach some form of agreement on views between multiple group objects in the sam group.

It does this using a structured framework for protocol composition,which incorporates ideas from systems such as the Unix "streams"framework and the x-kernel,but replaces point-to point communication with group communication as the fundamental abstraction.In horus group communication support is provided by stacking protocol modules that have a regular architectureand in which each module has a separate responsibility.A process group can be optimized by dynamically including or excluding particular modules from its protocol stack.

Horus also innovates by introducing run-time configuration,group communication interfaces full thread-safety,and supporting messages that may span multiple address spaces.Since horus does not provide control operations and has one single address format,protocol layers can be mixed and matched freely.In both streams and the x-kernel,the different protocol modules supply many different control operations,and design their own address format,both severely limiting such configuration flexibility.

2- A LAYERED PROCESS GROUP ARCHITECTURE

We find it useful to think of horus central protocol abstraction as resembling a lego block,the horus"system" is thus like a "box" of lego blocks.Each type of block implements a microprotocol that provides a different communication feature.To promote the combination of these blocks into macroprotocols with desired properties,the blocks have standardized top and bottom interfaces that allows them to stacked on top of each other at run time in a variety of ways.Obviously,not every sort of protocol block makes sense above or below every other sort.But the conceptual value of the architecture is that where it makes sense to create a new protocol by restacking existing blocks in a new way,doing so is staightforwad.

Techically,each horus protocol block is a software module with a set of entry points for down call and upcall procedures.For example there is a downcall to send a messsage and an upcall to receive a message.Each layer is idendified by an ASCII name and registers its upcall and down call handlers at initialization time.There is a strong similarity between horus protocol blocks and object classes in an object-oriented inheritance scheme and readers may wish to think of protocol blocks as members of a class hierarchy.

To see how this works,consider the horus message-send operation.It looks up the message send entry in the topmost block and invokes that fuction.This function may add a header to the message and will then typically invoke message-send again.This time control passes to the message send function in the layer below it.This repeats itself recursively until the bottom most block is reached and invokes a driver to actually send the message.

The specific layers currently supported by horus solve such problems as interfacing the systems to varied communication transport mechanisms overcoming lost packets eneryption and decryption ,maintaining group membership helping a process that joins a group obtain the state of the group merging a group that has partitioned,flow control,etc.Horus also includes tools to assist in the development and debugging of new layers.

Each stack of block is carefully shielded from other stacks.It has its own prioritized threads,and has controlled access to available memory through a mechanism called memory channels.Horus has a memory schedular that dynamically assigns the rate at which each stack can allocate memory depending on availability and priority so that no stack can monopolize the available memory.This is particulary important inside a kernel,or if one of the stacks has soft real-time requirements.

Besides threads and memory channels each stack deals with three other types of objects:endpoints,groups,and messages.The endpoint object models the communicating entity.Depending on the application it may correspond to a machine ,a process ,a thread ,a socket,a port ,and so forth.An endpoint has an address and can send receive messages.However as we will see later messages are not addressed to endpoint but to groups.The endpoint address is used for membership purposes.

A group object is used to maintain the local protocol state on an endpoint.Associated with each group object is the group address to which messages are sent and a view a list of destination endpoint addresses that are believed to be accessible group members.Since a group object is purely local ,horus technically allows different views of the same group.An endpoint may have multiple group objects allowing it to communicate with different groups and views.A user can install new views when processes crash or recover and can use one of several membership protocols to reach some form of agreement on views between multiple group objects in the sam group.

Horus provides a large collection of microprotocols.Some of the most important ones are:

Proposed Sidebar

COM-The COM layer provides the horus group interface to such low-level protocols as IP,UDP,and some ATM interface.
NAK-This layer implements a negative acknowledgement based message retransmission protocol.
CYCLE-Multimedia message dissemination .
PARCLD-Hierarchical message dissemination
FRAG-Fragmentation/reassembly.
MBRSHIP-This layer provides each member with a list of end points that are believed to be accessible.It runs a consensus protocol to provide it users with a virtually synchronous execution model.
EC-Flow control
TOTAL-Totally ordered message delivery .
STABLE-This layer detects when a message has been delivered to all destination endpoints,and can be garbage collected.
CRYPT-Eneryption/deenyption
MERGE-Location and merging of multiple group instance.

The message object is a local storage structure .It is interface includes operations to push and pop protocol headers.Message are passed from layer to layer by passing a pointer and never need be copied.

A thread at the bottom most layer waits for messages arriving on the network interface.When a message on to the layer above it.This repeats itself recursively.If necessary a layer may drop a message or buffer it for delayed delivery.When multiple messages .However since each message is delivered using its own thread ,this ordering may be lost depending on the scheduling policies used by the thread schedular .Therefore,horus numbers the message and uses event count synchronization variables to reconstruct the order where necessary.

2-Protocol Stacks

The microprotocol architecture of horus would not be of great value unless the various classes of process group protocols that we might wish to support can be significant functionality.Our experince in this regard has been very positive.

The layers FRAG,NAK and COM respectively break large messages into smaller ones,overcome packet loss using negative acknowledgements,and interface .Horus to the underlying transport protocols.The adjacent stack is similar,but provides weaker ordering and includes a layer that supports “state transfer”to a process joining a group or when groups merge after a network partition .To the right is a stack that supports scaling through a hierarchical structure in which each parent process is responsible for a set of “child”processes.The dual stack illustrated in this case represents a feature whereby a message can be routed down one of several stacks,depending on the type of processing required.Additional protocol blocks provide functionality such as data eneryption packing small messages for efficient communication ,isochronous communication .

Layered protocol architectures sometimes perform poorly.Traditional layered systems impose an order on which protocols process messages limiting opportunities for optimization and imposing excessive overhead.Clack and Tennenhouse have suggested that the key to good performance rests.Systems based on the ILP priciple avoid inter-layer ordering constraints and can perform as well as monolithically structure system.

3-Using Horus to build a robust groupware application

Earlier we commented that horus can be hidden behind standart application programmer interfaces.A good illustration of how thisdone arose when we interfaced the graphical programming language to horus.

A challenge posed by running systems like horus side with a package like windows .
That such packages are rarely designed with threads or horus communication stacks in mind .To avoid a complex integration task.

Architecturally,CMT consists of a multi-media server process that multicasts video and audio to a set of clients.We decided to replicate the server using a primary –backup approach.Where the backup servers stand by to back up failed or slow primaries.

4-Electra

The information of process groups into CMT required sophistication with horus and its intercept proxies.Many potential users would lack the sophistication and knowledge ofhorus required to do this hence we recognized a need for a way to introduce horus functionality in a more transparent way.This goal evokes an image of “plug and plug” robustness,and leads one to think in terms of an object-oriented approach computing.

The common object request broker architecture (CORBA) is emerging as a major standard for supporting object-oriented distributed environments.Object-oriented distributed applications that comply with CORBA can invoke one-another methots with relative ease.Our work resulted in a CORBA compliant interface to horus which we call electra .Electra can be used without horus,and vice versa ,but the combination represents a more complete system.

IMPLEMENTING BRANCH-AND -BOUND ALGORITHMS ON A CLUSTER OF WORKSTATIONS - A SURVEY, SOME NEW RESULTS AND OPEN PROBLEMS

Abstract

Networks of workstations running under a multiuser, multitasking operating system like UNIX are an increasingly commonplace personal computing environment. Due to their use as personal computers these workstations are typically underutilized most of the time. Thus it is attractive to develop software to use the ample free computing resources to configure a loosely coupled multi computer to solve computation intensive problems in a distributed fashion. In this paper we discuss the feasibility of implementing Branch-and-Bound algorithms for combinatorial Optimization on a cluster of workstations. Thereby we use experiences made by us when solving the Vertex-Cover-Problem on a cluster of 8 HP 9000 - 330 workstations under HP-UX connected via Ethernet and reports from literature about combinatorial optimization on multi computers. Besides presenting performance results we discuss programming techniques balancing for the workload, for interprocess communication and for distributed termination. Based on this evidence we conclude that given proper tuning a distributed Branch-and-Bound algorithm can yield satisfactory speed-up on a cluster of workstations. However, tools are needed that make the development and run-time control of such applications easier while preserving the favourable efficiency.

The Economics of Distributed Computing

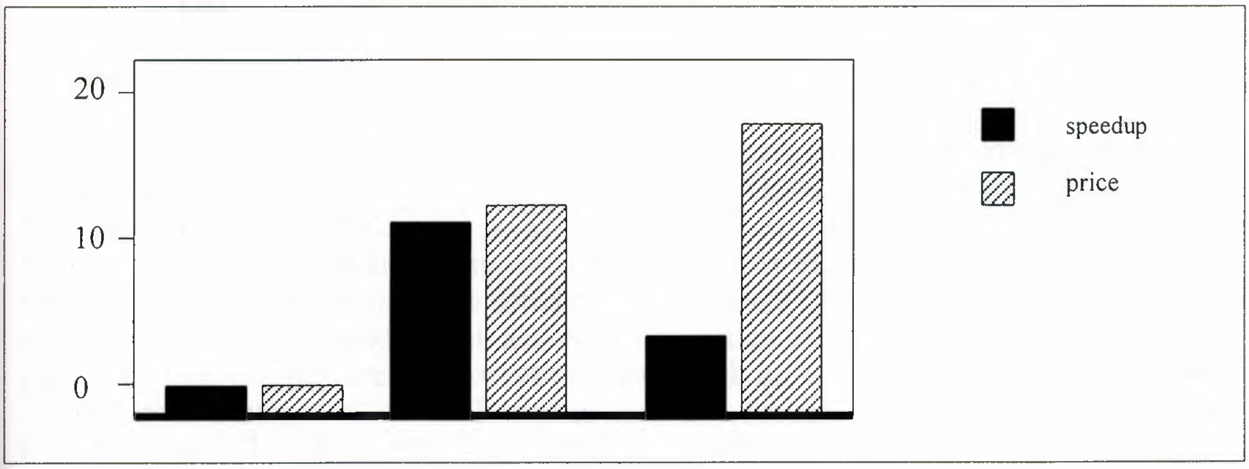
Organizations are increasingly using a network of workstations running under a multi user, multitasking operating system like UNIX to support their members with computing power that can be easily and flexibly accessed from the working place (see the table of computer unit sales taken from [LYNCH9O]). Typically, a workstation in such an environment is used primarily by it's owner, mostly for tasks like Computer Aided Design, Software Development or High Quality Typesetting. The programs supporting these tasks can be executed largely independent of programs run on other nodes, and the network is used primarily for sharing files, programs and I/O-devices. Therefore, a workstation is often underutilized, e.g. while its owner is occupied with non-computerized tasks or is executing computationally lightweight tasks like text-editing. In such an environment it becomes attractive to think about developing software that "collects" the free cycles on the workstations and uses these otherwise wasted resources to solve a problem that is computationally too demanding for one machine. Obviously, this approach is only several workstations can work simultaneously on different subproblems relatively independent feasible if the problem under consideration *can* be decomposed in such a way, that of each other.

Type of Computer	Unit Sales 1983	Unit Sales 1988
Mainframes	1,851	1,497
Workstations	3,460	180,000
Personal Computers	5,900,000	9,400,000

decomposable computation-intensive problem is the relative cost-effectiveness of such a multicomputer when compared to one strong single-processor machine. This sounds paradox from the point Another rationale for looking at a cluster of workstations as the target for implementing a of view of Grosch's Law, which states that whereby c represents the cost at a computer system, k **ist** a constant and e is the computing power of the system. In the Sixties, Grosch's Law was empirically valid, i.e. it could be observed that indeed the rate of growth of the cost of a computing system was under-proportional to its power (see e.g. [SHARPE69]). However, today we find increasing cost per MIPS for different types of comupters (see the table taken from [LYNCH9O] below):

Type Of Computer	Cost per MIPS (in Thousands USD)
Mainframe	180
Super – Mini	60
Small Department Computer	20
32-Bit Personal Computer	2

Thus, while computing power increases linearly as does cost when an additional processor is added to a multicomputer that solves a decomposable problem, the cost increases overproportional when one tries to achieve the same increase in power with a single-processor configuration in a higher machine class. Notwithstanding these economical arguments, paralleization ist the only alternative if no single-processor machine strong enough for a certain problem exists. To illustrate our arguments, we give cost and performance figures for a Branch-and-Bound algorithm for solving the Vertex-Cover-Problem, executed on a cluster of 8 HP 9000/330 workstations and on a HP *9000/845.* A HP 9000/330 is equipped with a 68020/68881 CPU/FPU and 4MB RAM, a HP 9000/845 is a multiuser system with HP Precision Architecture and 48 MB RAM. With the workstation-cluster we achieved an *7.5* increase in power via distribution at the 8-fold cost, while a speed-up of 2.35 via the use of the stronger single processor system would increase the cost by a factor 16.



The condition for achieving such savings is that the implementation exhibits sufficient speed. up, i.e. that one can indeed combine several "2CV to drive as fast as a Ferrari", as illustrated below, thus the following chapters are concerned with techniques for coordinating workstations to efficiently work together on the solution of one problem. Thereby we use the example given above and similar findings from literature to illustrate our ideas.

Cordinating distributed Branch-and-Bound algorithms

Branch-and-Bound is a standard principle to solve combinatorial optimization problems (for a reference of Branch-and-Bound algorithms see e.g. [LAWLER66]). The basic idea is to traverse the space of possible solutions in a tree-like manner, whereby new instances are generated via applying a "branching procedure" to existing ones. A "bounding procedure" is used to prune parts of the tree which cannot contain an optimum based on the knowledge obtained so far. Phrased in a more algorithmic way, a Branch-and-Bound algorithm functions as follows:

```
form an en~,ty pool of subproblems and initialize the bound
while there are subproblems left to inspect do:
        select one subproblem and apply the branching rule
        for each child do:
                compute the lower bound
                if a feasible solution with a better bound is found
                        then record it and update the boupd
                        else If the child cannot be pruned
                                then add it to the pool of subproblems
        end
end
```

One obvious way to construct a distributed version of such an algorithm is to simultaneously "branch and bound" from several subproblems by the available processors. On first sight, one might think that such a distributed algorithm exhibits linear speed-up. However, in [LA184] Lai and Sahni show that even when the processors can access the tree elements and the bound at the same speed as one processor, "anomalies" can occur. That is, it can happen that a distributed Branch-and-Bound algorithm which uses n_2 processors may take more time than one that uses n_1 processors even though $n_2>n_1$ or that one can achieve speed-ups that are in excess of n2/nl. The first phenomenon is due to the fact that often several nodes have the same value of the bounding function, so that the search direction of the processors can be distracted from a promising part of the search space. On the other hand, a distributed algorithm may find a good feasible solution earlier than a sequential one and can use it to prune nodes which would be expanded in the sequential case thus causing super linear speed-up.

