Exploiting low cost computer networks: applications to distributed processing

Neil Gray
University of Wollongong, nabg@uow.edu.au

Reinhold Friedrich Hille
University of Wollongong

Exploiting Low Cost Computer Networks: Applications to Distributed Processing

N.A.B. Gray and R.F. Hille

ABSTRACT

The AppleTalk system is shown to be a viable environment for experimentation in distributed processing. Two illustrative educational examples of distributed algorithms are presented and discussed. The first example is the deterministic equivalent of the well known non deterministic algorithm for sub graph isomorphism. The second example introduces parallelism by subdividing the travelling salesman problem into sub problems and constructing an approximate solution by combining exact solutions of the sub problems. Issues of computational complexity and efficient usage of the network are discussed.

INTRODUCTION

The idea of multiprocessing is not new in computer architecture. Many experimental designs have been used for special purpose machines and for experimentation with parallel processing (Pearson et al., Seitz). Several vector machines are commercially available (Cyber-205, Cray-XMP, Fujitsu VP-200). The recent resurgence of interest in parallel computation is due to a variety of reasons. For example, the development of VLSI technology has made relatively low priced 32 bit microprocessors avialble, so that multiprocessor systems composed of such microprocessors have become feasible. On the other hand it has become apparent that further increases of the performance of single processor machines have only limited scope because of restrictions arising from the physical size of machines and the finite velocity of light. Multiprocessor systems seem to hold much more promise of increased machine performance than refined single processor systems.

There are three basic schools of thought on the use of parallelism. The first concentrates on optimization and vectorization of compilers which can recognize parallelism and translate the source code into appropriate parallel programs. The second school concentrates on the design of new parallel algorithms. These algorithms will ultimately require new language systems for easy translation into parallel programs. The third school uses new computational models as a basis for the design of multi processsor machines (e.g. the data flow model, Gurd et al.).

The performance of a multi processor system depends on how well it solves the potential problems of partitioning, scheduling, control, synchronization, and memory access. The type of problem that can be solved effectively on a system with concurrent processing is dictated by the machine features and by the language model used to express the algorithm. There are four fundamentally different machine architectures which support parallel computations. (1) Vector machines in which the same operation is performed synchronously on an array of processors. This saves time on the instruction fetches as well as on the execution cycles. (2) Cystolic arrays of processors which perform particular algorithms by "pumping" the data through the cystolic array. Examples are matrix multiplication and fast Fourier transform. (3) Losely coupled groups of processors with access to shared memory. The main problem here is to resolve contentions between processors attempting to access the same part of memory. This enforces sequential behaviour. (4) Losely coupled processors with private memory. This avoids the problem of memory contentions but may require transfer of larger amounts of data (and perhaps programs) betwen processors.

COMPUTER SYSTEM

The discussion in this paper is restricted to systems of losely coupled independent processors with private meory. All communications between processes take place via the transfer of tokens between them. No special language which allows the expression of parallelism exists for this system. The programs for individual processors are written in a conventional programming language for sequential processors (Pascal). The computational task is partitioned by a program which runs on a *master processor*. The scheduling of tasks is also carried out by the master processor. Consequently, the problems of memory access which would arise in the case of shared meory are avoided altogether. Synchronization requirements are therefore minimal and remain the responsibility of the master node which must wait until all servers have completed their allocated tasks. No communication with the master processor is necessary during the execution of the tasks allocated to the *server processes*.

A network of microcomputers joined by a local area network provides an environment wherein limited research and teaching programmes on parallelism may be developed. In the last few years, networked systems of microcomputers have been introduced in a number of universities. Typically, these microcomputers are used as stand alone workstations for program editing and testing. Any network connections are used primarily for access to shared resources such as common printers or disk systems. However, once installed, such networks can be exploited for more exotic applications such as the modelling of parallel computations.

The objective of our project has been to explore means for utilizing a network of microcomputers for both research and teaching on parallelism. The system exploited in this project comprises a number of Macintosh computers mainly used for first year undergraduate teaching. These machines have recently been linked through the proprietary AppleTalk™ network.

The Macintoshs used in this project are standard 128K machines. Quite efficient code can be cross compiled for these machines by use of Apple's Lisa Pascal Development System. Floating point operations are interpretived and are slow. For other types of computation the Macintosh's performance is satisfactory.

