Partitioning computations and parallel processing

S RAMANI and R CHANDRASEKAR

National Centre for Software Technology, Gulmohar Cross Road No. 9,
Juhu, Bombay 400 049, India

Abstract. Local Area Networks (LANs) provide for file transfers, electronic mail and for access to shared devices such as printers, tape drives and large disks. But LAN do not usually provide for pooling the power of their computer workstations to work concurrently on programs demanding large amounts of computing power. This paper discusses the issues involved in partitioning a few selected classes of problems of general interest for concurrent execution over a LAN of workstations. It also presents the conceptual framework for supervisory software, a Distributed Computing Executive, which can accomplish this, to implement a 'Computing Network' named CONE. The classes of problems discussed include the following: problems dealing with the physics of continua, optimization as well as artificial intelligence problems involving tree and graph searches and transaction processing problems.

Keywords. Distributed computing; local area networks; process structures; decomposition; partitioning; parallel architectures.

1. Computing networks

A number of frameworks have been proposed for building networks of computing elements, which we call computing networks here. There has been considerable work on such networks. Several architectural proposals have been investigated. In this paper, we describe an architecture which we feel would be useful in efficient concurrent execution of selected classes of problems.

Multiprocessors, which were very popular in an earlier era, are now giving way to networks of processors. Architectural considerations (bus bottlenecks etc.), software design methodology and production economics are in favour of a large number of similar computing elements being put together to make large computing machines.

It is useful to review systems that are available today, either as laboratory prototypes or as full-fledged commercial products. Many of them are shared-memory machines with tightly coupled processors connected by some bus architecture. Communication is typically through the global memory using shared variables. Some systems use loosely coupled processors with no global memory, where communication is carried out by message passing, much in the spirit of the Distributed Computing System of Farber *et al* (1973). Hybrid schemes are also possible, where there is a mixture of local and shared memory. The machines surveyed below are representative of systems

available now. For each machine, a brief description is presented, followed by appropriate comments on the system.

1.1 *The Cosmic Cube (Caltech)*

The Cosmic Cube (Seitz 1985) is a network of 64 Intel 8086 processors currently in use at Caltech. These processors are connected as nodes of a six-dimensional binary cube. Figure 1 shows a four-dimensional binary cube. The network offers point-to-point bidirectional 2-megabit serial links between any node and six others. Each node has its own operating system and software to take care of messaging and routing functions. The Cosmic Cube is a multiple instruction multiple data (MIMD) machine, which uses message passing for communication between concurrent processes. Each processor may handle one or more processes. A 'process structure' appropriate to an application can be created, with the nodes being processes, and the arcs connecting them representing communication links. The connectivity of the hypercube is such that any process structure desired would fit in as a sub-graph of the hyper-cube. There is no switching between processors and storage. One of the drawbacks in such a scheme is that no code or data sharing is possible. It is claimed that speeds of five to ten times that of a VAX 11-780 can be achieved on this machine on common scientific and engineering problems. The system allows scaling-up to hypercubes of higher dimensions.

Programs for this machine may be written using an abstract interconnection model, which is independent of the actual hardware implementation of the system. Since the 64-node machine does not provide for time-sharing, a separate 8-node system is used for software development. A fair amount of effort has gone into hardware building. Much of this will need to be repeated, for instance, when the proposed change occurs from Intel 8086 to Motorola 68020 processors.

1.2 *The NYU Ultracomputer*

The New York University (NYU) Ultracomputer (Gottlieb *et al* 1983; Edler *et al* 1985) is a shared-memory, MIMD, parallel machine. The ultracomputer uses a fetch-and-add operation to obtain the value of a variable and increment it in an indivisible manner. If many fetch-and-add operations simultaneously address a single variable, the effect of these operations is exactly what it would be if they were to occur in any arbitrary serial order. The final value taken is the appropriate total increment, but the intermediate

Figure 1. A 4-dimensional binary cube.

values taken depend on the order of these operations. Given this fetch-and-add operation, Gottlieb *et al* (1983) have shown that many algorithms can be performed in a totally parallel manner, without using any critical sections. The system uses a message switching network to connect N (where N is a power of 2) autonomous processing elements (PE) to a central shared memory composed of N memory modules (MM). This network is unique in its ability to queue conflicting request packets. In unbuffered systems, this situation, caused by multiple outputs to the same port, would lead to retransmissions and hence a loss in efficiency.

A design for a 4096-node machine, using 1990s technology, is also presented in Gottlieb *et al* (1983). A small 8-node prototype has been built already (Serlin 1985), based on Motorola 68010s. IBM's RP3 (described later) is partially based on this design.