A cluster of workstations is a loosely-coupled multicomputer, i.e. each processor has a local memory, which it can access considerably faster than the memory of another processor via the network. For instance, in our configuration - HP 9000/330 workstations under HP-UX connected via Ethernet - we found that for a test graph with 70 nodes and average degree 30 a workstation can process 28.8 subproblems per second for the Vertex-Cover-Problem if the heap[1] containing the nodes of the Branch-and-Bound tree is stored locally. If it has to access a heap located at another workstation to store and retrieve subproblems it can process only 1.1 subproblems . Thus if one stores all subproblems on a particular workstation designated as "problem server", the other processors spend much more time in accessing the problem pool and bound than they would in the sequential case and speed-up suffers[2]. If one inspects the Branch-and-Bound algorithm given above, one finds that it is not necessary for a processor to be able to access all tree-elements but rather only one instance and the bound. Thus one can distribute the heap containing the subproblems to reduce access over the network. However, when using this storage method, strategies have to be devised to prevent that some processors run out of subproblems while others are still working ("idle time"). Another potential problem caused by distributed storage is that it can happen that a processor inspects subproblems which it could prune would it know the better bound stored in the memory of another processor ("search overhead"). Furthermore, the detection of the termination condition is more complicated than in the central case and one has to find a strategy for initializing each processor with a proper set of subproblems.

Initialization can be achieved if one node generates subproblems and distributes them to the other nodes. A method that uses fewer messages at the price of more local computing is to let all nodes begin to work at the same problem and to enumerate subproblems (see [VORNBERGER86]). As soon as a processor has a number of subproblems in its heap that exceeds its processor number it removes all subproblems except the last one generated from its local heap. Then it continues to work on that sub problem. Distributed Termination can be achieved by defining a Hamiltonian cycle on the network and one master processor (see IIL0LIING89]). If the master node is idle, it sends a "yellow token" to its neighbor. Idle nodes pass the token to their neighbor, until the master is reached again. When the master is idle and has not received any new subproblems while the token was circulating, it sends a "red token" to its neighbor. All nodes pass the "red token" on if they are idle and did not receive any subproblems since they have received the "yellow token". If the "red token" returns to the master, all nodes are idle and the master can start collecting the solution.

The trickier part in an implementation of a distributed Branch-and-Bound algorithm is to find a strategy for synchronizing the bound and supplying the processors with "good" subproblems to avoid, idle-time and search-overhead while keeping network traffic low. Two questions have to be addressed when developing such a strategy:

- What kind of strategy should be used?

That is, what kind of local events should trigger a communication to which node to synchronize the bound and/or to transfer subproblems in what fashion?

- How should the strategy be adapted to different problems and configurations?

Obviously, there is a trade-off between search-overhead, idle time and communication effort. The more efficient communication is, the more often a node can coordinate with other processors.

These questions have been the subject of a number of experimental studies. Before we present these results, we review the metrics used to describe the performance of a distributed Branch-and-Bound algorithm:

- Total run time(t)

  The time necessary to compute the solution. It includes the time needed to distribute the initial problem to the nodes, to detect that a solution has been found and the time needed to collect the solution.

- Idle time (idle)

This is the sum of all times a node sits idle waiting for new problems except the time until the node receives subproblems for the first time.

- Iterations (iter)

The number of iterations made by one node. This number is an indicator for the balancing of the workload across the nodes and an indicator how many iterations more/less than the sequential version had to be carried out to get the result.

- Iterations per second (itps)

This nuber indicates how fast new problems are generated. If we compare the itps rating of the sequential version with that of the parallel version, we see how additional communication effort affects the speed of computation.

- Speedup (su)

is defined as ratio of time needed by the sequential version (tl) to the time needed when using k nodes in parallel (tk)

- Efficiency (eff)

is a measurement of how well the processors are utilised. It is computed as speedup divided by number of nodes.

- Search-Overhead (reliter)

We let itersequ be the number of iterations needed by the sequential algorithm and define the search-overhead reliter as

The earliest study about load-balancing in Branch-and-Bound algorithms known to the authors was reported by Vornberger [VORNBERGER86]. He studied a distributed implementation of Held and Knrps Branch-and-Bound algorithm for solving the Travelling - Salesman-Problem (TSP). For this algorithm a minimum spanning tree algorithm is used as bounding procedure, subproblems are generated by fixing certain edges to be members of the cycle. Vornbergers target configuration was a ring of 16 personal computers equipped with Intel 8088 processors, 256 KB RAM and two R5232 serial ports that allow communication with each of the neighbours of a PC. His first strategy was that each processor works on its local heap and requests a problem instance from its neighbours only if its local heap is idle.

Experiments with this method of coordination revealed several pritfalls of this simple distributed mechanism:

- if it has to access a heap located at another workstation to store and retrieve subproblems initially, only the master node that starts the computation has subproblems, all other heaps are empty, causing the other processors to

  (inefficently) requesting work from their neighbours. Thus communication effort is wasted, as it does not reduce the initial idle time.

- the problem given upon request from an idle processor usually is finished quite fast, causing the receving processor to communicate again.

- as the bound obtained by a neighbouring node is only made known to a node if its or its neighbours heap is empty, search overhead is possible, i.e. problem instances are inspected even though somewhere in the network a bound is stored that indicates that these instances cannot possibly lead to the optimum.

To cope with these problems, Vornberger introduced three heuristics:

- At the beginning all k processors (not only the master) put the initial subtour into the heap, then iterate the while loop until at least k subtours are in the heap and then the processor with number i deletes from its heap all subtours but the i-tb cheapest This guarantees a fast distribution of disjoint , roughly equal sized, subtours to all processors.

- Upon request, not only one problem is sent but several, depending on some heuristic arguments such as the total number of subproblems in the heap and the difference between the lowest bound in the heap and the cost of the temporary solution.

- Processor A asks his neighbour B for work not only with an "unconditional request" when A has run out of reasonable subtours but also at certain intervalls (depending on an increase of the smallest lower bound in his heap by a certain constant) with a "conditional request". Such a request is only granted by B, if B detects that his own smallest lower bound is mailer than Ks smallest lower bound by more than this constant."

The f9llowing figure depicts the speed-up obtained with the improved strategy for 20 experiments with random graphs with 30 vertices, degree 4 and edge costs varying between 1 and 200. As can bee seen, via good tuning Vornberger could achieve a satisfactory speed-up - even on his rather limited configuration.

In [KUMAR87] Kumar et.al. study the impact of storage and coordination methods on Parallel Best-First Search of State-Space Graphs via a parallel A* algorithm, a Branch-and-Bound algorithm that always selects the element with the lowest bound for expansion. The experiments reported were conducted on a BBN Butterfly shared-memory multiprocessor, which consist of up to *256* processor-memory pairs whereby each processor's local memory is accessible to other processors via a fast switch. On this configuration Kumar et.al. implemented, among others, various versions of this algorithm for the TSP and the Vertex-Cover-Problem (VCP).

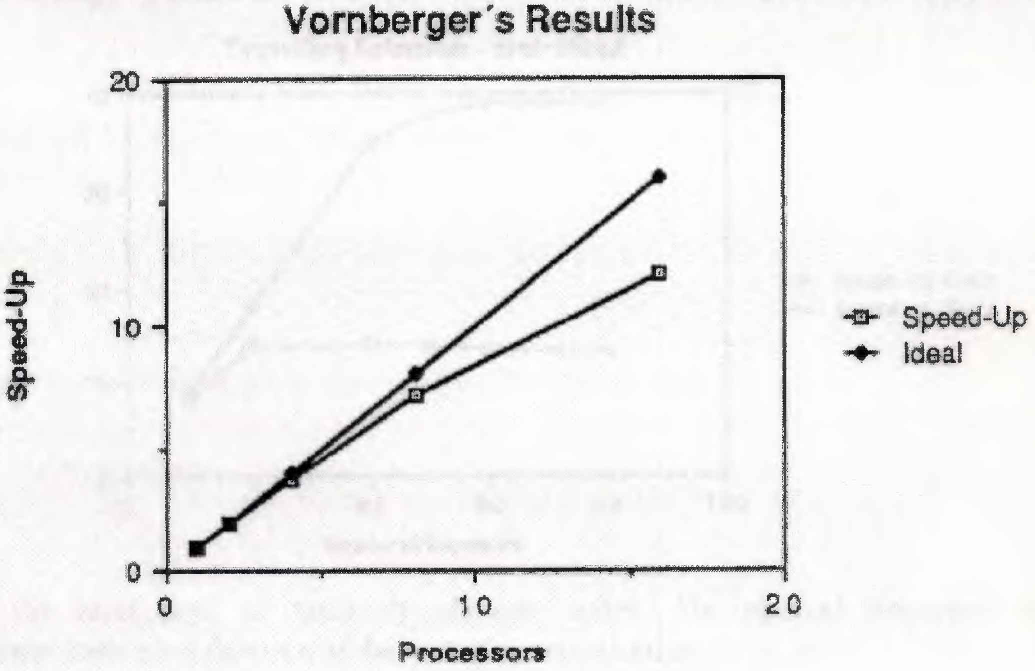
The VCP is defined on an undirected Graph G = (V,E). A subset U of V is called a Vertex-Cover, if for all vertices (u,v) either u or v is element of U. The solution for a given Graph G is that Veitex-Cover which has minimal cardinality.

To generate subproblems for a given subcover sp one applies the following rules (for further details see [MONIEN81ID]):

- select one node x from that part of G, which has not yet been covered by sp

- generate two new subproblems SP1, SP2

The bound b for a subproblem can be computed as b = ICI + IMI where CI denotes the cardinality of the subcover C and IMI denotes the cardinality of a "maximum matching" M in the restgraph (IMI is equal to the number of nodes which must be added to the subcover in order to cover the whole graph G).



The performance of the TSP with a centralized implementation of the subproblem pool is shown in the following figure. Since the BBN Butterfly is a shared memory machine, memory contention can occur. The maximum speedup of such a configuration is hence Texp/Taccess whereby Texp denotes the time needed to process a subproblem and Taccess denotes the time needed to access the subproblem pool. We see that in the 15 city-case contention inhibts further speed-up at about 13.7, while in the *25* city-case contention occurs only at a higher level. This is because the time to expand on subproblem in the TSP is $0(M^2)$, with M equalling the number of cities involved.

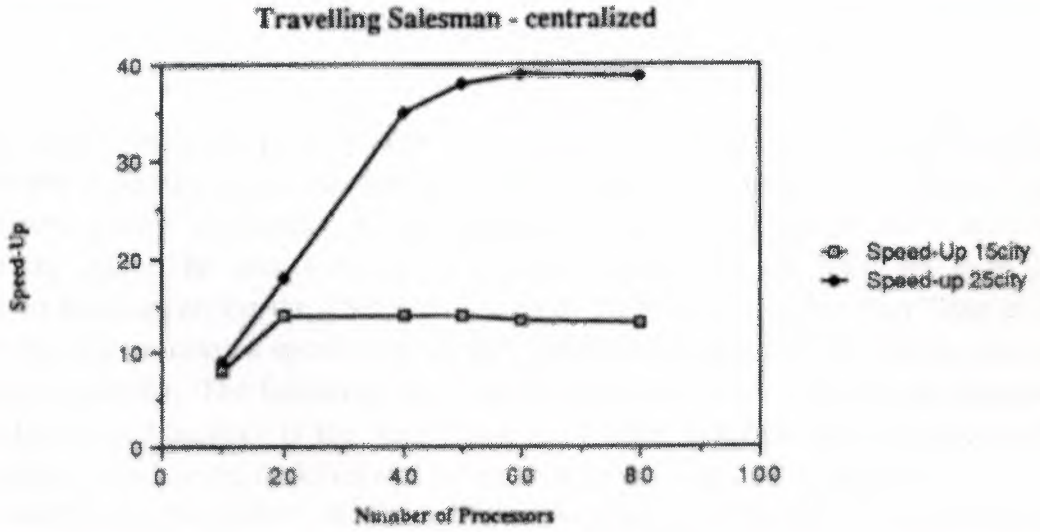
To avoid this contention, distributed control mechanisms can be adopted. Kumar investigate the following strategies:

Blackboard Strategy:

Each processor maintains a local heap. It selects the node to be expanded from the local subproblem pool and then compares this node with the nodes stored on a central "blackboard". If the local node is not within a certain limit, some good nodes are transferred from the blackboard to the local pool or vice versa. Then the node to be expanded is selected again from the local pool.

- **Random Communication Strategy**:

This strategy is suited for topologies with small diameters. Expanded subproblems



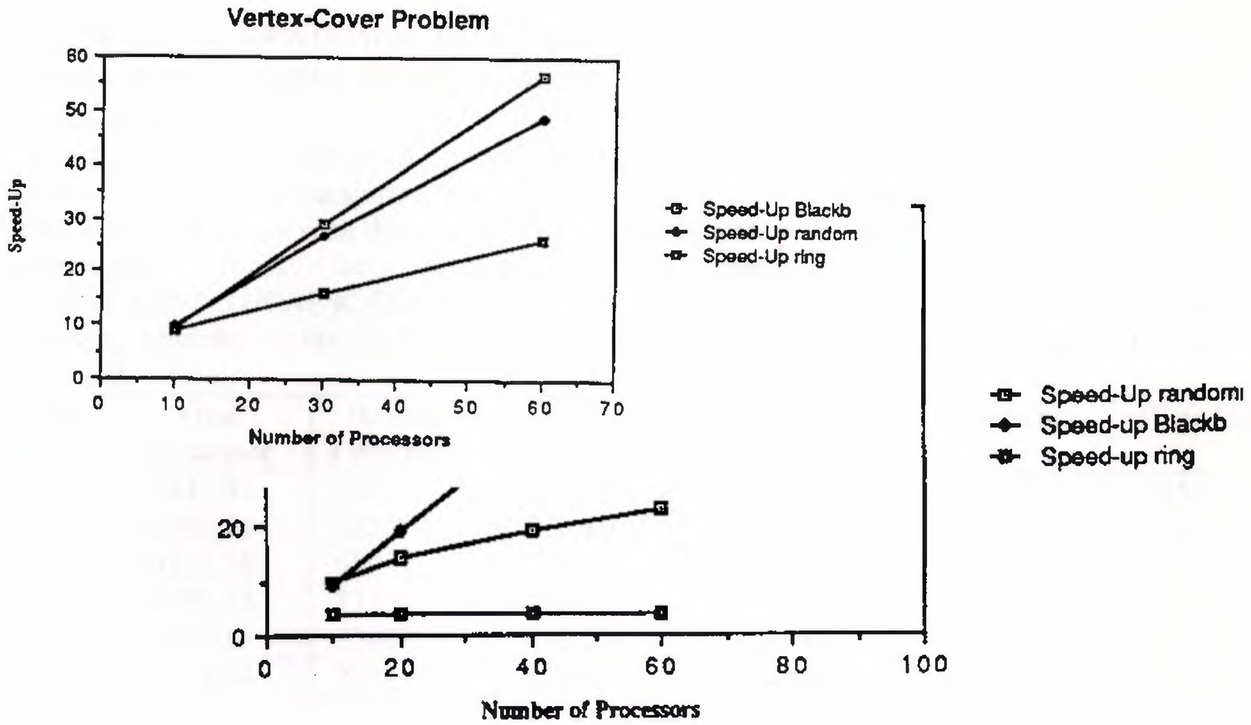
1

nto the local pool of randomly choosen nodes. The optimal frequency of communication is a function of the cost of communication.

- **Ring Communication Strategy**:

This strategy is preferably used with topologies with large diameters. Each processor forms an element of a virtual ring and has two neighbours. Newly generated subproblems are inserted into the pool of the selected neighbour.

Below the results for these strategies used to solve the TSP are given. The parameters of the strategies were determined experimentally. It turned out that the Blackboard strategy was superior to the other strategies, of which the random strategy showed better performance.