The AppleTalk network uses a simple shielded twisted pair cable on which data are broadcast. Individual devices are attached to this network through connector boxes that incorporate small transformers. These provide a passive tap on the network. The Macintosh computer includes special communications control chips that handle all the low level aspects of network monitoring. The network is slow. The transfer rate is only 230.4Kbits per second. This corresponds to a practical data transfer capacity of at best 20Kbytes per second.

The Lisa Pascal Development System includes a library of linkable AppleTalk Manager routines (Apple). These routines provide various data transmission services that are available to application programs. The various data transmission services are defined in terms of different protocols. Most applications would utilize only two of these protocols: Name Binder Protocol [NBP] and AppleTalk Transaction Protocol [ATP]. NBP uses AppleTalk as a broadcast medium. The main use of this protocol is in the initial phase of an application where a consumer process is seeking to establish contact with a server process running on some other machine. ATP provides guaranteed transfer of data packets between applications running on designated machines. New protocols can be defined and used in applications that require specialized facilities (e.g. other types of broadcast message transfer).

In summary, the project had available a number of quite fast microcomputers, each possessing a significant amount of local memory, and for which good code could be compiled. The network facilities were established and simple, reliable data transfer protocols were already defined. For practical applications the major limitation of the system is the low rate of data transmission over the network.

GRAPH MATCHING PROBLEM

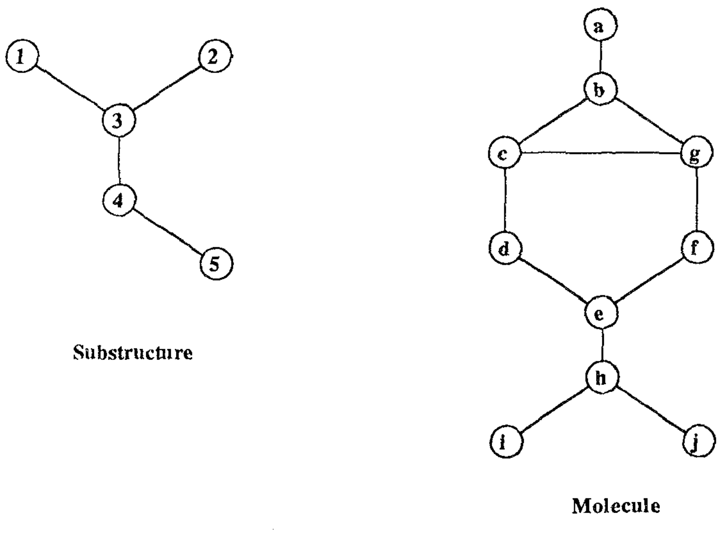
The first application completed for this project involved a distributed graph matcher. The problem of graph maching is conceptually simple, and there are several well known recursive graph matching algorithms. Scope for parallelism is intuitively obvious. A recursive graph matching algorithm establishes some initial partial matches among the atoms (nodes) of two graphs. Each subsequent recursive step entails extension of a partial match by incorporation of additional atoms. Each initial partial match is distinct and, consequently, the elaboration of different partial matchings can in principle be pursued in parallel. The conceptual simplicity of the graph matching problem and its evident suitability for parallel processing make it a good model through which students may be introduced to the problems of re desigmimg conventional algorithms in order to exploit parallel architectures.

Despite its somewhat esoteric sounding nature, the graph matching problem has considerable practical importance. The practical application of such algorithms is in the handling of chemical structural data. Chemical researchers, particularly those working on structure activity relationships in drugs, have frequent need for substructure searches. The chemist will have some substructure whose pharmaceutical activity is of interest. This substructure will be defined in terms of a set of atoms (graph nodes) and bonds (graph edges). The chemist requires the identity of all known molecules that incorporate the given substructure. Known molecules are held on files. The representation is in terms of atoms and interconnecting bonds. Computationally, the chemist's problem becomes one of attempting to match the defined substructure onto each known molecule and to report those molecules where a successful match has been established (substructures and molecules are simply graphs where distinguishing atom names may be attached to the nodes). The problem is usually generalized so that the system will report each distinct matching of the substructure in a molecule if more than one such matching is possible.