1.3 ZMOB (*University of Maryland*)

ZMOB (Rieger *et al* 1981) is a ring of 256 Z-80s, linked by a special purpose high speed, high bandwidth, 'conveyer belt'. Each processor has a local memory of 64 K, and has its own local operating system kernel. Messaging is through the special hardware boards which interface processors to the conveyer belt. The messaging routines provide for point-to-point or broadcast messages to be communicated across the ring. Message destinations may be specified using the actual address or by choosing the pattern of addresses (send to all processes in subset A, say). The entire network is connected to a central host machine.

The conveyer belt is a promising innovation. However, the machine is now size-limited for two reasons: the processors used are Z80As, which by today's standards are relatively small machines, in terms of both speed and capability; second, the special purpose hardware for communication is too tightly linked to the first design to permit easy scaling up.

ZMOB ideas were tested out with implementation; in the time taken for implementation, better hardware became available. Now ZMOB is stuck with the earlier design, and needs major design changes to change to faster processors. The conveyer belt, being a form of a bus, is subject to the usual problems of a single bus: it becomes a bottleneck when you scale up the architecture. One may consider multiple conveyer belts and interconnecting them, but the gateway nodes could become bottlenecks. It is clear that the ease with which the cosmic cube can be scaled up is not available in the network based on a conveyer belt.

1.4 NON-VON (*Columbia University*)

NON-VON (see Serlin 1985) is a two-stage tree-structured machine. There are two types of elements: Large Processing Elements (LPE) and Small Processing Elements (SPE). LPE have their private memories and are interconnected through a VAX. The SPE are 4-bit processors having a local store of 64 bytes each. Each LPE is at the root of a sub-tree of the entire machine, where the nodes of the sub-trees are SPE. Communication is either through the parent-child links or through a broadcast mechanism.

The LPE can work in an MIMD mode, running different processes, and generally being independent of each other. But the SPE are used in a single instruction multiple data (SIMD) mode. In this mode, each SPE receives instructions from an LPE, and all the SPE in the sub-trees belonging to that LPE execute these instructions simultaneously on

different data. Multiple binary searches, for example, can be performed concurrently on the NON-VON.

Though this configuration is restricted to tree structures, the mode in which the PE operate is very elegant.

1.5 DADO – Columbia University's Production Systems machine

DADO (Stolfo & Shaw 1982) is a tree-structured machine with about 100,000 processing elements (PE), meant to execute large production systems in a highly concurrent fashion. Each PE has a processor, a 2K local memory and an input/output switch. The PE are connected in a complete binary tree. Each PE is capable of acting in two modes: it can behave in an SIMD mode and execute instructions broadcast by some ancestor PE, or act in an MIMD fashion, executing instructions stored in its own local RAM, independent of other PE. A PE in the MIMD mode sets its I/O switch such that it is isolated from higher-levels in the tree.

DADO is based on NON-VON, but unlike NON-VON, is designed for a very specific function. Again, the architecture is interesting, with the possibility of MIMD/SIMD operation at the level of sub-trees in the complete tree.

1.6 Transputer

The transputer (Whitby-Strevens 1985) is a programmable VLSI chip with communication links for point-to-point connection to other transputers. Designed and marketed by INMOS, the transputer is standardized at the level of the definition of the programming language Occam. Occam (INMOS 1984), a CSP-like language (Hoare 1978), permits a multi-transputer system to be regarded as a collection of concurrent processes which communicate by message-passing via named channels.

A collection of transputers may be built to operate concurrently. Transputers may have special purpose interfaces to connect to specialized hardware. Thus, for example, workstations may be built out of a few transputers, some of which act as device controllers, some as interaction processors and some as applications processors. This approach allows redesign and experimentation at low cost.

Transputers directly implement Occam processes. Internally a transputer can behave like an Occam process; in particular, it can use timesharing to implement internal concurrency. Externally, a network of transputers can run Occam processes, which can use Occam message passing to communicate with each other.

The first transputer product is the T424, which is a general purpose 32-bit machine, with 4K of on-chip memory and four bidirectional communication links, which provide a total communications bandwidth of 8 megabytes (MB) per second. There is provision to include off-chip memory.

1.7 Other commercial systems

BBN BUTTERFLY: This machine uses a butterfly interconnection to connect 256 Motorola 68000 chips together. Each processor (soon to be upgraded to M 68020) has about 1 to 4 megabytes which can be partitioned into global and local memory. Each processor runs its own operating system (OS) kernel.