The VCP is more prone to speed-up anomalies than the TSP, because for this problem there arc many nodes in its state-space tree that have the same cost as the least-cost solution. Speed-up is therefore greatly determined by how early one processor expands the subproblem leading to the goal. The results of the centralized strategy for the VCP are even less favourable as those given for the TSP, since Texp of the VCP is smaller than Texp of the TSP. Therefore the processors spend most of their time in accessing the subproblem pool and thus causing congestion. The following table depicts the results for the distributed strategies. Again the blackboard strategy is the best. However, for this problem the difference to the random strategy, which again ranks second, is not as large as in the case of the TSP.

Another study that probes into the relative advantage of centralized versus distributed storage of the subproblem pool in distributed Branch-and-Bound algorithms is presented in [LULINGS9j. Lüling studied two strategies for solving the VCP on a network of transputers. One is based on a distributed heap structure, the second one uses a ternary tree structure to store all subproblems at a designated root node. Lüling uses the following rules to coordinate the local heaps in the distributed version:

- If the heapweight of processor i increases more than HEAPW_UP percent, node i sends subproblems to its neighbours.
- if the heapweight of processor j decreases more than HEAPW_DOWN percent, node i sends his local heapweight to its neighbours.

- if node i receives a message containing the heapweight of node j, it sends subproblems to node j, while heapweight(j) > (1+TRESH)*heapweight(i).

- subproblems are only distributed while heapweight(i) > MIN_HEAPW.

HEAPW_UP, HEAPWDOWN, TRESH and MIN_HEAPW are parameters of this strat whereby the heapweight is defined as number of elements stored in the heap that may lea a better solution.

For 20 graphs with 150 nodes and average degree 30 benchmarks were run on a networ up to 63 Inmos T800 transputers. One run was conducted on a transputer network consis of 60 processor nodes with diameter 5. For this configuration the parameters where the be performance could be observed in a number of experiments w HEAPW_UP=0.5,HEAPW_DOWN=0.2, TRESH=0.2 and MIN_HEAPW=:5. In following table the results for 5 typical graphs on a 60 processor topology with diameter 5 presented:

Graph Id	Time 1 processor	iterations 1 processor	time 60 processor	iterations 60 processors	speedup	effic
0	11842,97	38100	205,79	38149	57,54	0.9
1	20998,21	66335	356,89	66343	58,84	0.98
2	20826,10	65951	354,27	65958	58,79	0.97
3	10796,53	33828	181,61	32513	59,44	0.99
4	10555,15	33003	209,53	37944	50,37	0,83
	75018,96	237217	1308,09	240907	57,35	0,95

They show that the efficiency of this implementation is unformly high, the observe variance being quite small. Experiments on 32 nodes configured as ring revealed that in thi case good parameters for the coordination rules were IEAPWUP=0.2, HEAPW_DOWN 10, TRESH~=0.10, MIN HIEAPW=5. The results presented below are consistent with thos reported in [KUMAR87]: they show that the ring structure is not as efficient as the structur with smaller diameter, since small differences add up over a number of processors and yiel in large overall differences of work-load. This can be seen also from the parameter setting used: communication is much more encouraged with these low settings as opposed to the medium ones shown above.

Graph Id	Time 1proccessors	Iterations 1processors	Time 32processors	Iterations 32processors	speedup	efficiency
0	11515,95	38100	369,09	37194	31,20	0,975
1	20414,90	66335	669,27	66766	30,50	0,953
2	20257,59	65951	666,81	66307	30,37	0,949
3	10500,24	33828	391,12	38360	26,84	0,839
4	10087,81	33003	344,52	34470	29,28	0,915
	72776,49	237217	2440,81	243097	29,81	0,932

To compare these results with the performance with a centralized storage tests were performed using 63 transputer nodes configured as ternary tree, whereby only the root node stores the heap. To cut down on accesses of the root node, a subproblem that has the same value of the bounding function than its parent is not inserted into the central heap but kept local as input to the next iteration. In view of the results presented above, one would expect that this centralized strategy would lead to congestion quickly. However, as the table below shows, the centralized strategy outperformed the decentralized version:

Graph id	Time 1processor	iterations 1processor	time 63processor	iterations 63processors	speedup	efficiency
0	5488,69	37953	88,64	38033	61,92	0,983
1	9499,36	66768	152,60	66815	62,25	0,988
2	9401,62	66017	150,88	66035	62,31	0,989
3	5015,47	34038	81,06	34188	61,87	0,982
4	4807,94	32949	77,70	33005	61,88	0,982
	34212,98	237725	550,88	238076	62,11	0,986

This indicates how efficient the communication mechanism provided by the transputers is. As can be seen from the table below, the root process is only very slightly blocked by the additional communication necessary and that the load is well-balanced and idle times are small. Lüling has done additional research on ternary-tree based configurations which shows that for constant communication and storage management cost the speedup increases, if the computation time for a single subproblem on a slave processor is sufficiently high. However, it is questionable whether this result also holds for larger networks, especially as the memory of one transputer may not be large enough for storing the heap for larger problems so that a distributed storage becomes necessary from this point of view.

Computation time	152,6
Idle time root process	0,6350
Min idle time slaves	0,5496
Max idle time slaves	1,3743
Avg idle time slaves	1,1011
Iterations root process	895
Min iterations slaves	1007
Max iterations slaves	1133
Avg. iterations slaves	1063

The aim of our implementation is to study whether the techniques developed by Vornberger, Kumar et.al. and Lüling for coordinating distributed Branch-and-Bound algorithms on a tightly-coupled multicomputer are also applicable to achieve a satisfactory speed-up on a loosely-coupled multicomputer such as a cluster of workstations. We implemented both sequential and parallel Branch-and-Bound algorithms for solving the VCP on HP 9000/330 workstations using the C programming language. Randomly generated test-graphs with 120 nodes and average degree 30 were used for the performance measurements. With this input, the sequential version took 4:13 hours (15,180 seconds) on the average to complete on an unloaded workstation running in multiuser-mode with all usual background daemons active. On the average it performed 159,009 iterations at an average of 10.47 iterations per second. The distributed version was benchmarked using 8 unloaded networked HP 9000/330 workstations. Thereby we basically used Lülings coordination strategy described above. Compared to the communication in a transputer network, in our environment communication is slow. Thus we have to expect that the favourable parameter settings differ from those found by Lüling. Furthermore, on workstations under UNIX context switches are expensive and should be avoided. As will be described in the next section, we implemented the computions and heap management in seperate processes, whereby the heap management process checks the local heap and bound whether a coordination rule fires. To cut down context switches, we suspend

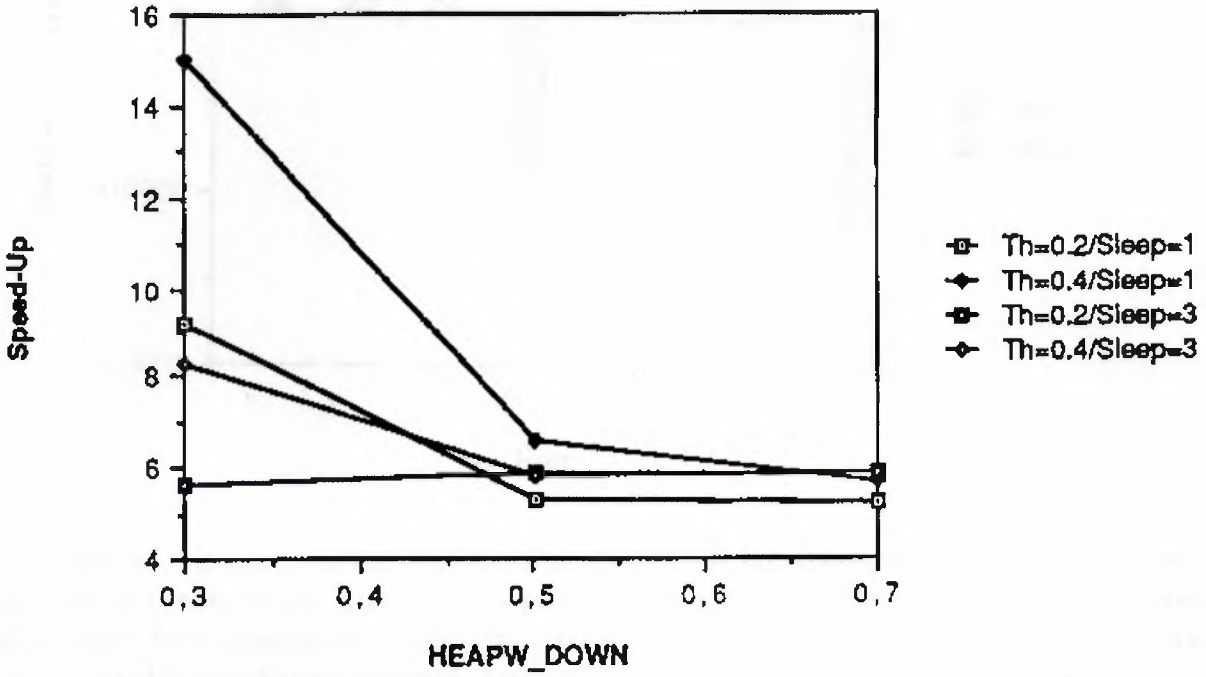
this prpcess after each scan for an interval speciefied by the parameter SLEEP. Due to t~ operating-system limitations the minimum value for SLEEP is one. Therefore we chose as test values for SLEEP one and three. A total of 180 experiments (at least 5 per parameter setting) was conducted with the following parameter-settings.

Parametreler	Values		
HEAPW- UP	0,5	0,7	
HEAPW-DOWN	0,3	0,5	0,7
TRESH	0,2	0,4	
MIN-HEAPW	5		
SLEEP	1	3	

The following table summarizes the values for the above introduced metrics per parameter setting:

HEAPW-UP	HEAPW-DOWN	TRESH	SLEEP	t	V(t)	iter	V(İT ER)	SU	V(SU)	rel ter	V(rel Iter)
0,5	0,3	0,2	1	1895,66	500,48	825,33	103,36	8,43	1,96	0,92	0,21
0,5	0,3	0,2	3	2794,33	232,56	624,66	250,60	5,46	0,46	1,20	0,04
0,5	0,3	0,4	1	1586,66	778,60	620,33	210,86	11,92	6,07	0,76	0,34
0,5	0,3	0,4	3	1523,33	523,63	164,66	33,35	11,26	4,58	0,74	0,28
0,5	0,5	0,2	1	3186,00	321,42	540,33	423,24	4,80	0,45	1,30	0,11
0,5	0,5	0,2	3	2678,40	248,00	392,00	319,97	5,70	0,50	1,17	0,06
0,5	0,5	0,4	1	2409,33	64,55	602,33	350,96	6,30	0,17	1,15	0,02
0,5	0,5	0,4	3	2534,33	118,60	739,66	143,27	5,99	0,28	1,21	0,05
0,5	0,7	0,2	1	3003,33	98,08	535,00	191,16	5,05	0,16	1,33	0,06
0,5	0,7	0,2	3	2659,33	93,22	600,00	430,68	5,71	0,20	1,21	0,02
0,5	0,7	0,4	1	2905,66	323,59	372,66	250,79	5,27	0,58	1,32	0,08
0,5	0,7	0,4	3	2545,33	32,50	305,33	239,35	5,96	0,07	1,19	0,03
0,7	0,3	0,2	1	1553,33	155,18	560,00	261,82	9,86	1,05	0,76	0,09
0,7	0,3	0,2	3	2747,00	494,98	508,33	328,14	5,71	1,15	1,06	0,18
0,7	0,3	0,4	1	1025,00	470,79	599,00	358,69	19,57	12,52	0,51	0,23
0,7	0,3	0,4	3	2847,66	479,55	211,66	15,25	5,48	0,99	1,11	0,13
0,7	0,5	0,2	1	2609,33	65,75	276,66	328,70	5,82	0,14	1,21	0,01
0,7	0,5	0,2	3	2463,66	378,19	461,66	192,59	6,27	1,05	1,10	0,09
0,7	0,5	0,4	1	2333,33	602,93	679,66	138,87	6,95	2,06	1,08	0,22
0,7	0,5	0,4	3	2684,00	50,38	796,33	174,43	5,66	0,10	1,22	0,03
0,7	0,7	0,2	1	2576,00	73,50	624,00	253,79	5,89	0,16	1,18	0,03
0,7	0,7	0,2	3	2534,00	257,27	677,00	338,60	6,03	0,60	1,16	0,06
0,7	0,7	0,4	1	2525,66	162,16	633,66	262,74	6,03	0,38	1,20	0,03
0,7	0,7	0,4	3	2575,00	166,04	408,00	414,39	5,91	0,39	1,21	0,04

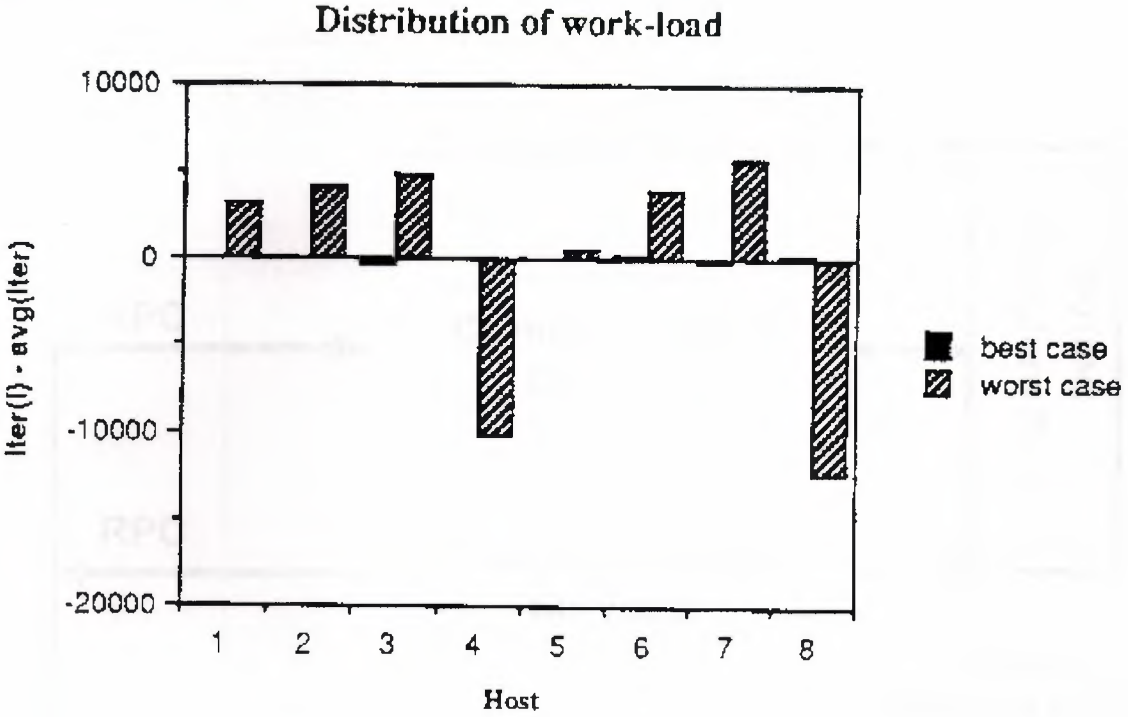
As already noted in IKUMAR87] and [LÜLING89], also in our experiments we found anomalous behavior, i.e. runs with reliter» 1 or reliter « 1. To check whether the parameter settings have significant influence on the total run time, we performed a variance analysis (for details on this technique see e.g.(FAHRMEIER84]). On the 1 percent level the parameters HEAPW_DOWN, TRESH and SLEEP and interactions of HEAPW_UP versus SLEEP and HEAPW_DOWN versus SLEEP were significant. On the 5 percent level the interaction of HEAPW_UP versus TRESH becomes significant, too. To illustrate the dependence of performance on the parameter settings graphically, the following figure shows the speed-up gained versus HEAPW_DOWN, given the parameters THRESH (Tb) and SLEEP.