Parallelism can be used at two levels in this problem. In the Chemical Abstracts Service's CAS ONLINE system, the main file of some six million known compounds is divided into ten sections each of which is held on and searched by a separate PDP-11 computer (Farmer). The chemist's request for a substructure search is passed to each PDP-11 and ten searches, using conventional matching algorithms, are pursued simultaneously. In addition to this parallelism at the data base level, one can implement parallelism at the graph matching level. It is this parallelism at the algorithmic level that is of current interest. Preliminary simulation studies have recently been reported for a proposed system that will utilize processors connected to a high speed (60Mbaud) bus (Wipke).

An example graph matching problem would be the identification of all instances of the substructure in the molecule as shown in Figure 1. In this example, there are no distinguishing names assigned to the atoms of molecule (graph) or substructure (subgraph).

Figure 1 An example problem for a graph matcher: find all ways of matching the atoms of the substructure onto the atoms of the molecule.



The matchings required are as summarized in Table 1 (as is illustrated by these data, additional bonds are permitted for atoms of the molecule that are matched to atoms of the substructure).

Table 1 Matchings of substructure and molecule.

Substructure Atom:	1	2	3	4	5
Molecule Atom:	a	g	b	d	e
	g	a	b	d	e
	a	c	b	g	f
	c	a	b	g	f
	b	g	c	d	e
	g	b	c	d	e
	d	g	c	b	a
	g	d	c	b	a
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.
	j	i	h	e	d

A conventional graph matcher like the one used in these chemical applications will first perform a preliminary analysis of the substructure to determine an optimal order for matching its atoms. This analysis involves various heuristics (e.g. match atoms with unusual names first) and graph theoretic considerations (e.g. match atoms of high degree first, then match their neighbors, next nearest neighbors and so forth). Such an analysis applied to the illustrated substructure would, in the absence of any distinguishing atom names, determine the optimal matching order as being (3 4 1 2 5). Once this preliminary analysis had been completed, data defining each molecule would be read and the main recursive matching routine invoked.

A somewhat simplified version of a recursive graph matching routine would have the general form indicated by the example in Figure 2.

Figure 2 The recursive graph matching algorithm.

```
gmatch(sn)
int sn;
{   int subatom, gn;
    if (sn>size_of_substructure) report_match()
    else {
        subatom = matchorder[sn];
        for (gn=1; gn<=size_of_molecule; gn++) {
            /* ignore any graph atom already used in this matching */
            if (gmatched[gn]) loop;
            /* ignore any graph atom of inappropriate type */
            if (atomname[gn]!=subname[subatom]) loop;
            /* ignore any graph atom of inappropriate valence */
            if (atomvalence[gn]< subvalence[subatom]) loop;
            /* perform check on already matched nbrs of gn */
            if (check_matched_nbrs(gn,subatom)) {
                /* extend partial matching, and perform recursive call */
                gmatched[gn] = subatom;
                smatched[subatom] = gn;
                gmatch(sn+1);
                /* restore previous partial match, and continue */
                smatched[subatom] = 0;
                gmatched[gn] = 0;
            }
        }
    }
}
```

If all atoms of the substructure are matched (sn > size_of_substructure) then a match can be reported. Matches are generally recorded in duplicate. The array element gmatched[x] will define the substructure atom onto which atom x of the molecule is matched, smatched[y] will define the atom in the molecule matched to atom y of the substructure. The match report will list one or other of these mappings depending on the particular needs of the application. If, however, the matching is still incomplete, the next atom (subatom) from the substructure must be taken in accord with the matching order as defined in the preprocessing step. Each atom in the graph must be checked. Those of appropriate atom type and valence, that are not already used in the partial match, are candidates for matching to subatom. The only complex step in the entire process, check_matched_nbrs(), involves verification that any already matched neighbors of gn are matched to appropriate neighbors of subatom. If all tests are satisfied, the matchings of molecule and substructure atoms can be recorded, the recursive call made, and on return from recursion the matchings can be deleted. Obviously, the search involves backtracking when partial matchings are generated that cannot be completed.

An algorithm along the lines of *gmatch* can serve as a good illustration of the application of recursive techniques to practical problems. Subsequently, it is possible to introduce an alternative formulation fora parallel architecture consisting of several processors with private memory. This architecture can be simulated on the AppleTalk network.