IBM RP3: This research parallel processor uses IBM's own RISC-like processor. A 512-node network, with 4 megabytes at each node, is expected to have a speed of over 1

giga instructions per second (GIPS), and 800 mega floating point operations per second (MFLOPS). The memory at each node is partitionable into various combinations of local and global memory; this will allow RP3 to be used to compare tightly coupled networks with loosely coupled, messaging sort of networks. The interconnection scheme uses a mixture of banyan and omega networks. (For details of interconnection schemes, see Anderson & Jensen 1977; Haynes *et al* 1982; Siegel 1979.) The aim of the current project is to build a 64-node subsystem; the 512-node system is expected to be built using eight such subsystems.

INTEL iPSC: Intel's Personal Supercomputer is a realization of the hypercube design. Each node here is an i286 chip running at 8 MHz, with the i287 numeric coprocessor. Interconnection is through 10MB bidirectional bit-serial links. Each node delivers about 35 kFLOPS to 50 kFLOPS. iPSC comes in 32-, 64- and 128-node versions, and a few systems have already been delivered to customers.

2. Local area networks as computing networks

Imagine the following situation in a typical university or a large office. There are a number of engineering workstations scattered all around, in offices, laboratories, terminal rooms etc. These processors are rated at about 1 MIPS each, and typically have local memories of 1 to 2 MB and local disk storage of over 10 MB. These processors are in use for varying periods of time. When they are not in use, these processors are switched off. All these processors are usually connected together as a local area network (LAN). While LAN have become practical, early hopes raised about distributed computing over LANs are yet to be realized. Most LANs provide for file transfers, electronic mail and for sharing devices such as printers. But there is little distributed computing in many LANs.

The situation as described above is fairly common today; where it is not, it will soon be. The important point to note is that these powerful processors, each capable of about one MIPS, are under-utilised, though they form part of a network, and are accessible across the network. What we need are alternatives to this situation which utilize the available resources in a better fashion. Ideally, in doing so, such alternatives should help us solve other problems.

We propose, in this paper, a distributed computing system called CONE (for Computing Network) which is designed to fit into this framework as described above. This specialized network is not a general purpose machine. Instead, this network is designed to be used in solving specific classes of problems. We explore these classes of problems, and put forth specifications for the proposed network. Some hardware problems and software issues are examined.

We should also note that many workers in the area of LANs have thought of distributed computing in the context of OS implementations. Partitioning an OS into processes which could run on different processors of a network has been a popular idea. We need to contrast this idea with the central idea of this paper: writing major applications as a set of processes, partitioning the work to be done among them, and using many processors of a LAN to execute processes concurrently. We should note that the objectives of the Cosmic Cube network are very similar. We discuss the differences in §3.

3. The structure of CONE – a computing network based on a LAN

We assume that all the processors are networked together using a high speed communication medium, using for example, Ethernet* (Metcalfe & Boggs 1976). We also assume these machines to be multiprogrammed. Most processors will be kept on all the time. This means that each processor will be available to the network with its local memory, whenever it is not being used as a stand-alone machine. Space on the local disk attached to each processor may or may not be available to the network, but typically, the local disk will be available for paging programs currently running on the CPU, even if it is initiated by a 'remote' user.

The communication network will be configured to be adaptive. If a machine goes off-line (because it is being used as a local workstation or because of some hardware or software malfunction), this network will bypass that processor. On the other hand, if a processor becomes available to the network, because it has been switched on or released from local use, the network will accept this processor and add it to the network. LAN workstations (processors) discussed here are visualized to be VAX-11/750, HP 9000 or similar machines. The networking configuration visualized is shown in figure 2.

Local area networks are clearly limited in their communication capability. All the schemes discussed in §1 use special interconnection schemes to provide for very high speed interconnection links, faster than what a LAN can provide. Some schemes use a very high speed conveyer belt, while the hypercube schemes use a large number of connections which work concurrently. The communication capability required is a function of the number of processors used and the level of interactions of partitioned problems. The ZMOB conveyer belt provides for 20 MB capability and the Cosmic Cube provides for $(2 \text{ MBIPS} \times 192 \text{ links}/8 \text{ bits} =)$ about 48 MB per second capability in its 64-node version.

In comparison to these, an Ethernet LAN operating at 10 MBIPS can at the most provide a 1 MB per second capability. But note that this is well in excess of the capability of the 8-node Cosmic Cube used for software development.

Figure 2. CONE- a distributed processing network. Machines 1 to n are the processors connected together in this distributed system. Interconnection is through a rapid transport mechanism, like an Ethernet connection. The processors may be of various types and capabilities.

* Ethernet is a trademark of the Xerox Corporation.