We see that for our configuration it is favourable to keep all parameters small as is the case in Lüling's ring-configuration. However, due to the more expensive communication our parameters in general are larger than those found by Lüling. The SLEEP parameter can be left at 1 and thus the functionality associated with it can be neglected. We would also like to point out that the significant speed-up gained with the parameter-settings HEAPW_DOWN=O.3, THRESH=O.4, SLEEP=1 is also due to a single anomalous speedup of *35 (!)* and therefore the minimum may be less distinct if more experiments are taken as basis for the determination of the mean speed-up. To illustrate what we can expect from our distributed system on average, we compute the means of the metrics over all runs conducted:

Variable	Mean	Std Dev.	Minimum	Maximum
Total runtime	2300.4	636,70	425,0	3597,0
Total iterations	556,63	285,27	16,0	998,0
Speedup	7,5	4,09	4,220	35,720
reliter	1,0	0,25	0,2120	1,4590

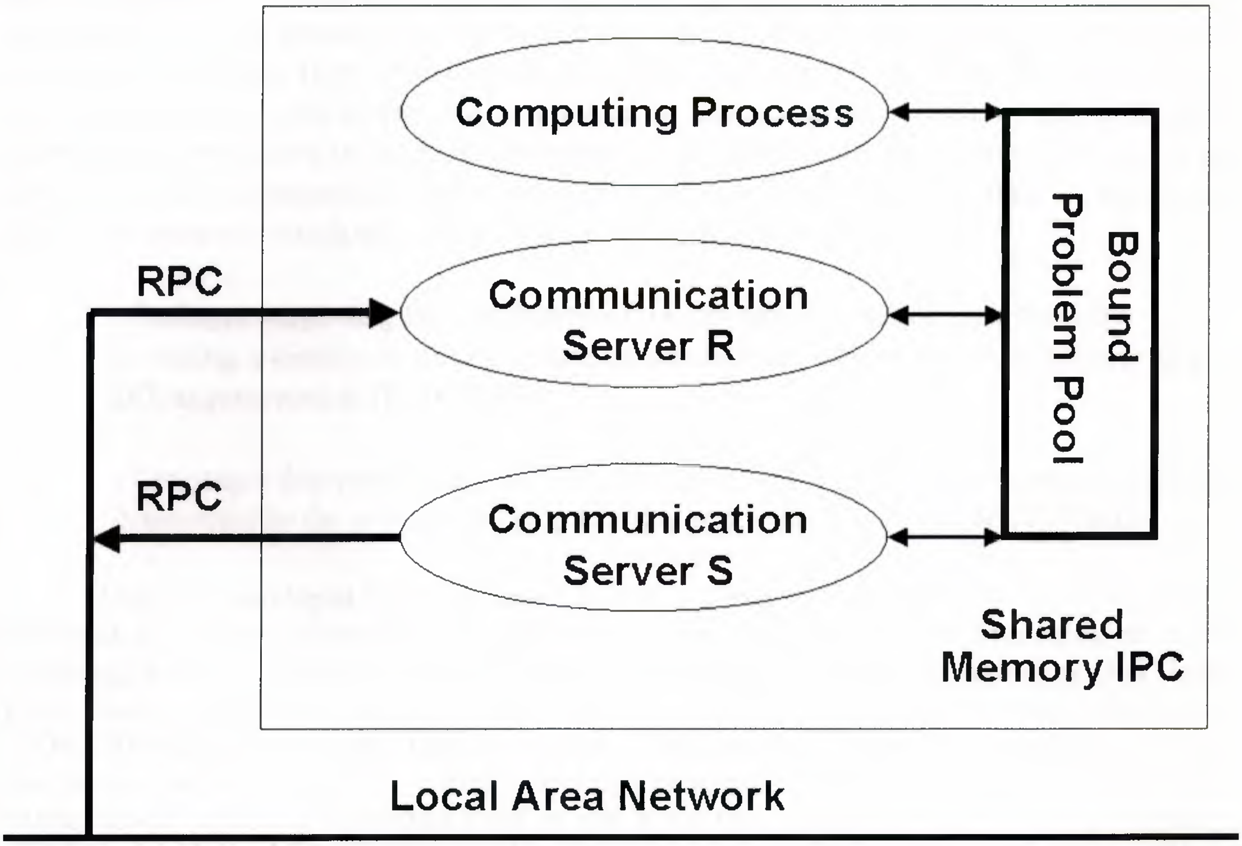
To provide further insight into the coordination achieved via our strategy, the following figure shows the distribution of the number of iterations across the eight nodes for the best and worst-case experienced in our tests.



In view of this experimental data and the results from literature presented above, it seems justified to conclude that from the viewpoint of performance Branch-and-Bound algorithms are suitable for execution on multicomputers. While the speed-up results are comparable for tightly-coupled and loosely-coupled systems, it seems that on the latter one has to expect a higher variance in the performance. However, besides exhibiting satisfactory performance an algorithm should also be easy to program, debug, main tam and use. As we shall demonstrate in the sequel, it is this area where additional work remains to be done.

Programming a and Controlling Distributed Branch-and-Bound Algorithm5 on a Cluster of Workstations

In Our application most of the time a processor "behaves" as if it would be a stand-alone computer executing a sequential Branch-and-Bound algorithm using its local heap. However, if the contacted. This introduces Condition-parts of the coordination rules are satisfied, another node is contracted. This introduces a nonsequential element as the receiver then has to stop computing and to execute the code for answering a communication request. A mechanism that supports this functionality and is available on many UNIX systems is a remote procedure call (RPC). an abstraction similar to the procedure-call-mechanism in sequential programming languages[3].With this mechanism we can separate the computing process from code dealing with coordination and leave the scheduling of the processes to the operating system. As RPC implements synchronous communication, deadlocks can occur if the sending and receiving of messages is implemented in the same code, namely if node A wants to communicate with node B, which at the same time wants to contact node A. Thus we separate these functions in two processes, Communication Server R and Communication Server S. All three processes of the application have to access the local heap, which we implemented using shared memory to which all processes connect. Therefore we arrive at the following structure:



Server S manages the shared memory segment (creation and deletion), starts Server R and the computing-process, does load-balancing and terminates server R and the computing process. If Server S runs on the master node in the network, it also handles insertion of the start element, time measurement, the end-of-computation-detection and the collection of results and statistics. Server R answers incoming requests. Basically, the computing process is identical to the sequential version. Additional features are that the automatic insertion of a start element into the

Local heap is suppressed and that statistics about the idle time are gathered. Furthermore, the program does not terminate on the "heapempty"-condition, but then enters an endless loop waiting for new problems. In order to inform the communication process about the status of the computing process, the computing process puts its current status (running or idle) into a predefined status-register in shared memory.

This distributed program system for solving the VCP as described above consists of twice the Lines of code of the sequential version. Its coding was considerably more elaborate, especially as it involved the use of the novel programming techniques described above. A similar finding L5 reported in [KALLSTROM88], where experiences made when a distributed simulated annealing algorithm to solve the traveling salesman problem was implemented on an IPSC hypercube via C and libraries, on a network of transputers in

OCCAM and in C on a Sequent Balance, a shared memory multiprocessor. Among others, Kallstrom and Thakkar found that programming multi computers in C via libraries is cumbersome and error-prone and that the resulting programs typically are much larger than their counter pieces for shared-memory multiprocessors. Reviewing the structure of our application depicted above one finds that only the computing process is specific for the VCP, while the two server processes could be used for other algorithms of this type too, albeit with different coordination rules. It thus seems plausible to implement the functions provided by these processes for general use and to provide an abstraction of a space of sub problems shared by all processors in the multi computer that the application programmer can use in a similar way like the sequential data structure. According to the level of generality, approaches taken in literature in this direction can be divided into two classes:

- Packages supporting the distribution of tree-traversal backtrack algorithms by providing a distributed data structure for storing the nodes of the backtrack tree (e.g. DIE as presented in (FINKEL87])

- Languages that provide access to distributed shared memory whose structure *can* be determined by the programmer (e.g. LINDA as discussed in [GELBRNTER89II)

DIB was developed by Finkel and Manber as a tool for supporting the distribution of backtrack algorithms. It requires the user only to specify a root-node, the branching and the bounding procedure. The distribution of subproblems and termination is handled by DIB. The load sharing algorithm used is rather simple, similar to the first strategy used in [VORNBERGER86], though DIB tries to avoid the transfer of nodes on lower levels of the tree that can be solved quickly to keep communication low. LINDA offers access to a shared "tuple Space" which is distributed over several hosts in a transparent way. The tupel space consists of tuples, which are content-adressable sequences of fields with a defined type. The tuples can be used to implement a variety of distributed data structures, whose access routines can be implemented via LINDA primitives that allow the insertion and deletion of tuples. Additional to storing passive data transparently, LINDA also provides an operation for the transparent execution of routines, whose results become passive tuples upon completion. LINDA is available for use with a number of programming languages. It has simple routines for storing the tuples at the different nodes (either full replication or storage at the node where a tuple was inserted into the tuple-space). To the authors' knowledge, no further coordination mechanisms have been implemented so far and current research on algorithms for distributed shared memory focuses on proper ways of transparently storing and accessing objects used by several nodes rather than on techniques for load balancing (see e.g. [STUMM9O]). Thus, such tools should provide language constructs that allow the application programmer to use problem- and configuration specific rules for sharing information among several nodes like those introduced above.

Besides these programming problems, also the run-time control of a distributed Branch-and-Bound algorithm on a cluster of workstations is non-trivial. Tightly-coupled multicomputers typically are "node-shared", i.e. at the beginning of an execution a user reserves the nodes he needs exclusively. In our case, the program is run as a collection of background processes on time-sharing machines, each one being controlled by its owner. Thus the implementation must be prepared to deal with a "nonconstant" configuration: the number of iterations per second on a node may decrease because its owner starts a foreground process or a node may become completely unavailable because its owner terminated the processes of the Branch-and-Bound application or due to a power failure or switch-off. Compared to an implementation on a tightly coupled system this makes the coordination more difficult as one also has to deal with load imbalances caused by other processes. To avoid the disturbance of other user processes, LINDA distributes load according to a "policy file" where the owner of a workstation specifies when and under what conditions he is willing to provide cycles on his machine for general use. A safeguard against incomplete tree-traversal caused by node failures is that a host stores all subprbblems it hands over to other hosts. If a host fails, the nodes that have sent subproblems to it reevaluate these instances. If problems were transferred more than one time, additional communications are necessary to determine the subtree lost. The price paid for this functionality is increased network traffic, since a node receiving problems must notify the sender when it has finished evaluating the subproblem handed over. Under DIR a node even does not await the detection of a failure and starts inspecting subproblems sent away if it becomes idle. With regard to the master node, other nodes can take this role if the previously determined master processor has filed. For instance, one can determine that the node having the lowest node-id should take the responsibilities of the master. Lets assume that the node with identification one is the master, that all nodes are idle, node 1 sends out the yellow token, but crashes before the token has completed its round. Now node n gets a timeout when trying to pass the yellow token back to the master. It sends a status message to all other nodes notifying them to remove node 1 from their list of available nodes. Upon receiving this message, node 2 discards the first entry from its node-table and realizes that it has become master. In this role it issues the yellow token and the cycle starts again.

Conclusion

Branch-and-Bound algorithms scale up well, given proper coordination. The speed-up on a cluster of workstations is comparable to the one arrived at when using tightly-coupled multicomputers like transputers or hypercubes, albeit the variance observed over different runs is higher. Thus thinking about the distribution of such an algorithm is worth the effort, especially if processor time available anyhow would be left unused. Moreover, the results demonstrate that for this type of algorithms it can be economically sensible to use several cheap slow processors instead of one single processor configuration. However, these benefits are not for free. Due to the fact that on a loosely-coupled system several applications are run at the same time by different users mostly in an interactive way, the control of a long-running application on such a multicomputer is more difficult than on a tightly-coupled system. Thus special techniques are necessary to avoid the disturbance of other applications and to ensure fault tolerance. Also programming of a distributed version of a Branch-and-Bound algorithm is considerably complex than implementing a sequential version. This is reflected not only in lines of code but also by the need for application programmers to cope with new programming problems such as dissemination of global information, load-balancing, distributed termination or deadlock prevention. Based on this evidence we argue that while it seems that from the viewpoint of performance the feasibility of using a cluster of workstations as a loosely-coupled multicomputer for running Branch-and-Bound algorithms is well demonstrated, it is decisive to give the application programmer tools that ease the programming task to foster the wide-spread use of networks of workstations also as devices for computation intensive computing.

PARALLEL COMPUTING: THEORY AND PRACTICE

PARALLEL BRANCH-AND-BOUND ALGORITHMS

The first two parts of describe parallelizations of the branch-and-bound algorithm for multiprocessors and multicomputers. respectively. We use as our problem domain the traveling salesperson problem described in All the parallel algorithms described in this section use the branching and bounding heuristics developed by Little et al.

The final part of this section discusses anomalies in parallel branch-and-bound algorithms conditions under which adding processors may result in slowdown or superlinear speedup.

Multiprocessor Algorithms

Mohan 1983) has developed two parallelizations of the traveling salesperson algorithm presented . The first parallel algorithm involves a parallelization of the for loop: the second parallel algorithm executes the repeat loop in parallel.

As presented before. the for loop has a natural parallelism of 2—each node has only two children. However, by selecting *k* edges to be considered for inclusion or exclusion, the number of children of each node increases to 2k since constraints reflecting all combinations of inclusion and exclusion must be generated. The modified algorithm is Clearly this data-parallel algorithm is appropriate for 2k processors.

The second algorithm creates a number of processes that asynchronously explore the tree of subproblems until a solution has been found. Each process repeatedly removes the unexplored subproblem with the smallest lower bound from the ordered list of unexplored subproblems. Then it decomposes the problem (unless it can be solved directly), and inserts the two newly created subproblems in their proper places in the ordered list of problems to be examined. A process must have exclusive control of the list in order to insert and delete elements, but the time taken for these tasks is relatively small compared to the time needed to decompose a problem. Thus contention for this list should not be a significant inhibitor of speedup.

The speedup of these two parallel algorithms on Cm* (a NUMA multiprocessor) is contrasted .The first algorithm achieves extremely poor speedup. The additional processors spend most of their time creating nodes that are never explored, because their lower bounds are too high. Mohan's second algorithm achieves, with 16 processors. a speedup of about 8 when solving a 30-vertex TSP. The major obstacle to higher speedup is the number of nonlocal memory references made by the processors.