One alternative formulation of the problem is based on the concept of tasks. A task is a request that a given n atom partial matching be extended to include the (n+1)th atom. The processing of a task is similar to the processing involved in the gmatch routine. Unmatched atoms of the molecule that might match with the next substructure atom must be identified. The processing of a task has three possible outcomes: (1) *failure*, the partial match cannot be extended, (2) *solution*, the extended partial match involves the entire substructure, and (3) *branch,* the generation of one or more possible continuations which must be investigated.The three possibilities correspond to the primitives of the same names of non deterministic algorithms. The routine gmatch is actually the deterministic formulation of the well known non deterministic algorithm.

For the substructure and molecule used in the example, a task involving the extension of a partial match ((3,b)) will result in the generation of two new tasks involving the extension of partial matches ((3,b), (4,c)) and ((3,b), (4,g)). Of the four as yet unmatched molecular atoms of appropriate type and valence for matching to 4, only c and g satisfy checks on already matched neighbors.

In addition to a matching routine that extends a given partial match and returns new tasks, the reformulated problem requires a control routine. The control routine will maintain a queue of tasks. The queue would be initialized to contain all alternative one atom matches. For the example data, this initial queue would comprise the matchings ((3,b)), ((3,c)), ((3,e)), ((3,g)), and ((3,h)). Partial matchings would be removed from the queue and passed to the matching routine for extension. The resulting extended matchings returned would be appended to the task queue or, if they represented completed matchings, passed to the reporting routines. This reformulated solution can be coded first for a single processor machine.

Of course, this reformulation of the problem admits parallel implementation. The queuing of tasks must be administered by one processor which we call the master node. Any number of other machines, server nodes, can be used to execute the code that extends partial matches. Both master node and server nodes require the data defining the connectivity of molecule and substructure. These data should be supplied to the servers in an initiation phase of the process. Subsequently, server nodes will request partial matching tasks. In response to each such request, the master node will select a task from its queue and return this task via the network.

The master node will then prompt the server for results. Each extension by one atom of the partial match as found by the server will be returned to the master node for queuing or reporting. When a server has found all possible one atom extensions of its assigned partial matching, it will issue a request for a new task. In addition to handling the allocation and return of tasks, the master node is responsible for obtaining the initial problem specification, for the preprocessing of the substructure, and for the establishment and maintenance of network connections.

Thus, the parallel implementation involves the following steps:

1. The master node reads the graph matching problem, analyses the substructure to determine the best matching order and identifies the server nodes. Tasks will be divided among all servers that respond to the server identification messages broadcast by the master node.
2. The master node issues each server node a definition of the molecule and of the substructure.

3. The master node initializes its queue to include all distinct matches for the first substructure atom.

4. The master node waits for requests and responses from server nodes:
   a. When responding to a task request from a server, the master node will return a partial match taken from its task queue.
   b. Responses from servers will be appended to the task queue or reported as appropriate.

5. The process terminates when the master node's task queue is empty and all server nodes are waiting for tasks.

The AppleTalk system includes tools that permit the monitoring of traffic on the network. Using these tools, it is possible to obtain information defining the patterns and timings of data transfers on the network. The data shown in Table 2 are from an edited recording of data traffic recorded during one run of the graph matcher. The physical node numbers that appear in the actual trace have been replaced by logical names (MN=master node, SN1= 1st server node, etc.). Times are in milliseconds from an arbitrary start of recording time. Messages are identified by code bytes in the message header, these codes are expanded into English text in Table 2. Students can obtain an understanding of the overall behaviour of the system from such recorded data and can thus evaluate various alternative formulations of the graph matching task or other similar distributed processing problems.

This parallel version of the graph matcher runs readily on the AppleTalk network. Because of the low transmission rate of the network it runs more slowly than a conventional recursive graph matcher on a single processor machine. As illustrated by data in Table 2, server nodes generate their partial solutions quickly and spend most of their time waiting for requests from the master node. Any gain from parallel search is more than balanced by delays resulting from the transmission of packets that define partial matches.

Table 2 Example network traffic recorded during a run of the graph matcher.

