A Computational Study of Distributed Processing Environments with Parallel Multiperiod Assignment Method: A Dedicated Machine versus a Workstation Cluster

Babita Gupta[1] and Jay E. Aronson[2]

[1] California State University, Monterey Bay, Monterey, CA, USA
[2] Department of Management, Terry College of Business, The University of Georgia, Athens, GA, USA

We provide a computational study of two of the more commonly distributed processing environments using a robust parallel multiperiod assignment branch and bound implementation running in the Parallel Virtual Machine (PVM) environment. We describe the multiperiod assignment problem (an np-hard, combinatorial, network-based integer programming problem), an efficient parallel algorithm for its solution, and implementation details. The algorithm is designed for a large-grain computing environment and uses a Master/Slave (or monitor) configuration. Minimal communication among the processes is used. We then describe the PVM software system and the computational platforms being analyzed. We provide a computational comparison of runs on an IBM SP-2 and a cluster of IBM RISC 6000 POWERStations. The results indicate that losses in efficiency occur when employing parallelism in a cluster, but some gains in efficiency can be obtained by a single processor with multiple processes running in a time-shared mode. However, in general, the cluster provides poor parallel performance when compared to the dedicated IBM SP-2 for which linear and superlinear speedups can be routinely obtained. The results may be generalized to any branch and bound method that utilizes a similar tree search strategy.

Keywords: Network Programming, Parallel Algorithms, Assignment Problem, Distributed Computing

1. Introduction

The multiperiod assignment problem is an important specialization of the three dimensional assignment problem, which is a generalization of the classical two dimensional assignment problem. This model describes the optimization problem of assigning m activities (jobs) to n persons over T discrete time periods. In this model, which is the most general case, two types of costs are considered. There is a cost of assigning a person to an activity in each time period, and a cost of transferring a person from an activity in each time period to another activity in the following period. The transfer cost can be set high as a penalty for situations where no employee is allowed to repeat a job assignment for two consecutive time periods. This situation may arise in a hazardous job environment.

One application of the multiperiod assignment model is the case where there are n machines with different efficiencies, or costs for operating on different sets of task requirements at each of n locations over several time periods (Aronson 1986). Any machine can be installed and operated at any location to perform the required tasks. In addition, the machines can be moved, at a cost, from one location to another at the end of each time period. The cost of moving or transferring heavy equipment between locations may be substantial. Other applications of the multiperiod assignment problem occur in the scheduling of parallel activities on concurrent processor computers, the assignment of aircraft to routes, the assignment of salesmen to territories, the assignment of groups within an organization to various projects, and other problems in manpower planning. Franz and Miller (1993) present a recent application involving the large-scale, multiperiod staff assignment problem for scheduling medical residents into rotations at a university teaching hospital. By selecting appropriate costs and using dummy persons and activities, it is also possible for the multiperiod assignment model to describe the hiring and firing of employees, the activation and deactivation of manufacturing divisions, the startup and shutdown of variable time length projects, the assignment of communication satellites into stationary orbits over time, and the facility location/warehouse version of the three dimensional assignment problem (Pierskalla 1968; see Elnidani and Aronson 1991).

In our model, the number of time periods is not restricted to equal the number of persons or activities. Also, the number of persons n, is not restricted to equal the number of jobs m. However, in general, we assume that the number of persons equals the number of activities (jobs), that is, m = n, since if m ≠ n, dummy persons or jobs can be added. Costs associated with assigning a dummy person to a job, or a person to a dummy job are zero, unless specified penalties are required by the model.

Here, we present a computational comparison along with a parallel branch and bound algorithm and its implementation for solving multiperiod assignment problems on the Parallel Virtual Machine (PVM) software system: 1) on a cluster of IBM RISC/6000 POWERStations; and 2) on a dedicated IBM Scalable PowerParallel System (IBM SP-2). The algorithm is designed for a large-grain computing environment.

The purpose of our study is to identify some inherent advantages and disadvantages of the workstation cluster environment for parallel algorithm implementation and use. The workstation cluster environment has evolved into an implicit standard computing environment, but, for algorithmic research, there is little control over the load of the individual workstations and/or over their communication traffic. Consequently, it is difficult to determine the usefulness of this environment for parallel optimization.