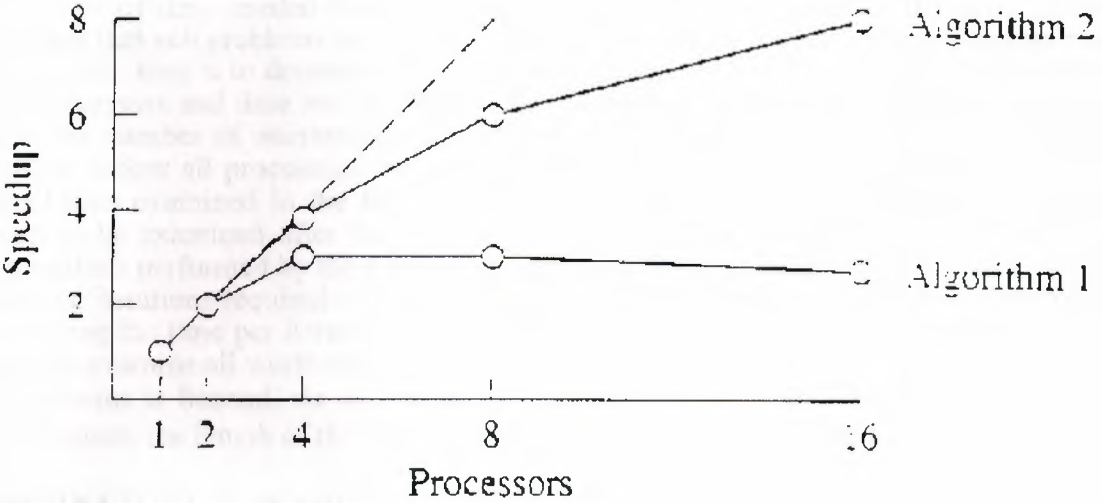
COMBINATORIAL SEARCH

TRAVELING SALESPERSON (UMA MULTIPROCESSOR):

```
begin
            reduce weight matrix, determining the root's lower bound initially only the
            root is in the state space tree while true do
            select the unexamined node in the state space tree
                    with the smallest lower bound
            if the node represents a tour then exit the loop endif
            select the k edges whose exclusion increases
                    the lower bound the most
            for the $_2k$ cases representing all inclusion-exclusion
                    combinations of the selected edges do
               create a child node with the correct constraints
               find the lower bound for the child node
```

```
endfor
endwhile
end
```

High-level description of a parallel traveling salesperson algorithm developed by Mohan (1983). The algorithm is designed for implementation on a UMA multiprocessor, but it does not achieve good speedup.



Multicomputer Algorithms

Quinn (1990) has implemented four variants of an algorithm to solve the traveling salesperson problem on hypercube muiticomputers. The algorithm uses distributed priority queues of unexamined subproblems—one queue per processor. The performance of the parallel algorithm depends upon the heuristic the processors use to exchange unexamined subproblems with each other. Major portions of this subsection first appeared in Quinn (1990).

Let p — $_2d$ denote the number of processors. Assume that the branching factor of the state space tree is *k;* i.e., assume that each node in the tree has *k* children. Let *N* be the minimum number of constraints that must be added to the original problem in order to produce a subproblem that is solvable. In other

PARALLEL COMPUTING: THEORY AND PRACTICE

Words, any solution node must have depth > *N* in the state space tree. Let x be the time needed to examine a subproblem and either solve it or decompose it into *k* subproblems. Let *k* be the time needed to transfer a subproblem from one processor's priority queue to another processor's queue. and assume that both the sender and the receiver processors must devote time X to the transfer.

The asynchronous branch-and-bound algorithm distributes the unexamined subproblems among the processors. In each of its iterations every processor with a nonempty priority queue removes the unexamined subproblem with the smallest lower bound and either solves the problem directly or divides it into *k* subproblems. (Note: Although each processor iterates through a sequence of operations. there is no synchronization among processors.) If a processor divides a problem into *k* subproblems. it puts the subproblems into its priority queue, then uses a heuristic to send *in* of its unexamined subproblems to neighboring processors. where *in* < *k.* At the beginning of execution. Processor 0 contains the original problem in its priority queue. Because each processor distributes *in* subproblems every iteration. lIp) = ~log,,₁ p1 iterations are sufficient to provide every processor with at least one unexamined subproblem.

In order for a solution to be found and guaranteed optimal. two conditions must be met. First, at least one of the solution nodes (and hence all of its ancestors in the state space tree) must be examined. Second processors must examine all nodes in the state space tree whose lower bounds are less than the cost of the optimal solution. The execution time of the algorithm is determined by whichever event occurs last. The event occuring last is determined by the number of processors and the shape of the state space tree.

To derive an expression for the execution time of the parallel algorithm, we first determine the amount of time needed to examine all of the worthwhile nodes in the state space tree. Assuming that sub problems are exchanged evenly among processors. Every iteration requires time x+2mλ*:* time x to decompose or solve a subproblem. Time *m* to send *in* subproblems to other processors and time *mλ* to receive (on average) *m* subproblems from other processors. If S is the number of worthwhile subproblems in the state space tree. I(p) is the number of iterations before all processors are actively involved. and G(p) is the number of worthwhile subproblems examined in the first *1(p)* iterations, then S — *G(p)* worthwhile subproblems remain to be examined after the first 1(p) iterations. If the percentage of worthwhile node examinations performed by the *p* processors after the first *lip)* iterations is *E(p).* the number of additional iterations required to examine all worthwhile subproblems is[F(S — *G(p)/(pE(p))].* Multiplying the time per iteration by the number of iterations, we see that the amount of time needed to examine all worthwhile

siihnrnhb~ms is Second. we determine the amount of time needed for the search to reach a solution node, the length of the critical path. Let *M* denote the depth of the

COMBINATORIAL SEARCH

Solution node in the state space tree. Let *T(p)* denote the number of transfers on the critical path from the root of the state space tree to the solution node. In other words. *T(p)* is the number of times that subproblems leading to the solution are transferred from one processor's priority queue to another processor's queue. Every transfer incurs a penalty of x~'^{2} + *A*. The *A* term is the time needed to perform the transfer. The x/^{2} term is the expected delay before the subproblem can be evaluated by the destination processor. since the destination processor is likely to be in the middle of decomposing another subproblem when the transfer begins. The total amount of time needed for the search to find a solution is

$$(M + 1)(x + 2m\lambda) + T(p)(x/2 + \lambda) \qquad (13.2)$$

Since the asynchronous algorithm completes when both previously mentioned conditions are met, the execution time of the algorithm is the maximum of the times in expressions 13.1 and *13.2.*

Quinn has tested the model by implementing four parallel best-first, branch-and-bound algorithms to solve the traveling salesperson problem. All these algorithms use the reduction heuristic of Little et al. The algorithms have been implemented on a ^{6}4-processor nCUBE 3200 hypercube multicomputer.

All four algorithms have been executed on the same set of ten 30-vertex graphs. The edge weights are asymmetrical and randomly chosen from a uniform distribution of integer values ranging from 0 through 99. Every algorithm beings with Processor 0 possessing the original problem, and relies upon successive subproblem decomposition steps to work toward a solution.

During an iteration every processor with a nonempty priority queue removes the unexamined subproblem with the smallest lower bound and either solves the problem directly or divides it into two subproblems. It sends one unexamined subproblem to a neighboring processor and receives (on average) one unexamined subproblem from a neighboring processor. Quinn determined the parameters needed for the analytical model by recording the actions taken by the processors during their solution of the ten 30-vertex problem instances. All these parallel algorithms had the following parameters: S *559,* *k* = 2, *m* = 1, x 125 msec, and $\lambda = 1$ msec. Values of *G(p), E(p),* and *T(p)* varied from algorithm to algorithm.

All four algorithms use the following rule to distribute subproblems among the processors. Let p — $_2d$ be the number of processors. On iteration *i* Processor *j* sends the unexamined subproblem to Processor *r,* where *r* is found by inverting bit *(i* mod *d)* of *j.* With this distribution rule *1(p)* = log$_2$ p.

Each algorithm has a unique heuristic for choosing which unexamined sub-problem to send to a neighboring processor. Algorithm 1 puts the newly created subproblem with the edge inclusion constraint into the priority queue and sends the subproblem with the edge exclusion constraint. Algorithm 2 puts the newly created subproblem with the smaller lower bound into the priority queue and sends the subproblem with the higher lower bound. Algorithm 3 puts both newly created subproblems on its priority queue, then deletes the second-best

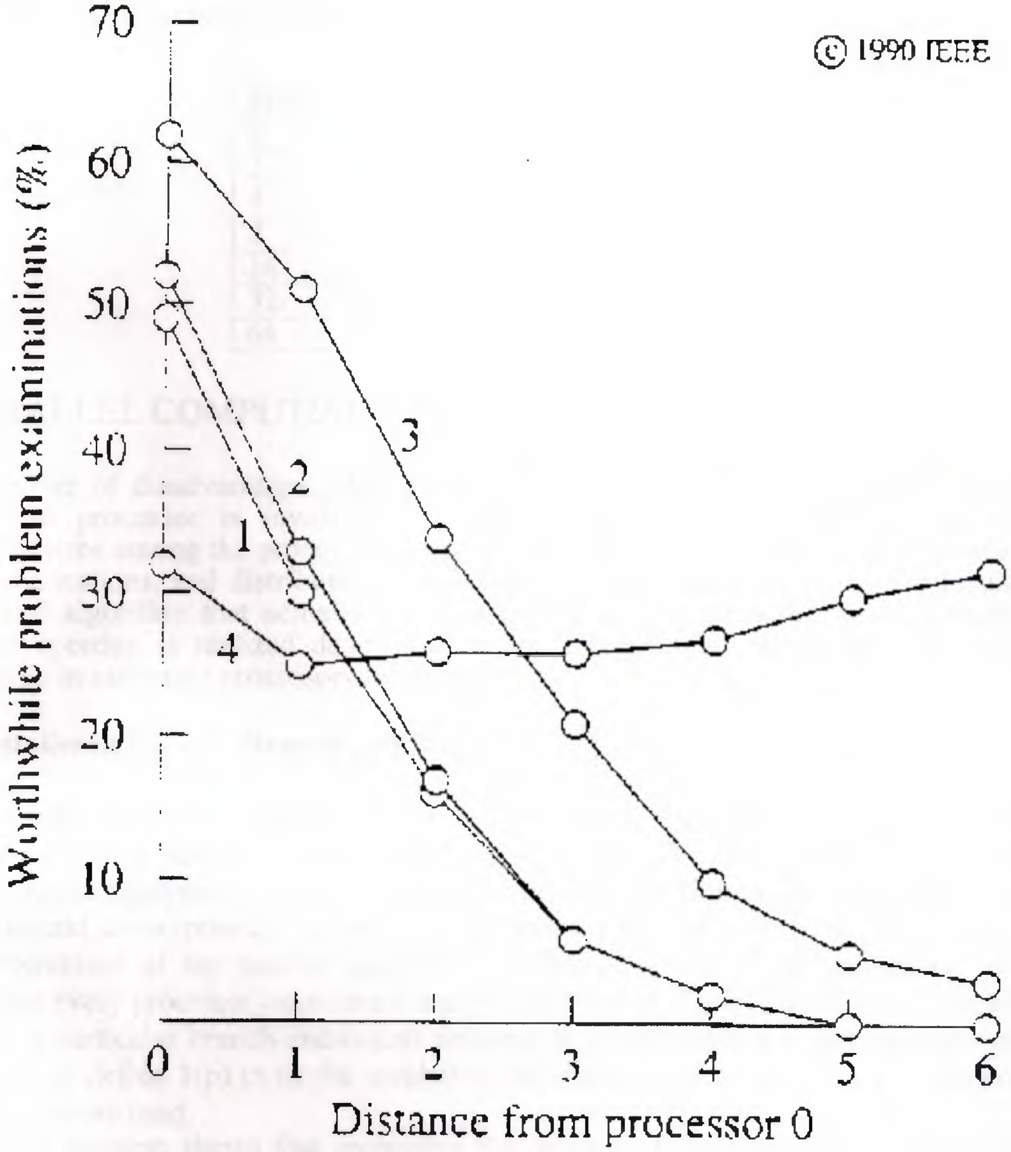
Processors		Alg.1	Alg.2	Alg.3	Alg.4
1	Actual	1,00	1,00	1,00	1,00
	Predicted	1,00	1,00	1,00	1,00
2	Actual	1,88	1,87	1,89	1,93
	Predicted	1,90	1,90	1,96	1,96
4	Actual	3,52	3,43	3,73	3,65
	Predicted	3,58	3,58	3,85	3,85
8	Actual	5,67	5,43	6,59	6,50
	Predicted	5,76	5,69	7,09	7,09
16	Actual	7,39	6,86	10,20	9,12
	Predicted	7,58	7,00	11,33	10,51
32	Actual	8,07	7,96	12,92	11,50
	Predicted	7,89	7,89	15,20	12,91
64	Actual	7,01	7,34	12,57	12,99
	Predicted	9,05	6,27	13,74	13,10

Actual and predicted speedups of asynchonrous branch-and-bound algorithms 1, 2, 3 and 4 on nCUBE 3200. Values represent averages over ten 30-vertex instances of the traveling salesperson problem with asymmetrical integer distances.

Problem from the priority queue and sends it. Algorithm 4 puts both newly created subproblems on its priority queue. then deletes the best problem from the priority queue and sends it.

The upper entries of Fig. 13-1 ‘ indicate the speedup measured on the nCUBE 3200 for each of these four algorithms. The lower, italicized values are the speedups predicted by Quinn's model. Despite the simplifying assumptions, most notably the assumption that all subproblem decompositions require the same amount of time, the model is a reasonably accurate predictor of speedup.

For the solution of a 30-vertex traveling salesperson problem on 64 processors. execution time is dominated by the time needed to examine all worthwhile subproblems. Hence the difference in speedup among the four asynchronous algorithms is a reflection of how well they kept processors busy doing useful work. plots the percentage of worthwhile subproblem examinations as a function of distance from Processor 0, the processor given the initial problem. as the algorithms execute on a six-dimensional hypercube. The “distance” between two processors is the length of the shortest path in the hypercube linking them. The significant differences in the curves illustrate how a simple change in the subproblem distribution heuristic can have a dramatic effect on the efficiency of the parallel algorithm, by increasing or decreasing the percentage of time various processors spend examining useful subproblems.



© 1990 IEEE

Because the execution times of the asynchronous algorithms on the 30-vertex traveling salesperson problem are dominated by the time needed to solve all worthwhile subproblems, Fig. 13-12 does not validate expression 13.2. To illustrate the precision of this part of the model, we present the performance of Algorithm 4 solving 10 instances of the 20-vertex traveling salesperson problem. For this smaller-sized problem the speedup of Algorithm 4 is constrained by the time needed to traverse the critical path because: (1) the state space tree has relatively few worthwhile subproblems and (2) subproblems leading to the solution are frequently transferred from one processor to another. contrasts the actual and predicted speedups of this algorithm.

We can use the model to predict the performance of these algorithms on other multicomputers because changing the value of λ does not affect the values of the other parameters.

To summarize, multicomputer implementations of parallel branch-and-bound algorithms that keep unexamined subproblems in a single priority queue have a

Actual and predicted speedups for Algorithm 4 solving 20-vertex traveling salesperson problem. Values represent averages over ten problem instances.