Source	Destination	Time	Message
MN	SN1	4779	SN1 given copy of graph and subgraph.
SN1	MN	4799	SN1 acknowledges receipt of data.
MN	SN2	4807	SN2 given copy of graph and subgraph.
SN1	MN	4826	SN1 requests matching task
SN2	MN	4830	SN2 acknowledges receipt of data.
SN2	MN	4847	SN2 requests matching task.
MN	SN1	5090	SN1 given task of extending partial match 3-b.
SN1	MN	5095	SN1 acknowledges.
MN	SN1	5103	Request solution(s) from SN1.
SN1	MN	5124	SN1 returns 1st solution 3-b, 4-c.
MN	SN1	5129	Acknowledgement.
MN	SN2	5179	SN2 given task of extending partial match 3-c.
SN2	MN	5184	SN2 acknowledges.
MN	SN2	5192	Request solution(s) from SN2.
SN2	MN	5212	SN2 returns first solution 3-c, 4-b.
MN	SN2	5216	Acknowledgement.
MN	SN1	5327	Request next solution from SN1.
SN1	MN	5333	SN1 returns its second solution 3-b, 4-g.
MN	SN1	5337	Acknowledgement.
MN	SN2	5410	Request next solution from SN2.
SN2	MN	5417	SN2 returns its second solution 3-c,4-d.
MN	SN2	5421	Acknowledgement.
MN	SN1	5494	Request next solution from SN1.
SN1	MN	5500	SN1 announces completion of first task.
MN	SN1	5503	Acknowledgement.
SN1	MN	5525	SN1 requests new task.
.	.		.
.	.		.
.	.		.
MN	SN1	5696	SN1 given task of extending partial match 3-e.
SN1	MN	5700	SN1 acknowledges.
.	.		.
.	.		.
.	.		.
MN	SN2	6584	SN2: task of extending partial match 3-b, 4-c.
.	.		.

THE TRAVELLING SALESMAN PROBLEM

The previous example of a parallel implementation of the graph matching algorithm demonstrates that the ratio between time spent in the transmission of data and time spent processing is an important parameter in the effectiveness of parallel algorithms for processors with private memory. The relatively small amount of processing performed at each step in graph matching makes this application ill suited to effective utilization of such parallelism as the AppleTalk system offers. The basic processing step in graph matching involves a loop, a few tests and the call of a procedure involving a second loop that checks neighbors. This will amount to a few thousand machine instructions, or significantly less than 0.01 seconds of computation time. Only a few bytes (≈64) are needed to define matching tasks. However, these bytes must be copied among buffers on both server and master node and transmitted over a slow network using a request-response-release protocol that requires a minimum of *three* message transfers and an elapsed time possibly greater than the processing time for those data.

Practical applications for the AppleTalk system will be those that involve substantial computations at each processing step. If several seconds of computation time are required at each processing step, then data transfer overhead becomes insignificant. A number of possible computation intensive applications are now under investigation. One preliminary study on a second graph theoretic problem has been completed.

This second problem is the determination of a near optimal solution of the travelling salesman problem with a large network ($N > 50$). The problem is defined as a complete weighted graph of order N, in which the weights of edges represent the Euclidean distances between the vertices.

Rather than implementing a parallel version of a recursive branch and bound algorithm similar to the back track algorithm in the first example, the problem is broken down into a number of sub problems of finding optimal tours of smaller groups of cities. The sub problems are determined by a master process. For each of the sub problems a server process computes an optimal tour using a branch and bound algorithm which is programmed as the usual sequential process. The tour of the entire network is then constructed out of the sub tours by finding a minimum spanning tree of a graph in which the sub graphs form the vertices. The tour constructed in this fashion is not optimal because each of the tree edges must be traversed twice. However, the total length of the tree edges can be kept small by judicius selection of sub problems. The algorithm involvers the following steps:

MASTER PROCESS:

1. Subdivide the given Euclidean graph G_1 into subgraphs of appropriate size and place them into a queue of sub problems.

2. Send all problems on the queue to server processors and place the solutions onto the solution queue in the order in which they are received.

3. Treat the subgraphs as vertices of another graph G_2 with distances being the minima of all possible distances between pairs of vertices of both subgraphs. Construct the minimum spanning tree of G_2.

4. Construct the complete tour which must traverse each sub tour edge once and each spanning tree edge twice.

SERVER PROCESS:

By branch and bound algorithm find the optimal tour of the given sub graph and return the solution to the master process.