Most of our observations are generalizable to parallel integer programming implementations. In the following sections, we present the multiperiod assignment problem, a brief discussion of the distributed processing environments, the parallel algorithm and its implementation, comparative parallel computational results and comparison with the serial implementation, a summary and conclusions.

2. The Multiperiod Assignment Problem

The multiperiod assignment problem is a special case of the planar three-dimensional assignment problem (for an excellent survey, see Gilbert and Hofstra 1988; for related work see Gilbert and Hofstra (1987, 1992) and Hofstra (1990)).

There is a cost associated with the assignment of person k to job j, in time period t which may be different from that of assigning person k to job j in period t' where $t' \neq t$. Another set of costs is introduced, that is the cost of transferring person k from job j in period t to job l in period $t+1$. The problem of optimally assigning n persons to n jobs over T time periods is the *Multiperiod Assignment Problem* (MAST) stated as follows (see Aronson 1986):

$$\min \sum_{k=1}^{n} \sum_{(i,j)\in A} C_{ij}^{k} X_{ij}^{k} \qquad (1)$$

subject to

$$A^k X^k = r^k, \quad k = 1, \ldots, n \qquad (2)$$

$$\sum_{k=1}^{n} X_{ij}^{k} \leq 1, \quad t = 1, \ldots, r \qquad (3)$$

$$X = 0, 1 \qquad (4)$$

where k is the commodity (job or person) to be assigned,

i, j are the activities, and

t is the time period.

Constraint set (2) represents n independent, specialized shortest path problems (see Figure 1). The n activities are represented by the 'rows' of nodes and arcs. The assignment arc sets A_t for $t = 1, \ldots, T$ and transfer arc sets T_t for $t = 1, \ldots, T-1$ are shown. There are nT mutual capacity constraints, which incorporate only the assignment arcs. The mutual capacity constraints (3) prevent the assignment of

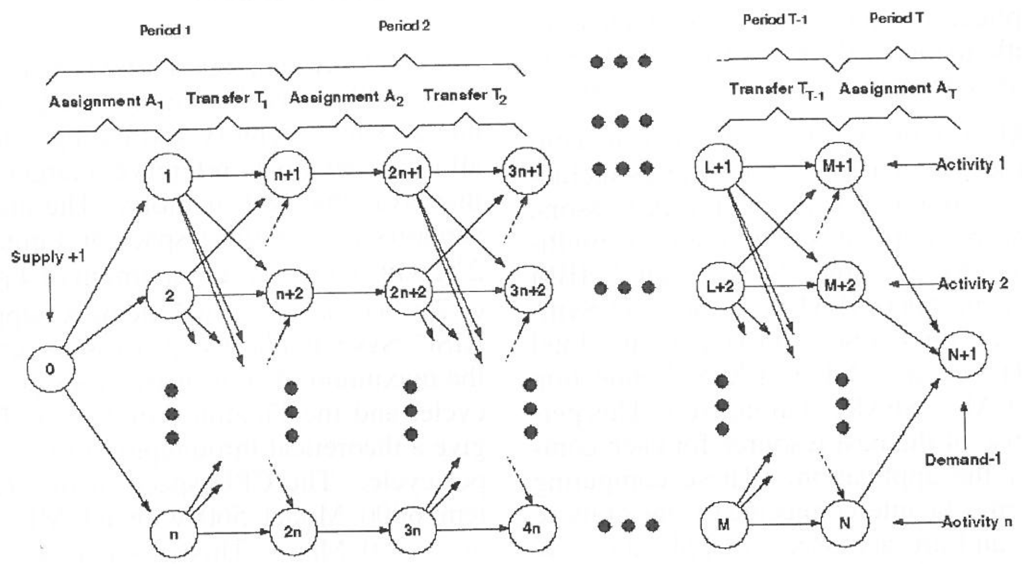

Fig. 1 The Multicommodity, Multiperiod AssignmentProblem Network of Commodity$i$.

an activity (job) to more than one commodity (person) in each time period. Constraint set (4) imposes the integer restrictions. Because the requirement in each of the shortest path problems for the source is $+1$ and for the sink is -1, the upper bounds imposing limits on the arc flows to be 1 are redundant.