Processors	Actual	Predicted
1	1,00	1,00
2	1,63	1,87
4	1,92	2,00
8	1,90	2,01
16	2,04	2,01
32	2,26	2,01
64	2,21	2,04

PARALLEL COMPUTING: THEORY AND PRACTICE

Number of disadvantages. One processor must have a disproportionately large memory, and that processor is involved in every communication. Distributing the unexamined subproblems among the processors balances the memory requirements. reduces the number of communications, and distributes the messages over the network, which can result in a more practical algorithm that actually achieves higher speedup. Whether or not the potential for higher speedup is realized depends upon the effectiveness of the subproblem-distribution heuristic in assigning processors useful work.

Anomalies in Parallel Branch and Bound

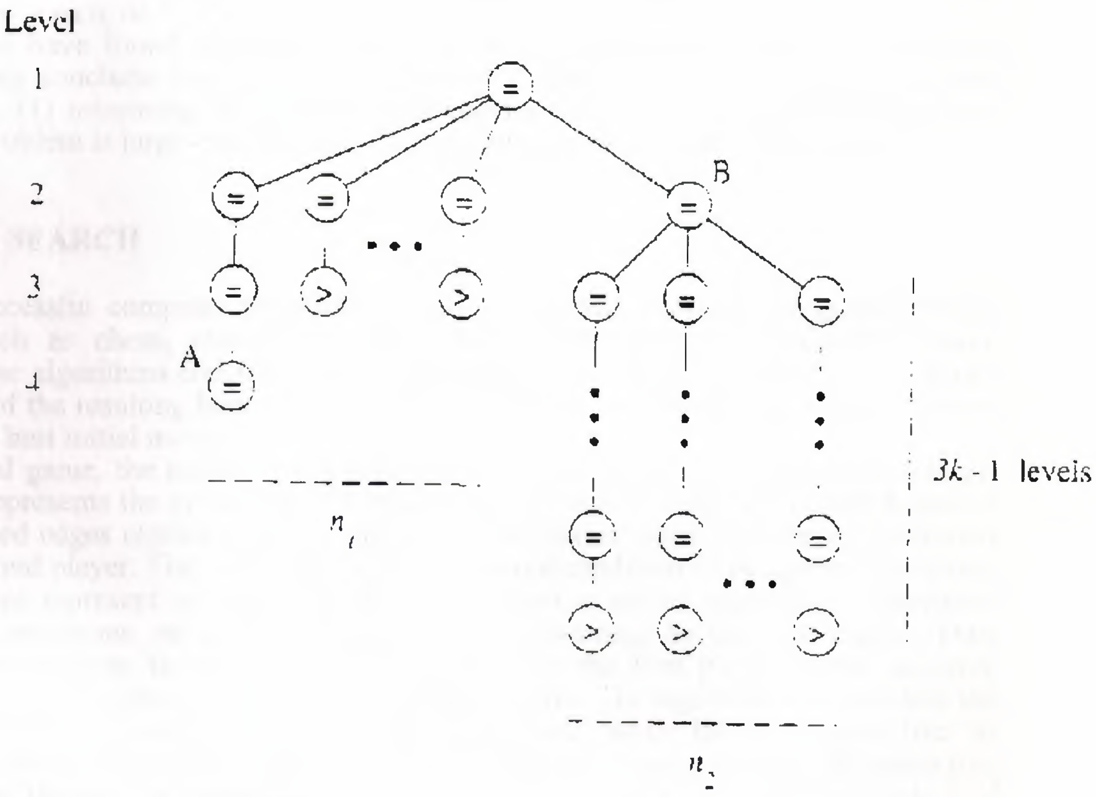
In this section we present Lai and Sahni's (1984) analysis of the speedups theoretically achievable by a parallel branch-and-bound algorithm. We must make a *few* assumptions in order for the analysis to be manageable. Assume that the time needed to examine any node in the tree and decompose it is constant for all nodes in the state space tree. Furthermore, assume that execution of the parallel algorithm consists of a number of 'iterations.' During each iteration every processor examines a unique subproblem if one is available and decomposes it. Given a particular branch-and-bound problem to be solved and a particular lower bounding function g. define 1(p) to be the number of iterations required to find a solution node when p processors are used.

The first theorem shows that increasing the number of processors can actually increase the number of iterations required to find a solution.

Theorem 13.1. Given *n~* < n~ and *k* > 0, there exists a state space tree such that *kI(nl) <1(n2)*. (See Lai and Sahni [1984].)

Proof Consider the state space tree shown in All nodes labeled have the same lower bound, which happens to be the value of the least-cost answer node (node A(. Nodes labeled">" have a lower bound greater than the value of the least-cost answer node. When ii$_l$ processors conduct the search, on the first iteration the root node is expanded into n1+1 children nodes. The second iteration consists of expanding the *n~* leftmost nodes at level 2 into *n* nodes at level 3. Of the nodes at level 3, n1—1 of them cannot lead to the solution and are discarded. On iteration 3 the remaining node at level 3 and node B are expanded. Since the node at level 3 leads to the solution node, the algorithm terminates. Hence I*(nl)* = 3.

When n2 processors conduct the search, the first iteration is the same: The root node is expanded into *nl* + 1 children nodes. On the second iteration, however, all *nl* + 1 nodes at level 2 are expanded, yielding n_l +n2 nodes at level 3. Since only n2 nodes at level 3 can be expanded on iteration 3, it could happen that the *n2* rightmost nodes would be the nodes chosen. If we assume the processors expanded the n2 rightmost nodes at level *3, n2* nodes at level 4 would be created, and iterations 4, 5, 6, . . . , 3k could be devoted to a wild-goose chase, expanding nodes down the right part of the tree. The solution node A would be expanded on iteration 3k + 1. Hence $I(n2)$ = 3k + 1. Combining the two results yields $k\ I(n_1)$ = 3k < 3k + 1 = $I(n_2)$.



Because many nodes have a lower bound equal to the value of the least-cost answer node-f*—we see the anomaly described in the Theorem 1. What would happen if $g(x) = f\sim$, whenever x is not a solution node?

Definition .1. A node x is critical if $g(x) < f$.

Theorem 2. If g(x) = f^* whenever x is not a solution node, then $I(1) > I(n)$ for all n> 1. (See Lai and Sahni (1984).)

Proof By the definition of the best-first branch-and-bound heuristic, only critical nodes and least-cost answer nodes can be expanded. In addition, every critical node must be expanded before any least-cost answer node is expanded. Hence if the number of critical nodes is *in.* then $I(1) = m$. When $n > 1$, at least one of the nodes expanded each iteration must be a critical node (Prob. *13-5).* Hence a least-cost answer node must be examined no later than iteration *m.* Thus if the number of critical nodes is *in,* then $I(n) < m$. Therefore $I(1) > I(n)$ for all $n > 1$.

The following theorem proves that increasing the number of processors can actually cause a disproportionate decrease in the number of iterations required to find a solution node.

Theorem 13.3. Given n1 < n2 and $k > n2/n1$, then there exists a state space tree such that $I(n_1/I(n2) > k > n2/n1$. (See Lai and Sahni [1984].)

Proof This is left to the reader as Prob. 13-6.

Theorem 13.4. If $g(x) = f^*$ whenever x is not a least-cost answer node, then $I(1)$ / I

$(n) < n$ for $n > 1$. (See Lai and Sahni [1984].)

Proof Let m be the number of critical nodes. Then $1(1) = m$ (Theorem 13.2). All critical nodes must be expanded before the parallel branch-and-bound algorithm can terminate (Prob. 13-7). Hence $1(n) > rn/n$, or $1(1)/1(n) < n$.

Lai and Sahni have found anomalous behavior in some instances of the 0-1 knapsack problem, but they conclude that anomalous behavior is rarely encoun tered in practice and that, in general. (1) increasing the number of processors will not increase execution time (assuming the problem is large enough), and (2) superlinear speedup cannot be expected.

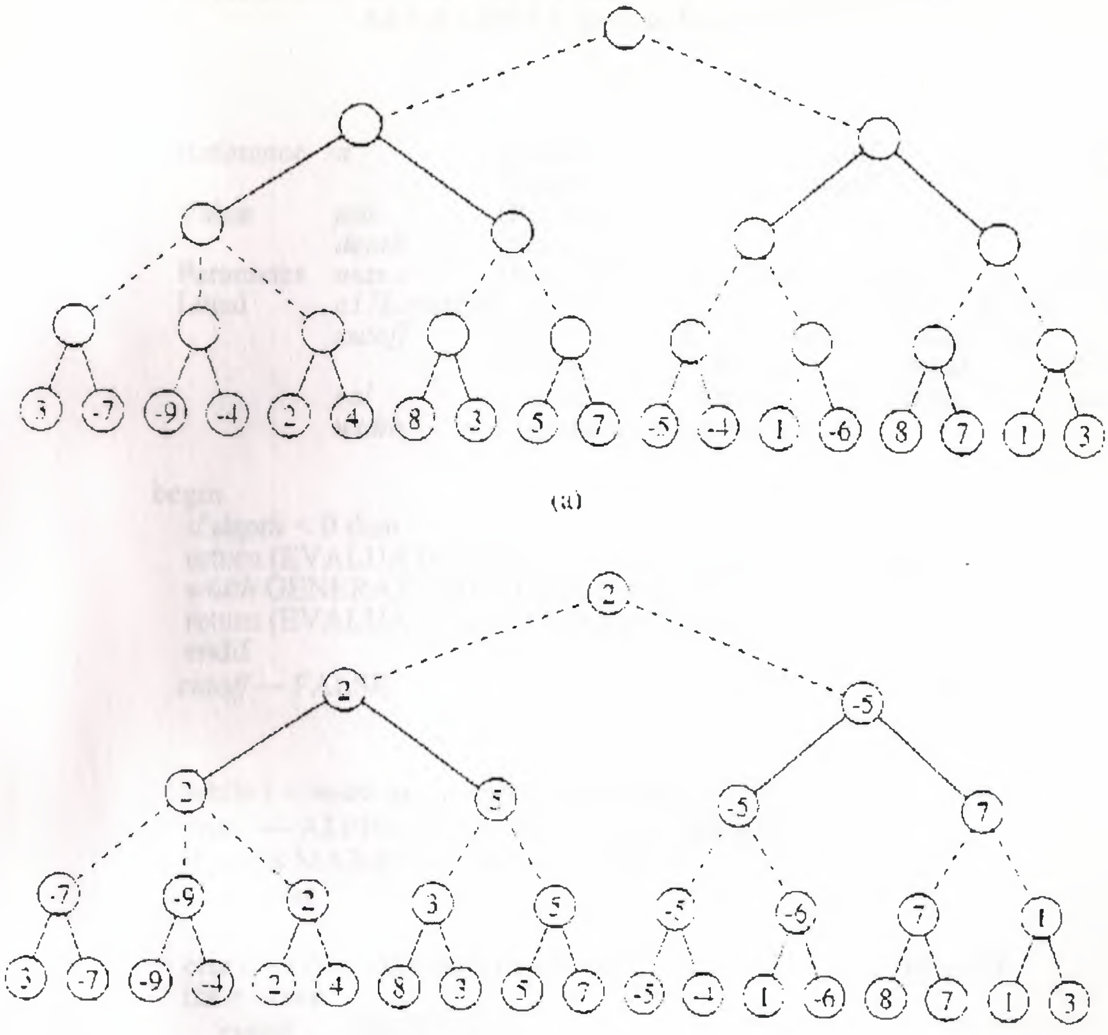
ALPHA-BETA SEARCH

The most successful computer programs to play two-person zero-sum of games perfect information, such as chess, checkers, and go, have been based on exhaustive search algorithms. These algorithms consider series of possible moves and countermoves , evaluate the desirability of the resulting board position then work their way back up the tree of moves to determine the best initial move.

Given a trivial game, the minimax algorithm can be used to determine the best strategy. Figure 13-16a represents the game tree of a hypothetical gain with rules left unstated, played for money. Dotted edges represent moves ma hr the first player: solid lines represent moves made by the second player. The root of the tree is the initial condition of the game. The leaves of this game tree represent outcomes of the game. Interior nodes represent intermediate conditions. The outcomes are always put in terms of advantage to the first player. Thus positive numbers indicate the amount of money won by the first player, while negative numbers indicate the amount of money lost by the first player. The algorithm assumes that the second player tries to minimize the gain of the first player, while the first player tries to maximize his or her own gain, hence the name of the algorithm. Figure 13-l6b is the same tree with the values of the interior nodes filled in. The value of this game to the first player is 2. If the first player plays the minimax strategy he or she is guaranteed to win at least two dollars.

Stockman (1979) has pointed out that a game tree is an example of an AND/OR tree. The AND nodes represent positions where it is the second player's turn to move. The OR nodes represent positions where it is the first player's turn to move.

Nontrivial games such as chess have game trees that are far too complicated to be evaluated exactly. For example, de Groot has estimated that there may *be* 3884 positions in a chess game tree (de Groot 1965). Thus current chess-playing programs examine moves and countermoves only to a certain depth, then, that point, estimate the value of the board position to the first player. Of course, evaluation functions are unreliable. If a perfect evaluation function existed, the need for searching would be eliminated (Prob. 13-9). As we have seen, all



(a)

(b)

possible moves and countermoves from a position p to some predetermined lookahead horizon can be represented by a game tree. The minimax value of the game tree can be found by applying the evaluation function to the leaves of the tree (the terminal nodes), then working backward up the tree. If it is the second player's move at a particular nonterminal node in the game tree, the value assigned is the minimum over all its children nodes. If it is the first player's move, the value assigned is the maximum over all its children nodes.

Given a game tree in which every position has *b* legal moves, it is easy to see that a minimax search of the game tree to depth *d* examines *bd* leaves.

It is generally true that the deeper the search, the better the quality of play. That is why alpha-beta pruning has proven to be valuable. Alpha-beta pruning, a form of branch-and-bound algorithm, avoids searching subtrees whose evaluation cannot influence the outcome of the search, i.e., cannot change the choice of move. Hence it allows a deeper search in the same amount of time.