In view of the limited communication capability of LANs, one should view the CONE as a scheme for mid-range parallel processing. Where only a small (say, less than 10) number of processes are required, shared memory schemes may suffice. When a large (say, over fifty) number of processors are required, the LAN bandwidth would not suffice for interconnection. But for parallelism in between the two extremes, CONE is a serious contender.

While LANs are modest in their communication capability, they have a ready-made, rich infrastructure for supporting computation. Using readily available software on LAN workstations, it is possible to speedily implement CONE. It would be possible to have several applications, each with its own process structure distributed over the network, run in parallel on one CONE system. It would be highly desirable to operate in a mode where the CONE system is multiprogrammed in the sense that each CPU is timeshared between a few unrelated processes. This will ensure that processor utilization is high, even in the presence of considerable messaging activity. One way to achieve this is to combine local usage of the workstations for editing, word-processing etc. with a major (CPU-intensive) distributed application. LAN workstations supporting time-sharing and multi-tasking have all the infra-structure necessary for this.

CONE systems can be used for research in partitioning specific problems, in testing out designs for distributed computing systems and for developing software. Systems developed in this manner can then be scaled up using special communication schemes such as hypercubes.

Another option to scale up applications implemented on a CONE is to use multiple Ethernets (or rings) with suitable gateways connecting them. Applications can still be partitioned over a large number of processors, even if some of them have to be accessed through a gateway. There will be a net increase in communication capability when this option is adopted. The traffic across the gateways will be fairly low.

In passive branching bus systems such as Ethernets, there is no significant benefit obtained by taking into account the topology of the network while distributing processes. This is true for token passing rings as well as the Cambridge ring. However, when multiple LANs are used to create a large CONE system, considerations of 'locality' are very important. One would like to cluster related processes within one network to avoid overloading the gateway nodes.

4. Partitionable problem classes

The machine as described above is capable of handling certain classes of problems efficiently. The reason for this is simple. Assume a network of N identical processors of which N' ($\leq N$) processors are available at any given time. For simplicity, assume that all the processors are identical, with maybe minor differences in memory and disk capacities. We need to decompose a problem into a set of P processes, and run these P processes on the N' processors available. For all cases where $P \leq N'$, we can assign a processor for each process, without any conflict. However, if $P > N'$, some processors will have more than one process to run, under time-sharing.

There are various decisions to be taken at different stages of this execution. One of the first is to decide how the problem may be decomposed into tractable, logical, independent modules. Seitz (1985) suggests that we should not look for automatic partitioning of 'dusty old FORTRAN programs'. He takes the view that users should

think of their problems in terms of concurrent processes which communicate through messages. A number of applications can, in fact, be implemented in this manner, with relatively small message traffic.

Another decision evolves from the dynamic nature of the network. Users may log in and start using their system in a stand-alone mode, or log off and release their processors at any time. Thus the number of available machines N' is constantly changing. This factor has to be taken into account by the scheduler which assigns processors to processes. Note that the situation where additional processors become available is easy to handle; but what should the scheduler do if some user wishes to have exclusive use of a processor which is running an active process for someone across the network? Obviously, the state of the external process would have to be preserved and shipped off to another processor available for the task.

At this juncture, we do not propose to detail such possible solutions to all such issues. We merely wish to point out that these have to be taken into account before a complete architecture of this type can be realised. The issue which we shall concentrate on is the partitionability or decomposability of problems. Clearly this is important in a distributed set-up, since this decides the classes of problems that may be solved efficiently using this architecture.

In general, problems handled by this machine have to be decomposable in one of a known number of ways. We will list some permissible decompositions, and problem areas which are covered by these decompositions. In addition to a network of processors, there is a need for a Distributed Computing Executive (DCE) which will manage network-wide resources.

5. The Distributed Computing Executive (DCE)

The Distributed Computing Executive (DCE) performs the following tasks:

- * distribute processes of a given process structure appropriately over the network;
- * acquire resources made available by individuals releasing their nodes;
- * send code and data to process bases and set up the required process structure;
- * support inter-process communication;
- * handle termination of processes;
- * collate information from processes which have terminated;
- * release acquired resources when they are needed for exclusive use by users;
- * consolidate all information to prepare a solution to the given problem.

Note that this DCE can be either centralized or distributed. Again, we do not wish to argue the case for either side but wish to emphasise that the actual modality of implementation must take this also into account. Note also that we assume that some form of inter-process communication facility (IPCF) is available for use by DCE.