The identification of sub graphs is somewhat ad hoc. The total area covered by the points (cities) is broken down into a number of strips each of which is again subdivided into rectangles. The number of strips and rectangles is determined such that the limit of the group size is not exceeded. Boundaries betweeen strips and rectangles are moved until all sub graphs have the same order or their orders are as close in size to each other as possible. Because the computation time required by the branch and bound algorithm grows exponentially with the number of points it is important to limit the size of the groups and to make the differences between group sizes as small as possible. At the same time, the groups should be reasonably large for two reasons:

(a) the larger the groups, the fewer tree edges will have to be traversed and the approximation can be expected to be good;

(b) the sub tasks should require a substantial computational effort compared with data traffic so that the communication overhead in the network becomes insignificant.

Some experiments were carried out with different numbers of server processors and with problems of various sizes. Results are illustrated in Figure 3 and summarized in Table 3. The times given in Table 3 are in seconds and measure the time elapsed between the reading of the data and the completion of output. (Times for execution using Berkeley Pascal on Unix are included to so as to provide a more familiar measure of the cost of these calculations. Berkeley Pascal utilizes an interpreter for intermediate code and is therefore slower than compiled Pascal for the Macintosh in which only the floating point operations are interpreted). The time for a uniprocessor solution is the best measure of the cost of solving these problems on a Macintosh. The difference in times between the uniprocessor solution and that obtained by master node with one server node gives an indication of the size of the overhead associated with use of AppleTalk. This overhead is about 5% of the cost of the computation. It includes the cost of establishing the existence of servers and the costs associated with polling for incoming messages and with the repeated prompting of correspondents when data do not arrive within specified times.

Results obtained with the first two data sets illustrate quite effective use of parallelism. Of course, there is no simple linear increase in performance with the use of increasing numbers of servers. Inevitably, one gets situations where some servers will be working on a second or third task while other servers are idle (e.g. a four server solution does not offer much advantage over a three server solution for data set 1 where there are nine tasks corresponding to nine groups of cities). These problems will be diluted in a situation where the number of sub tasks is very large compared to the number of servers.

Some small savings in time are still possible by better coding. For example, the master node does not start to determine the minimum spanning tree until the last subgraph tour has been returned, although there is no need to wait.

The results for data set 3 are less satisfactory. These results exemplify the need for the master node to arrange an appropriate balance among tasks created for distribution and parallel processing. The strategy employed in set 3 was such that the last task issued would be approximately 50 times as costly to process as any of the other tasks. This highlights the importance of ensuring that all sub tasks have the same size. The system's behaviour degenerates to that of a sequential process after the first 13 small tasks have been processed. The analysis of the probable costs of different tasks is obviously application specific. In a situation where it is impossible to balance the sizes of the sub problems, some estimate of computation costs could define a priority order for task allocation. The large task from data set 3 would be issued first and would occupy one of the servers while the other 13 tasks would be handled by the remaining ones. This would result in a slight improvement of the performance.

Figure 3 Relative performances for different processor configurations as used for a anumber of different data sets for the travelling salesman problem.