The alpha-beta algorithm, displayed in Fig. 13-17, is called with four arguments: *pos,* the current condition of the game; a and B, the range of values over which the search is to be made: and *depth,* the depth of the search that I

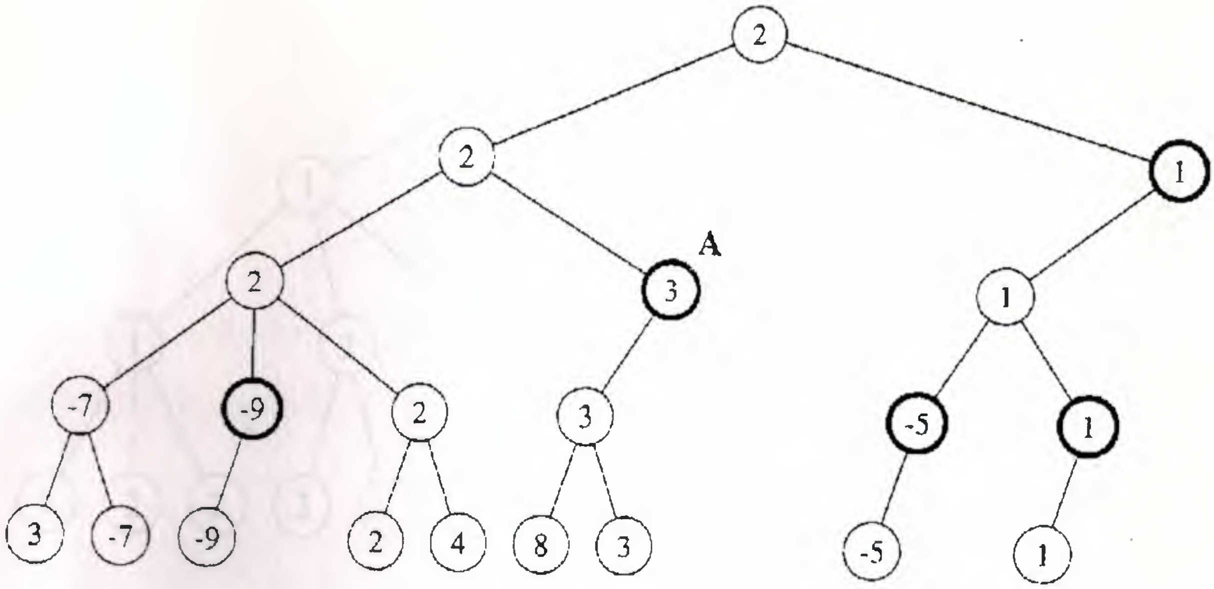
ALPHA.BETA ***(pos, a,* B, *depth)***

```
Reference   a           {Lower cutoff value)
                        (Upper cutoff value)
Value       pos         (Position)
            depth       (Search depth)
Parameter   max.c       (Maximum possible number of moves}
Local       c17L.rnax.c] (Children of pos in game tree)
            cutoff      (Set to TRUE when okay to prune)
                        (Iterates through legal moves)
            val         (Value returned from search)
            width       (Number of legal moves)

begin
   if depth < 0 then
   return (EVALUATE (pos)) (Evaluate terminal node) endif
   width GENERATE.MOVES(pos) if width = 0 then
   return (EVALUATE (pos)) (No legal moves)
   endif
  cutoff ~- FALSE

   while i < width and cutoff = FALSE do
    vat ~— ALPHA.BETA (clIi], a, ~, depth-i)
   if pos is MAX-NODE and tat > a then
      a — tat

   else if ~OS is MIN-NODE and tat < ~ then ~ *— tat endif endif
   ifa > ~then
      cutoff    TRUE
   endif
   j *—i-r-l
   endwhile
   if pos is MAX-NODE then return a else return ~
   endif
end
```

Sequential alpha-beta algorithm.

to be made. The function returns the minimax value of the position *pos.* The original board position is a MAX-NODE. Every child of a MAX-NODE is a MIN-NODE. Every child of a MIN-NODE is a MAX-NODE.

To illustrate the workings of the alpha-beta algorithm, consider the game tree in This tree represents the same game as that in



PARALLEL COMPUTING: THEORY AND PRACTICE

Except that nodes not examined by the alpha-beta algorithm are not included. When the algorithm begins execution, a —cc and B cc. The algorithm traverses the nodes of the tree in preorder; the values of a and ~ converge as the search progresses.

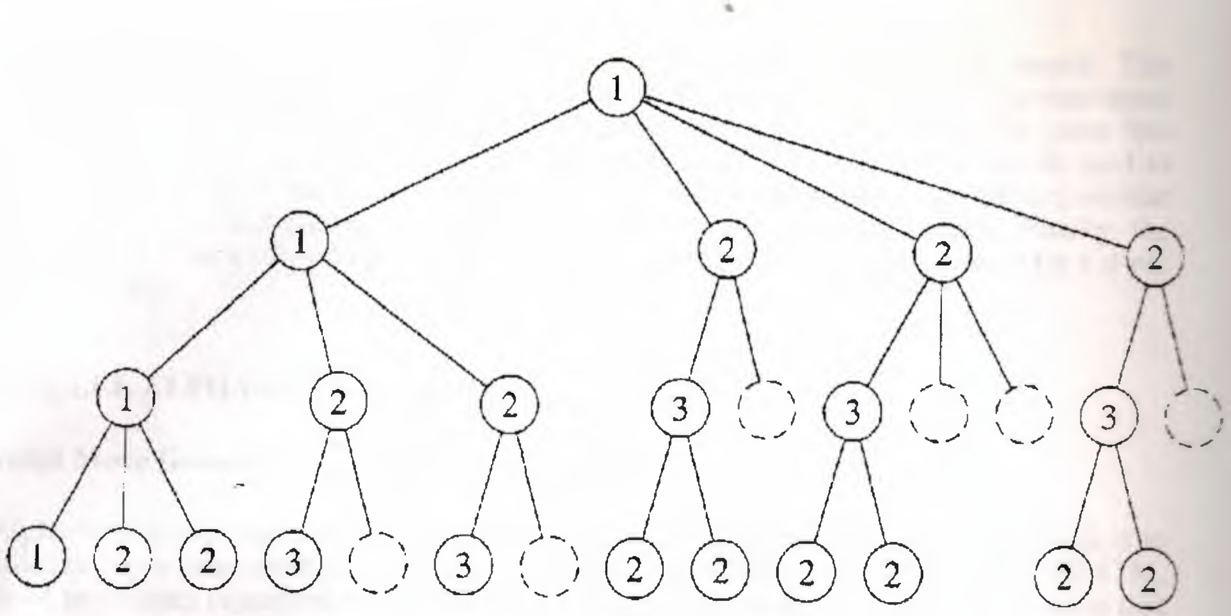
The nodes drawn in heavy lines in Fig. 13-18 represent places where pruning occurs. To explore the conditions under which pruning happens let us consider an arbitrary interior node in the search tree. When the search reaches this node, we know that some choice of moves that has already been considered leads to a value of at least a for the player moving first. We also know that correct play on the part of the opponent will ensure that the first player cannot get a~ value more than ~. Hence a and ~ define a window for the search.

If the interior node *pos* is a MAX-NODE, then it is the first player's move. If *vat,* the value of the game tree searched from node *pos* is greater than a, then a is changed to *vat,* a better line of play has been found for player one.

Analogously, if the interior node *pos* is a MIN-NODE, then it is the second player's move. If *vat,* the value of the game tree searched from node *pos* is less than ~, then ~ is changed to *vat;* a better line of play has been found for player two.

However, if at any time the value of a exceeds the value of B, there is no need to search further. It is in the best interests of one of the players to block the line of play leading to node *pos.*

For example, consider the node labeled A in . The value returned from the search of the first child of A is 3, which is greater than 2, the value fl. It is not in the second player's interest to allow play to reach this position, since there is another line of play guaranteeing a value no higher than 2. Hence there is no point in continuing the search from this game position.



To what extent can alpha-beta pruning reduce the number of leaf nodes that must be examined? The algorithm does the most pruning on a perfectly ordered game tree, that is, a game tree in which the best move from each position is always searched first (see Fig. 13-19). Assuming a perfectly ordered game tree with a search depth of *d* and uniform branching factor *b,* Slagle and Dixon (1969) have shown that the number of leaf nodes examined by the alpha-beta algorithm is

Opt(b, d) = *b[d/*^{2}*1* ± bLd/^{2}J —In other words, in the best case it is

possible for the alpha-beta algorithm to
examine no more than approximately twice the square root of the number of nodes searched by the minimax algorithm.

Definition 13.2. The effective branching factor of an algorithm searching a game tree of depth *d* is the dth root of the number of leaf nodes evaluated by the algorithm.

Casting Slagle and Dixon's result in terms of this definition, an alpha-beta search reduces the effective branching factor from *b* to *-.115* when searching a perfectly ordered game tree.

Of course, a perfectly ordered search is not possible in practice. However, experimental evidence indicates that sequential alpha-beta algorithms often search no more than 50 percent more nodes than would be searched if the tree were perfectly ordered. Hence in practice the alpha-beta search algorithm exhibits much higher performance than minimax.

Two common enhancements to the alpha-beta search algorithm are aspiration search and iterative deepening. Aspiration search makes an estimate of the value *v* of the board position at the root of the game tree, figures the probable error *e* of that estimate, then calls the alpha-beta algorithm figuring the probable error *e* of that estimate, then calls the alpha-beta algorithm With the initial window *(v — e, v — e).* If the value of the game tree does indeed fall within this window of values, then the search will complete sooner than if the algorithm had been called with the initial window (—cc, cc). If the value of the game tree is less than *v — e,* then the search will return the value *v — e,* and the algorithm must be called again with another window, such as (—cc, *v — e).* Similarlv, if the value of the game tree is greater than *v + e,* then the search returns the value *v + e,* and another search will have to be done with a modified initial window, such as *(v + e,* cc).

Another variant on the standard alpha-beta algorithm is called iterative deepening. Each level of a game tree is called a ply and corresponds :o the moves of one of the players.

Iterative deepening is the use of a *(d—* 1)-ply search to prepare for a *d-ply* search. This technique has three advantages (Marsland and Campbell 1982). First, it allows the time spent in a search to be controlled. The search can continue deeper and deeper into the game tree until the time allotted has expired. Second, results of the *(d* — 1)-ply search can be used to improve the ordering of the nodes during the *d-ply* search, making the node ordering similar to perfect ordering, and allowing the alpha-beta search n execute more quickly. Finally, the value returned from a *(d* — 1)-ply search can be used as the center of the window for a d-ply aspiration search.

PARALLEL ALPHA-BETA SEARCH

Parallel Move Generation and Position Evaluation

Alpha-beta search has a number of opportunities for parallel execution One approach is to parallelize move generation and position evaluation. The custom chess machine HITECHTM, with 64 processors organized as an 8 x 8 array has taken this route. However, the speedup that can be achieved with this aproach is limited by the parallelism inherent in these activities. Further improvements in speedup lie in parallelizing the search process.

Parallel Aspiration Search

Another straightforward parallelization of the alpha-beta algorithm is tone by performing an aspiration search in parallel. If three processors are z~ailable, then each processor can be assigned one of the windows (—cc,: —*(v—e, v+e),* and *(v+e,* cc). Ideally the processor searching *(v—e, v±~* will succeed, but all three processors will finish no later than a single processor searching the window (—cc, cc). More processors can be accommodand by creating more windows with smaller ranges. Baudet (1978a, 1978b) explored parallel aspiration on the Cm* NUMA multiprocessor.

PARALLEL COMPUTING: THEORY AND PRACTICE

Work on parallel aspiration for the game of chess has led to two conclusions. First, the maximum expected speedup is typically five or six, regardless of the number of available processors. This is because *Opt(b, d)* is a lower bound on the cost of alpha-beta search, even when both a and ~ are initially set equal to the value eventually returned from the search. Second, parallel aspiration search can sometimes lead to superlinear speedup when two or three processors are being used.

Parallel Subtree Evaluation

Many believe that significant speedups can only be achieved by allowing processors to examine independent subtrees in parallel. There are two important overheads to be considered. Search overhead refers to the increase in the number of nodes that must be examined owing to the introduction of parallelism. Communication overhead refers to the time spent coordinating the processes performing the searching. Search overhead can be reduced at the expense of communication overhead by keeping every processor aware of the current search window. Communication overhead can be reduced at the expense of search overhead by allowing processors to work with outdated search windows.

For example. consider this simple method of performing alpha-beta search in parallel. Split the game tree at the root, and give every processor an equal share of the subtrees. Let every processor perform an alpha-beta search on its subtrees. Each processor begins with the search window (—~. ~), and no processor ever notifies other processors of the changes in its search window. Clearly this algorithm minimizes communication overhead. What is the speedup achievable by this method?

Theorem 5. Given a perfectly ordered uniform game tree of depth *d* and branching factor *b*. the number of node examinations performed by alpha-beta search in the first branch's subtree is(See Hyatt et al. [1989].) *Proof.* Slagle and Dixon (1969) showed that the minimum number of nodes examined from a type 1 node of depth *d* is *br(dl)/*^{2}*1* ±bL(d^{1})/^{2}1 — 1. In a perfectly ordered game tree, the first child of a type 1 node is also a type 1 node, so we simply replace *d* with *d* — 1 in their expression.

Theorem.5 demonstrates that the examination of the first branch of a perfectly ordered game tree takes a disproportionate share of the computation time. For example, consider a 10—ply search of a perfectly ordered tree that has a branching factor of 38 (such as a chess game tree). The minimum number of

COMBINATORIAL SEARCH

Node examinations is 158,470.335. The minimum number of node examinations in the first branch is 81,320,303. By Amdahl's law it is clear that if only one processor is responsible for searching the first move's subtree, speedup will be less than two.

In addition, because every processor's search must begin with —oc and ~ as the values for a and ~. respectively, the parallel algorithm will not prune as many subtrees as the sequential algorithm. A complete elimination of communication overhead creates significant search overhead.

Let's look at the other extreme. What must be done to eliminate search overhead completely? We will make the assumption that the game tree is perfectly ordered. Look at Fig. 13-19. If we want to eliminate search overhead, we must ensure the parallel algorithm prunes the same nodes as the sequential algorithm. First consider searching the subtree of a type 1 node. The first child is a type I node; the remaining children are type 2 nodes. Searching subtrees rooted by type 2 nodes requires up-to-date values of a and ~ in order to prune all but the first children of the type 2 nodes. To get up-to-date values, the search of the subtrees rooted by type 2 nodes cannot begin until the search of the subtree rooted by the type I node has finished, returning a and ~. However, once the values of a and ~ are known, all type 2 nodes may be searched in parallel without processor interaction.

Next, let's look at the search of a subtree of a type 2 node. Since all but the first child are pruned. there is no parallelism to be exploited.

Finally, consider the search of a subtree of a type 3 node. All its children are type 2 nodes, and these nodes may be searched in parallel without processor interaction.

In practice, search trees are not perfectly ordered, but this exercise has demonstrated that a parallel alpha-beta algorithm can significantly reduce search overhead by delaying the search of some subtrees until more accurate bounds information is available.

Distributed Tree Search

Ferguson and Korf (1988) have developed a parallel tree searching algorithm called Distributed Tree Search (DTS), which, when evaluating game trees, has achieved good speedups. Although the DTS algorithm is suitable for solving a variety of tree search problems, we will describe its use as a tool to perform parallel alpha-beta search.

The DTS algorithm executes by assigning processes to nodes of the search tree. Each process controls one or more physical processors. When the algorithm begins execution, a single process, called the root process, is assigned to the root node of the search tree. It controls the entire set of physical processors performing the search.

When a process is assigned to a nonterminal node, it generates the children of that node by evaluating the legal moves. The process assigns processors to

The children nodes based upon the processor allocation strategy. For example, if the search is using a breadth-first processor allocation scheme, one processor is allocated to each child node until there are no more processors to allocate. At this point a new process is created for each child node that is allocated at least one processor. The parent process suspends operation until it receives a message from another process.

When a process is assigned to a terminal node, it returns the value of that node and its set of allocated processors to the parent, then terminates.

The first child process to complete the search of its subtree sends a message with its values of a and ~ to the parent. It returns its set of processors to the parent and terminates. The parent process wakes up when it receives the message from its child. It reallocates the freed processors to one or more of its active child processes. It may also send one or more of its child processes new values of a and ~. The reallocation of processors from quicker processes to slower processes produces efficient load balancing. Notice that in this scheme a child process may be awakened by its parent, which is passing along additional processors. After reallocating processors. parent processes suspend operation until they receive another message. When all child processes have terminated, the parent process returns a. ~. and the set of processors to the parent and terminates. When the root process terminates, the algorithm has completed.