A significant fraction of LAN workstations are UNIX[†]-based computers. UNIX provides for very elegant communication among members of a family of processes derived from a common ancestor. A worthwhile option for DCE is to provide for a pipe-like IPCF capable of use between processes even if they are running on different processors. This will be an extension of the pipe scheme, enabling the use of LAN communication to extend pipes beyond the boundary of a processor. Note that the

[†]UNIX is a trademark of Bell Laboratories.

whole process structure of an application can be treated as a family, the DCE being their ancestor. Rashid (1980) describes an IPCF for UNIX. One implementation reported by him uses UNIX version 7 features such as multiplexed files.

5.1 Scheduling

The DCE has to keep track of the number of nodes accessible to it as well as the CPU and memory utilization at each such node. On the basis of a second-to-second appraisal of the situation, the DCE can create processes in lightly loaded nodes, and reallocate processes as required by the situation. The DCE also has to keep track of the dynamically changing process structures of the applications running on it, along with the extent of local usage on all workstations accessible to it. The use of a dedicated node to run the DCE has several advantages. However, this raises issues of fault-tolerance and of handling overloads on the DCE node.

There is need to develop an optimal scheduling algorithm, taking into account the factors discussed above, as well as the costs of interprocess communication and process migration. It would be highly desirable if the DCE could aggregate processes having heavy communication with each other in common nodes, wherever possible, thereby reducing communication loads on the network.

6. Process structures for 'physics' problems

Many practically important problems are concerned with continuous spaces, in 3 or 2 dimensions. If the continuum is divided into a cellular grid, we may in general assume that each grid is affected directly only by its neighbours.

The easiest way to decompose such problems is to divide them into subproblems, each concerned with a sub-space of the problem space. Using an n -dimensional hyper-object as the model in the general case, we can divide the hyper object into n sub-objects. That is, we can distribute the problem across a planar 2-dimensional (2-D) matrix of processors, or across a 3-D cube of processors. Other regular polygons in 2-D, 3-D and higher dimensions may also be used as convenient. The important thing to note is that the connectivity here is usually to the "nearest neighbours". Thus on a square planar grid, each processor will be connected to its four nearest neighbours.

Typically each processor in this grid will execute a program identical to that of its neighbour. All processors will initially be sent an identical copy of the program to be executed. We assume that memory costs are low enough to make such storage of multiple program copies feasible. Each processor then initializes itself, and reads in its own data from a file created by the parent process for just this processor. The data is then processed. Communication with neighbours helps to solve issues of interaction over the boundaries. When all the processing is over, this processor writes out its state (or just values, as programmed) into a file. The parent process scans all such output files and collates the information.

A typical problem that may be decomposed in this manner occurs in numerical weather prediction. There is so much data that has to be processed that conventional architectures do not meet the need. The problem is fairly regular in 3-D space, since the same sort of analysis has to be done at various levels. Again, typically, it is only the neighbours which affect each segment of space, so that communication can be restricted to them.

7. A simple atmospheric model – an example

Let us take a computational example and obtain figures for the CPU, memory and communications capabilities required for a CONE system. The problem that we discuss is that of a simple atmospheric model.

Briefly, the problem is as follows. Consider a chunk of the atmosphere which is a cuboid 1000 kilometres long, 1000 kilometres wide and 10 kilometres high (see figure 3). The variables are pressure, temperature, density, humidity, velocity vectors in three axes etc. We want to model this chunk of atmosphere and the changes that occur here as a function of time.

We start by defining a cell size of $2.5 \times 2.5 \times 1$ km. This is the basic unit of the atmosphere on which processing will be carried out. Assume that 100 FLOPS are required to update the state of the variables in each cell. Further, assume that this calculation has to be done every 100 seconds. From these figures, we calculate the processing capability required of the CONE system.

7.1 Processing capability

The number of cells in this system is 1.6 million $[(1000 \times 1000 \times 10)/(2.5 \times 2.5 \times 1)]$. The number of FLOPS required per update is 100, and the time one can take to carry out this

Figure 3. The problem space and an individual cell.

update is assumed to be 100 seconds. Thus the net processing power of the CONE system must be $1.6 \text{ million} \times 100/100 = 1.6 \text{ MFLOPS}$.

If we decide to use processors with the computational power of an Intel 80286 (or comparable processors) for building a CONE, we can estimate the number of such processors required for this problem. Each 80286 with a 80287 coprocessor delivers approximately 40 kFLOPS. Thus we need about 40 of these processors to build a CONE for the simple atmospheric model described above.

7.2 Memory requirements

We assume that each cell requires 10 words of 8 bytes each, or 80 bytes each. Thus for 1.6 million cells, we need a total memory of $1.6 \times 80 \text{ MB}$, that is, 128 MB. Assuming a total of 40 processors in the CONE system, each processor must have 3.5 MB of memory, which is a manageable figure.