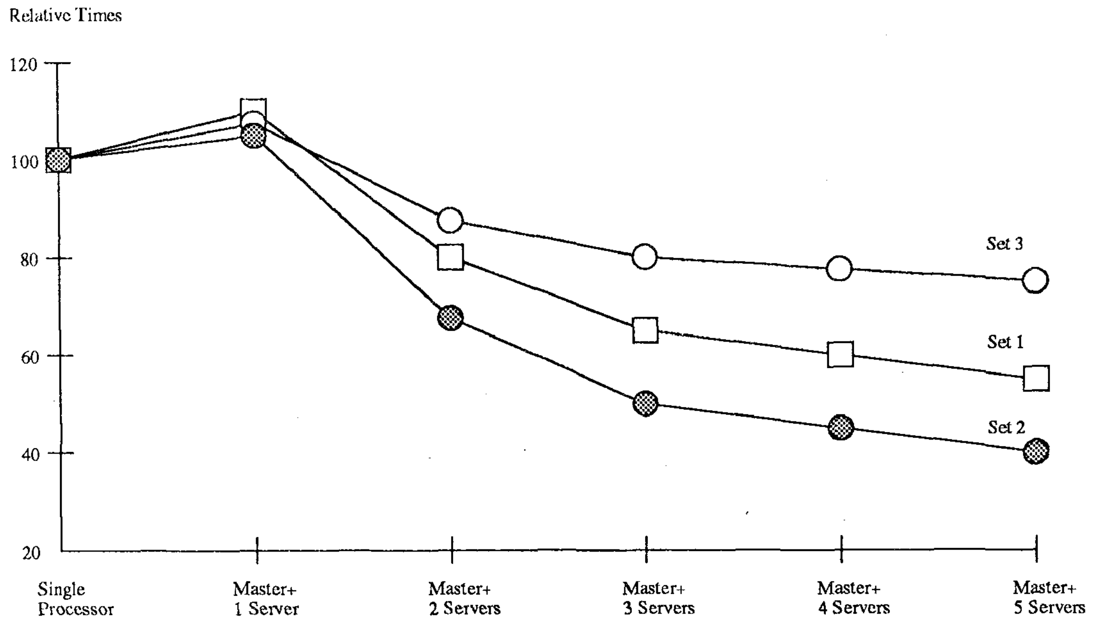


Table 3 Elapsed times for completion of travelling salesman tours.†

Data set	1	2	3
No of cities	54	98	100
No of tasks	9	14	14
Time for uniprocessor solution:	82	435	1072
master node + one server node:	89	458	1131
master node + two server nodes:	64	270	930
master node + three server nodes:	52	216	872
master node + four server nodes:	50	182	844
master node + five server nodes:	45	163	819
Unix Pascal:	66	499	1587

†The times are in seconds; they measure time elapsed from when the problem data are read by the master node Macintosh to when the master node has completed printing results. The Unix Pascal time is the total of user and system times as obtained when running the uniprocessor solution with the Berkeley Pascal system on a Perkin-Elmer 3230.

DISCUSSION

The number of server nodes used in these experiments was restricted by limitations of the AppleTalk manager routines. A node, such as the master node computer, can have at most 12 active *sockets* (sockets are logical entities that are used when defining data paths employed by an application). Some sockets are reserved for use by NBP and similar control routines. In our current applications, each connection between master node and server node requires two sockets. Some sharing of sockets could be realized through alternative approaches to the routing of data. However, the limit on the number of sockets cannot be completely circumvented.

There are applications where a master node server node model would not be appropriate. More general mechanisms for communication may be needed. For example, there are algorithms that determine a canonical numbering for a graph using a recursive backtracking scheme somewhat similar to the graph matching procedures. Partial solutions can be tested against the best canonical numbering found previously, and can be abandoned if inferior. If a canonical numbering algorithm were implemented on a parallel system, it would be appropriate for a server node that discovered a better candidate numbering to broadcast these data to all other servers (so that the other servers could take advantage of the result to limit the number of candidates that they considered). Such broadcast communications between server processes do not match well with the socket to socket message transfer facilities of AppleTalk ATP protocols. However, new protocols could be defined that would allow such communications. These aspects of the use of AppleTalk have not yet been explored in our project.

Much further work is also possible at the level of designing new algorithms for a number of well known computational problems to be used on such a system. For example, a better approximation of the optimum solution of the travelling salesman problem could be found by determining in the master node an optimal tour of the graph G_2 whose vertices are the sub graphs of the problem. Entry and exit points for the sub graphs are then determined in order to minimise the lengths of the tour edges which connect the sub graphs to each other. Once the entry and exit points of each sub graph are known, the server nodes can compute optimal Hamiltonian paths through the sub graphs.

REFERENCES

APPLE (1985): *The AppleTalk Manager: A Programmer's Guide,* in *Inside Macintosh,* Apple Computer Incorporated.

FARMER, N.A. and O'HARA, M.P. (1980): *CAS ONLINE: A New Source of Substance Information from Chemical Abstracts Service,* Database, Vol **3**, pp10-25.

GURD, J.R., KIRKHAM, C.C., and WATSON, I. (1985): *The Manchester Prototype Data Flow Computer,* Comm ACM, Vol **28**, pp34-52.

PEARSON, R.B., RICHARDSON, J.L., and TOUSSAINT D. (1985): *Special Purpose Processors in Theoretical Physics,* Comm ACM , Vol **28**, pp385-389.

SEITZ, C.L. (1985): *The Cosmic Cube,* Comm ACM , Vol **28**, pp22-33.

WALKER, P. (1985): *The Transputer,* Byte, 1985 (May), pp219-235.

WIPKE, W.T. and ROGERS, D. (1984): *Rapid Subgraph Search Using Parallelism,* J. Chem. Inf. Comp. Sci., Vol **24**, pp255-262.