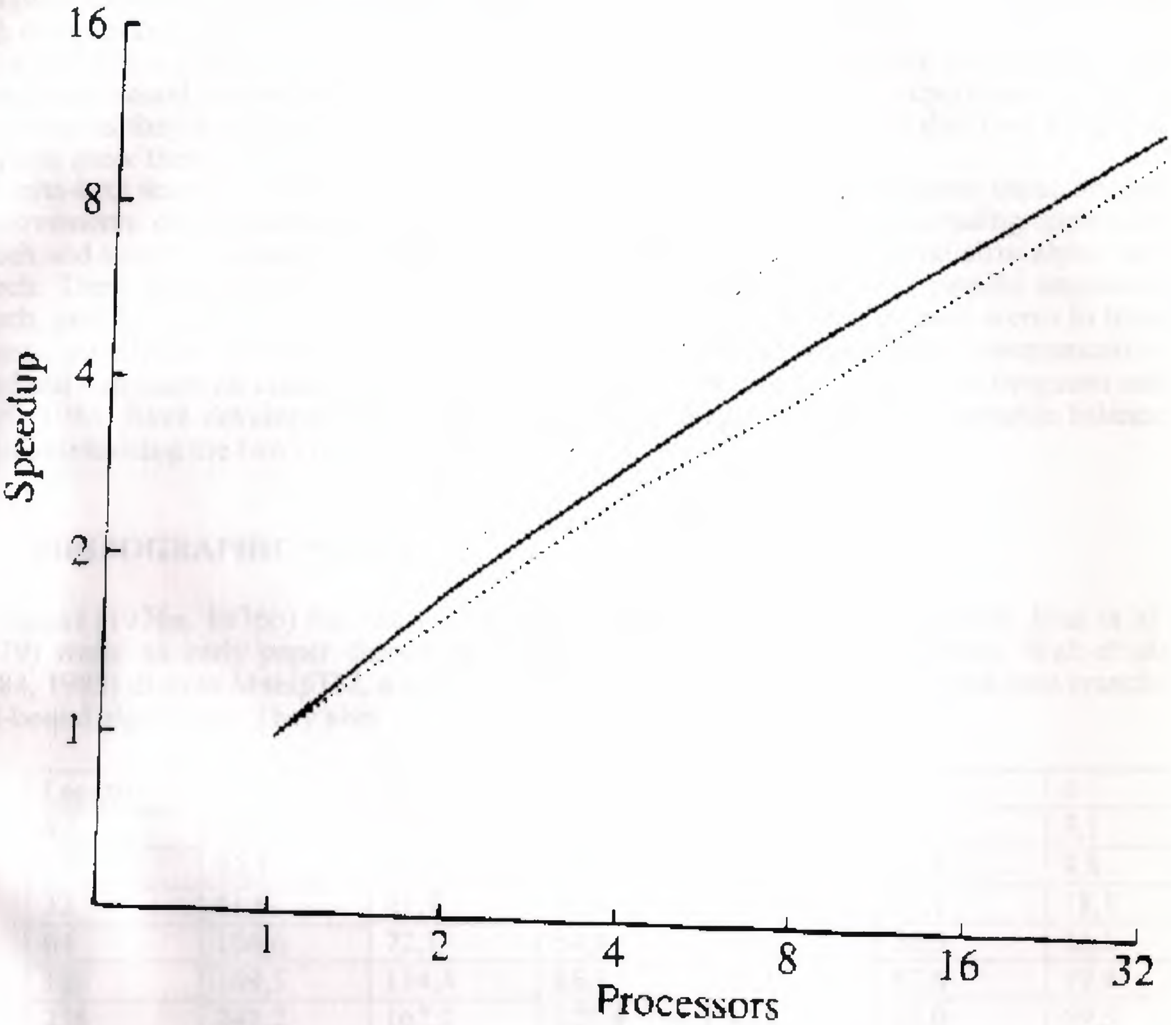
Three implementation details improve the performance of the DTS algorithm. First, every blocked process should share a physical processor with one of its child processes. In this way all processors stay busy. Second, when a blocked parent process is awakened, it should have higher priority for execution than processes corresponding to nodes deeper in the search tree. Third, when the search reaches a point where there is only a single processor allocated to a node, the process controlling the processor should execute the standard sequential alpha-beta search algorithm.

Proof The execution time of the sequential algorithm is proportional to the number of leaf nodes it evaluates, or *0 ((bx)~~) — 0 (b" ').* The DTS algorithm with breadth-first allocation distributes processors evenly among the branches of the search tree, until there is one processor per node. This occurs at depth O(logb *p).* The time complexity of this part of the search is also O(log~ *p),* since allocations at the same level in the tree occur in parallel. Once the search has reached a point where there is one processor per node, every processor performs the sequential alpha-beta algorithm on the remaining subtree of depth *0(d —* logs *p).* The time needed for these searches is *O(bx(d—ogb P)),* since the effective branching factor is *bx.* Propogating values back to the root has time complexity O(log~ *p).* The overall time complexity of the DTS algorithm is O(log~ *p+b'~*^{4}*'O5bP).* As the depth *d* grows, the second term dominates, and the parallel time complexity is *O(bxdiogh P)).* The speedup is the sequential time complexity divided by the parallel time complexity, or

To test the DTS algorithm, Ferguson and Korf (1988) have implemented the game of Othello. Their node-ordering function results in an effective branching factor of about b^{66}. The program implements parallel alpha-beta search using the DTS algorithm. Ferguson and Korf executed the algorithm on 40 midgame positions using 1, 2, 4, 8, 16, and 32 nodes of an Intel iPSC hypercube multicomputer. They estimated the speedup achieved by the program by dividing the number of node evaluations performed by the sequential algorithm by the number of node evaluations performed per processor by the parallel algorithm. For example, they estimate an average speedup of 10 for 32 processors. Figure 13-20 plots the speedup achieved by their algorithm.

Ferguson and Korf have implemented another processor allocation strategy, called bound-and-branch, which corresponds closely to the algorithm described at the end of the last subsection. When the search reaches a type I node, all processors are allocated to the leftmost child. After the search returns with cutoff bounds from the subtree rooted by the leftmost child, the processors are assigned to the remaining children nodes in a breadth-first manner. When the search reaches a node having type 2 or 3, cutoff bounds already exist, and the processors are assigned in breadth-first fashion.

Ferguson and Korf have empirically determined that the bound-and-branch strategy achieves higher speedup than the breadth-first allocation strategy, even when the node ordering is not perfect. They have implemented a version of the Othello program that uses iterative deepening and the bound-and-branch



PARALLEL COMPUTING: THEORY AND PRACTICE

processor allocation strategy. The actual speedup achieved by the program is 12 on 32 processors.

SUMMARY

One way to differentiate between combinatorial search problems is to categorize them by the kind of state space tree they traverse. Divide-and-conquer algorithms traverse AND trees: the solution to a problem or subproblem is found only when the solution to all its children is found. Branch-and-bound algorithms traverse OR trees: the solution to a problem or subproblem can be found by solving any of its children. Game trees contain both AND nonterminal nodes and OR nonterminal nodes.

Parallel combinatorial search algorithms for all these trees have been proposed. The speedup achievable through the parallel search of an AND tree is limited by propagation and combining overhead.

Mohan (1983) has implemented programs to solve the traveling salesperson problem on a NUMA multiprocessor. Quinn (1990) has implemented programs to solve the same problem on hypercube multicomputers. Their work demonstrates the potential for implementing branch-and-bound algorithms on MIMD computers. The fundamental problem faced by designers of parallel branch-and-bound algorithms is keeping the efficiency of the processors high by focusing the search on the nodes the sequential algorithm examines.

Lai and Sahni (1984) have given examples of state space trees for which parallel best-first branch-and-bound algorithms can show anomalous behavior, such as superlinear speedup. Experiments they have performed with the simulated parallel solution of the 0—1 knapsack problem show that anomalous behavior can really occur, albeit rarely.

Alpha-beta search has proven to be an efficient method for evaluating game trees. Several improvements on the standard alpha-beta search have been invented, including aspiration search and iterative deepening. Several methods have been proposed to parallelize alpha-beta search. These methods include parallel move generation and evaluation, parallel aspiration search, and the parallel search of independent subtrees. Only the third method seems to have enough parallelism to scale to massively parallel machines. Minimizing communication overhead can cause an unacceptable amount of search overhead, and vice versa. Ferguson and Korf (1988) have developed the bound-and-branch strategy to keep an acceptable balance while minimizing the two kinds of overhead.

BIBLIOGRAPHIC NOTES

Ibaraki (1976a, 1976b) has analyzed sequential branch-and-bound algorithms. Imai et al. (1979) wrote an early paper describing a parallel branch-and-bound algorithm. Wah et al. (1984, 1985) discuss ManipTM, a computer specifically designed to execute best-first branch-and-bound algorithms. They also

Length	1	2	3	4	5	6
8	17,2	11,7	8,9	7,1	5,9	5,1
16	33,1	22,4	16,9	13,7	11,4	9,8
32	61,6	41,7	31,4	25,2	21,1	18,1
64	106,6	72,1	54,4	43,7	36,5	31,1
128	169,5	114,4	86,1	69,1	57,6	49,4
256	241,2	162,2	121,4	97,1	81,0	69,5
512	304,8	203,4	152,4	121,4	101,6	87,1
1024	351,1	233,4	174,9	139,8	116,4	99,8
2048	380,8	252,2	188,8	150,8	125,6	107,6

Load Balancing

The goal of *load balancing* is to keep processor nodes busy and have them finish roughly at the same time. We say a program is *balanced* if its computation is equally distributed across all processors. Valuable processor cycles are wasted if some nodes have to wait on others to finish. More important, the greatest speedup is possible only when all processors are busy, all of the time.

An application should be analyzed to make sure it is balanced. If the *work load* is known beforehand, it is possible to statically determine a balanced distribution of work at compile time. On the other hand, if the work load is not known beforehand, the parallel processors must dynamically adjust the load. Static techniques can be applied by the programmer, but dynamic techniques must be applied by either the operating system or the application software during program execution.

• Chapter 9 discusses a number of techniques for static and dynamic load balancing. We merely describe a few simple techniques for dynamic balancing here.

There are several heuristics for *dynamic load balancing.* In what follows, we show two variations of the same load balancing heuristic given in Ranka. In both versions, the load is balanced by averaging the load over processors that are directly connected. In heuristic Hi, a processor transmits its entire work load, including the necessary data, to its neighbor processor. In heuristic H2, however, a processor transmits only the amount of work that is in excess of the average work load.

It is left as an exercise for the reader to find the differences between Hi and H2 and to determine the cases in which each heuristic is better than the other (see problem 6).

PROGRAMMING HYPERCUBES

Hi

Load Balance HI ()

```
For(i=0:i<CubeSize :i++)

        SendMyLoad to neighbor processor along dimension i:

        Receive HisLoad from neighbor processor along dimension i.

           and append to MyLoad;
        Avg = (MyLoadSize + HisLoadSize) /2:

        if  (MyLoadSize > Avg) MyLoadSize = Avg:
           else if (HisLoadSize ) Avg) MyLoadSize += HisLoadSize — Avg:
```

Load Balance H2

```
For (i=0:i<CubeSize:I++)

        SendMyLoad to neighbor processor along dimension ::

        Receive HisLoad from neighbor processor along dimension i.
        Avg = (MyLoadSize ± HisLoadSize)          2:

        if (MyLoadSize > Avg)

        Send extra load (MvLoad Size — Avg) to neighbor processor along dimension. 0;

           My-LoadSize=Avg:

           else if IHisLoadSize > Avg

        Receive extra load (Avg — HisLoadSize from neighbor processor along dimension

           >MyLoadSize -~- HisLoadSize — Avg;
```

Overhead is always associated with dynamic load balancing; therefore, we should be careful when using this technique to balance the load. Before incorporating a load balancing scheme into an algorithm, one must weigh the potential reduction in time required to complete the work against the time required to balance the load. If it takes longer to balance the load than to complete the work, it is not practical to balance the load using this method. It might be even better to perform the algorithm without dynamic load balancing.

Load balancing also depends on the way the problem is distributed in the system. Distributing a problem among the nodes in a parallel computer can be done through either *domain decomposition* or *control decomposition.* In domain decomposition, the domain of the input data are partitioned and the partitions are assigned to different processors. In control decomposition, program tasks are divided and distributed among processors.

CONCLUSION

This project that I explanition of parallel distributed systems.The department applications of parallel and distributed systems has its traditional field of work in the complex areas of databases and information systems.A multitude of system and application projects has been carried out,constantly exploring new subject areas.

There are types a distributed processing systems and function distribution systems.List the main reasons for function distribution.

In the following discussion we will list a number of traditional application areas and note a few of the more unusual application.The idea is to stimulate the readers imagination not to list good application.

We describe the main characteristic of distributed systems,their classification end programming techniques.Examples demonstratethe application areas of distributed systems.

The osf distributed computing environment (DCE) is an industry standart, Vendor-neutral set of disributed computing technologies.It provides security services to protect and control access to data,name services that make it easy to find distributed resources,and a highly scalable model for organizing widely scattered users,services,and data.DCE runs on all major computing platforms and is designed to support distributed applications in heterogeneous hardware and software environments.DCE is a key technology in three of todays most important areas of computing security theworld wibe web and distributed system.

Horus a flexible group communications system.

Branch and bound algorithm scale up well,given proper coordination.The speed-up on a cluster of worksatations is comparable to the one arrived at when using tightly-coupled multicomputers like transputers or hypercubes albeit the variance observed over different runs is higher.Thus thinking about the distributed of such an algorithm is worth the effort,especially if processor time available anyhow would be left unused.Moreover,the results demonstrate that for this type of algorithms it can be economically sensible to use several cheap slow processors instead of one single processor configuration.However these benefits are not for free.Due to the fact that on a loosely-coupled system several applications are run at the same time by different users mostly in an interactive way,the control of a long-running application on such a multicomputer is more difficult than on a tightly-coupled system.Thus special techniques are necessary to avoid the disturbance of other applications and to ensure fault tolerance.Also programming of a distributed version of a branch and bound algorithm is considerably complex than implementing a sequential version.This is reflected not only in lines of code but also by the need for application programmers to cope with new programming problems such as dissemination of global information ,load –balancing ,distributed termination or deadlock prevention.Based on this evidence we argue that while it seems that from the viewpoint of performance the feasibility of using a cluster of workstations as a loosely-coupled multicomputer for running branc and bound algorithms is well demonstrated .

REFERENCE

1-Introduction to Parallel Processing.
(Bruno Codenotti)

2-Highly Parallel Computing.
(George S.Almast)

3-Parallel Processing in Cellular Arrays.
(Yakov Fet)

4-Parallel Algorithms Design &Analysis
(Pranay Chandhur)

5-Parallel Algorithms in Computational Science.
(Springer-Verlag)

6-An Introduction to Distributed&Parallel
Computing . (Joel M.Crichlow)

7-Distributed Systems Concepts&Design
(George Coulouris /Jean Dollimore/Jim
Kindbeg)

8-Distributed Computer Control Systems.
(G.Rodd/ K.D Muller)

9-Introduction to Computer Performance
Analysis with Mathematica.
(Dr.Arnold O.Allen)

10-Internet