Given this memory on each processor, we find that we can fit about 40,000 cells on each processor. Assuming a stack height of ten in the atmospheric bins, we find that we can model a column roughly 65 cells long, 65 cells broad and 10 cells high in each processor (see figure 4).

7.3 Communication requirements

Consider the problem space as having been distributed among the 40 processors with each processor handling a column of $65 \times 65 \times 10$ cells. There should be communication between the cells on each of the 4 vertical faces of each such column. (The top and bottom faces of each column need not communicate with anyone else.) Each such vertical face is 65 cells long and 10 cells high. Each cell has 80 bytes of information

Figure 4. A column of cells handled by one processor.

which it has to communicate to its neighbour in the column abutting it. Thus the total communication required in one cycle of 100 seconds is $(4 \times 650 \times 80) = 208,000$ bytes, that is, about 200 kilobytes (KB) over 100 seconds. In one second, therefore, we need only 2 KB of information transfer per processor or about 80 KB for the whole CONE system. This is well within the bandwidth of LANs, and so communication will not be a bottleneck in this system.

In other words, communication requirements related to 100 seconds of computing in this application can be handled in 8 seconds of network capacity. This assumes that an Ethernet implementation using the standard 10 megabytes per second (MBPS) transmission capability is efficient enough to deliver 1 MBPS of throughput. It would be valuable to ensure that computing and communication over the LAN are interleaved, avoiding bottleneck situations in which all processors lie idle, waiting for communication to be completed.

7.4 Building better CONE systems

There are various 'worst case' assumptions that we have made here. We can better these, and consider the results.

The first assumption is with regard to the processor. Instead of using an Intel 80286 with a 80287, we may decide to use a faster processor. This will increase the computational power by a significant factor. If we use a faster processor, we can do more computation in the same time. It is therefore possible to use smaller cell sizes, so that the modelling obtained is better. This increase in processing will lead to an increase in communication requirements, but since we are well within the limits imposed by Ethernet, there is no likelihood that the upper bound would be reached. Thus one can vastly increase the computational power of a CONE system by choosing a faster processor.

There is another aspect to choosing a better processor. If we choose a processor with a greater addressing capability, we can increase the amount of memory available with each processor. This will mean that we can pack more cells, in the example above, in each processor. By increasing the number of cells in each processor, and by decreasing the total number of processors used, we reduce the demand for data communication. This, in turn, will increase total system throughput.

Several LANs using a fibre optic medium for communication, with a bandwidth exceeding 100 megabits per second have been reported. Fibre optic links used in telecommunication have already reached over a gigabit/sec of communication capability (Alvey 1984; Fukutomi 1984). This increases communication capabilities ten-fold over that of Ethernet. As and when such LANs are standardised and become widely available, they would considerably increase the range of applicability of CONE systems.

Where LAN capability does not meet the communication requirements of a CONE system, we can use multiple LANs connected together through gateways. Consider a situation where a problem such as the atmospheric modelling problem is partitioned onto multiple LANs connected through gateways. For example, if we partition the atmospheric problem space into 4 parts, each of size $500 \times 500 \times 10$ km, we can map each part onto a LAN as in figure 5. If these LANs are connected through gateways, we will find that the intra-LAN message traffic is far higher than the traffic through the gateways.

Figure 5. Partitioning a large problem onto multiple LAN connected through gateways.

In the case being discussed here, intra-LAN messages would be $2 \text{ KB/node} \times 10 \text{ nodes (per LAN)} = 20 \text{ KB/s}$. Messages going from one LAN to one of its neighbours would be $200 \times 10 \text{ cells} \times 80 \text{ bytes/100 s} = 1600 \text{ bytes/s}$. Since the message size is 80 bytes, intra-LAN messages would be 250. Traffic from any node to one of its neighbours would be 20 messages/s. Thus, even if unit time is made smaller than 100 s (as we had assumed in our problem), the messages passing through will not choke the gateways. The advantage of partitioning lies in the fact that each LAN can support about 1 MBPS of communication (if it is an Ethernet), thereby providing a total communication capability far in excess of that of a single LAN.

8. Enhancing physics process structures

The regular polygon model as described above is too limiting for some classes of problems. We introduce a few refinements in it, so that we retain some of the conceptual simplicity of the model, while we reduce some restrictions.

The first such refinement would be to provide non-uniform grids/interval. That is, cells would be smaller in critical areas, so that these areas would be dealt with in fine detail. For example, in our weather problem, it may make sense to look at cuboids 1 kilometre high at lower levels and 2 kilometres high at greater altitudes. This will provide for fineness of detail where required.

That brings us to the idea of a 'scale'. If we define a unit cell to have a dimension of one unit in each direction, a compound cell may be, say, 8 units long, 4 units wide and

2 units high. The scale in each direction is 8, 4 and 2 respectively. This compound cell will occupy 64 units of volume. Where cells of this size are used, processing needs will be reduced to 1/64th of what it would have been otherwise. Scaling will thus help us operate on larger chunks of the problem space.

If we introduce scaling, we would be tempted to put together pieces at different scales. To avoid possible pitfalls in this solution, we may disallow arbitrary modularity. A good example of such restrictions would be the following:

- a) All cells will be of the same size within the sub-volume handled by a process.
- b) Different processes may employ different 'scales'.
- c) All processes will employ a common IPCF format independent of the 'scale' they are using.

9. Process structures for transaction processing

There are massive transaction processing needs in application areas such as airline reservations which have been met by large computers. Applications like this can be implemented over a large computing network. The process structure would involve an interface process running on every workstation serving an airline staff member handling transactions. This interface process would communicate with a set of database machines, each handling a specific database. For example, all flights originating from a city could be handled by one database machine; several such machines could handle all flights of the airline.

The process structure here is a matrix, where any interface processor could talk to any database processor. Electronic mail could be handled by a separate processor, the 'mail machine'. This process structure provides for a modular increase of network size to meet recurring demands for computing. It also provides for graceful degradation of system performance in the face of processor failures. Appropriate backup schemes, such as having duplicate databases, and having one processor take over the work of another processor which goes down, can be implemented.

It is very important to note here that, in this application, the communications load is low. The data from an interface processor to a database processor and back would be small enough to be carried over a data communication link. This opens up the possibility of a computing network of this type being geographically distributed. A wide area network (WAN) instead of a LAN, could provide the communication infrastructure for this application.

10. Trees and graphs as process structures

We now examine other models of decomposition different from the regular (3-D) structures examined above. The most common of them would be the tree or graph models. In contrast to cubic models that we have seen above, these trees and graphs may dynamically grow and shrink. Thus, in addition to DCE, we may need some additional processes to manage these structures.

Artificial intelligence work frequently involves tree search algorithms. Theorem proving and deduction usually involves graph traversals. A variety of game-playing situations involve traversing minimax trees.

Another class of problems, which involve optimization where branch and bound techniques are used, uses tree searches.

A third class of problems involving a process structure in the form of a tree or graph encompasses a very large number of applications. This class of non-parallel (NP)-complete problems are solved by deterministic machines performing elaborate searches through problem spaces in the form of trees/graphs. Parallel architectures offer the promise of speeding up these searches, using different CPU to explore different sub-spaces of the problem space.

Thus this model of decomposition is widely applicable. In a network as has been described here, tree and graph decompositions can be easily handled. We need two additional processes to manage tree/graph growth – a GROWER process and a FARMER process. Using these, arbitrary tree and graph models, which grow with the problem may be obtained.

10.1 *The GROWER process*

In a tree model, the GROWER process will supply new child processes on request, and keep track of the links between parent and child processes. DCE will prime the process with its program and data and initiate it. Each new child process will then start executing and communicating as necessary. Here, communication can occur only between a child process and its parents. All sibling-level communication will be through the parent, or in general, through some ancestors. Whenever a child process finishes execution, it will signal so to its parent, and to the FARMER.

In a graph model, things are slightly more complicated. The GROWER process has to supply new processes and link them up to other processes as demanded by the computation. The GROWER has also to keep track of all links in the graph. DCE will again initiate the new process after downloading its program and data. In a graph, direct communication is possible between any two immediately connected processes. Therefore, any process can communicate with all its immediate neighbours during its active life. When a process finishes its task, it sends a signal to all the processes it is connected to. Each process maintains a count of the number of processes connected to it. When this count reaches zero, it sends a signal to the FARMER process, indicating that it has no further work to carry out, performing an “exit”.

10.2 *The FARMER process*

The FARMER process interacts with DCE in all its functioning. When it gets a ‘finished’ signal from a process, it communicates with the process and collects results from it. It deallocates the process resources and updates the communication links tables. It feeds all the collated information to the DCE.

Note that both the GROWER and the FARMER processes access the same link tables. If a centralized resource pool exists, both these processes may need to access tables related to the pool. Otherwise, these processes would be independent of each other.

11. **Communication structures with Bulletin Boards**

There has been considerable interest in using Bulletin Boards for communication between arbitrary pairs of processes (e.g., the Hearsay-II Speech Understanding

System, Erman *et al* 1980). There are many situations that we can imagine, where a computational structure (cubic, tree or graph model) can profitably use a bulletin board. This will primarily be to speed up communication between arbitrary processes in the structure.

The 'price' of a bulletin board is surprisingly low. In addition to a process which behaves as a bulletin board, all that is required is a set of entries in the communication links table. All problems which fit into a graph or tree model will fit as well or better in this new model.

12. Conclusions

As outlined above, there are a number of features that distinguish CONE from other computing networks. This scheme involves the CPU and networking hardware and software that are there already, including an infrastructure that supports time-sharing in individual CPU.

As fibre-optics based LAN become commercially available, the scheme will become more valuable. Meanwhile, it can provide an environment for R & D in distributed computing and for software development. With thousands of LAN in operation around the world, the distributed computing executive described above can find wide-spread application. Nodes can be taken out or added to the network at any time. Software development can proceed without waiting for hardware development; hardware can be built later, whenever one is ready for it.

It is a pleasure to thank Prof V Rajaraman, Dr K C Anand, Shri Ajit Dewan and Shri Paritosh Pandya for their comments on a draft of this paper. This work was funded by the Knowledge-based Computer Systems Project of the Department of Electronics, Government of India.

References

- Alvey J 1984 in *The new world of the information society* (eds) J M Bennet, T Pearcy (Amsterdam: Elsevier Science Publishers) pp. 31–35
- Anderson G A, Jensen E D 1977 Computer interconnection structures: taxonomy, characteristics and examples. *Comput. Surv.* 7: 197–213
- Edler J, Gottlieb A, Kruskal C P, McAuliffe K P, Rudolph L, Snir M, Teller P J, Wilson J 1985 Issues related to MIMD Shared-memory Computers: the NYU Ultracomputer Approach. *Conference Proceedings, 12th annual international symposium on computer architecture* (Silver Spring, Md: IEEE Computer Soc. Press) pp. 126–135
- Erman L D, Hayes-Roth F, Lesser V R, Reddy D R 1980 The HEARSAY-II speech understanding system: integrating knowledge to resolve uncertainty. *Comput. Surv.* 12: 213–253
- Farber D J, Feldman J, Heinrich F R, Hopwood M D, Larson K C, Loomis D C, Rowe L A 1973 The distributed computing system. *Proc. COMPCON 73* (New York: IEEE Computer Soc. Press) pp. 31–34
- Fukutomi R 1984 Toward the realization of an information society in Japan: Development of the information network system. In *The new world of the information society* (eds) J M Bennet, T Pearcy (Amsterdam: Elsevier Science Publishers) pp. xxvii–xxxii
- Gottlieb A, Grishman R, Kruskal C P, McAuliffe K P, Rudolph L, Snir M 1983 The NYU ultracomputer – designing an MIMD shared memory parallel computer *IEEE Trans. Comput.* 32: 175–189
- Haynes L S, Lau R L, Siewiorek D P, Mizell D W 1982 A survey of highly parallel computing. *IEEE Comput.* 15: 9–24

- Hoare C A R 1978 Communicating sequential processes. *Commun. ACM* 21: 666–677
- INMOS 1984 *Occam programming manual* (Englewood Cliffs: Prentice-Hall)
- Metcalf R M, Boggs D R 1976 Ethernet: distributed packet switching for local computer networks. *Commun. ACM* 19: 395–404
- Rashid R F 1980 An inter-process communication facility for UNIX, Technical Report CMU-CS-80-124, Department of Computer Science, Carnegie-Mellon University
- Rieger C, Trigg R, Bane B 1981 ZMOB: A new computing engine for AI. *Proc. International Joint Conference on Artificial Intelligence* (Los Altos: William Kaufmann) pp. 955–960
- Seitz C L 1985 The cosmic cube. *Commun. ACM* 28: 22–33
- Serlin O 1985 Parallel processing: fact or fancy? *Datamation* 1: 93–105
- Siegel H J 1979 A model of SIMD machines and a comparison of various interconnection networks. *IEEE Trans. Comput.* 28: 907–917
- Stolfo J S, Shaw D E 1982 DADO: a tree-structured machine architecture for production systems, AAAI-82. *Proc. American Association for Artificial Intelligence* (Los Altos: William Kaufmann) pp. 242–246
- Whitby-Strevens C 1985 The transputer. *Proceedings, 12th Annual International Symposium on Computer Architecture* (Silver Spring, Md: IEEE Computer Soc. Press) pp. 292–300