MAST may be solved by a specialized branch and bound algorithm (Aronson 1986) (see Elnidani (1986), Elnidani and Aronson (1990a, 1990b, 1991) for a different, specialized multicommodity branch and bound algorithm and implementation). First, we find a feasible integer solution to the problem using a specialized shortest path based heuristic to obtain an incumbent solution, yielding an upper bound to the optimum objective value. Relaxing the mutual capacity constraints (3), that is, allowing multiple assignments, we again solve the problem for each commodity using a specialized shortest path algorithm to obtain a lower bound to the optimum objective value. Branching is done on infeasible assignments. So, in a time period in which a mutual capacity constraint is violated (e.g., persons 2 and 7 each perform job 3 in period 5), we do the branching using a depth first rule (left branch: person 2 is assigned job 3; right branch: person 2 is not assigned job 3). We move down the left branch and solve its subproblem with the mutual capacity constraints relaxed. Whenever a better feasible solution is found, we update the incumbent solution. The fathoming procedure follows standard rules (in terms of when we branch to the right move up and then down to the right). Next, we briefly discuss the computational software platform PVM for handling communication.

3. The Parallel Virtual Machine (PVM)

Advanced systems are being developed to permit the utilization of a heterogeneous network of parallel and serial computers as a unified general and flexible concurrent computational resource. One of them is the Parallel Virtual Machine (PVM) (Sunderam 1990; Sunderam and Geist 1992), which supports the message-passing, shared memory and hybrid paradigms. The user views the PVM system as a loosely coupled, distributed-memory computer programmable in C or Fortran with message-passing extensions (Beguelin et al. 1991). PVM allows supercomputer-level concurrent computations to be carried out on interconnected networks of heterogeneous computer systems. This makes PVM very cost effective as the supercomputer-level computations can be attained at a fraction of the cost of single-system distributed computing platforms. The main advantage of PVM is derived from its ability to execute applications in existing network environments without the need for specialized hardware or software enhancements or modifications.

The PVM system may be configured to contain a variety of machine architectures including sequential processors, vector processors, distributed-memory or shared-memory multiprocessors (for example: Sun 3, Sun 4, IBM RISC System/6000, IBM SP-2, Sequent Symmetry, Cray, Intel iPSC/860 Hypercube, Intel IPSC/2 Hypercube, Alliant FX/8, Connection Machine CM-2, SPARCstation, etc.). This permits the use of the best resource for each component of the application. These computing elements may be interconnected by one or more networks and are accessed by application via a library of standard interface routines. These routines allow the initiation and termination of processes across the network as well as communication and synchronization between processes.

An interesting aspect of PVM is that processors accept work within the system if they are deemed to have the capacity to perform it. Several "virtual processors" may run on a single computer in the network, and other users may simultaneously be utilizing the processors through time-sharing. Thus, if a processor is idle within PVM, it may still be actively working on other tasks, in other words, machine cycles are not lost if the parallel job becomes idle. Application programs are composed of components that are subtasks at a moderately large level of granularity. During execution, multiple instances of each component may be initiated.

PVM provides two options to effectively use any multiprocessor that may be available to an application: 1) the ability to treat multiprocessors as an atomic resource, that is, applications may run programs that are hardcoded for specific multiprocessors under PVM control; and 2) the provision for dynamic incorporation of application modules in a selective manner. At run time, the user or PVM can select the most appropriate module to use, depending upon the specific machines on which the application will run. PVM is used as the primary parallel communication scheme in both the cluster of workstations and the IBM SP-2.

3.1. IBM RISC System/6000 POWERStation

The RISC System/6000 (IBM 1990) architecture is a Reduced Instruction Set Computer. It has a virtual memory addressing scheme that allows programs to reference a larger data area than available real memory. The architecture supports a real address space of 4 gigabytes, or 2^{32}, with a theoretical maximum of 4 petabytes, or 2^{52} because of virtual memory support. The RISC System/6000 a processor can execute the maximum of four instructions per machine cycle, and the Floating Point Unit (FPU) can give a theoretical throughput of two operations per cycle. The CPU speed in the RISC System/6000 Model 560 with 64 MB of memory is 50 Mhz. This gives peak megaflops rate of FPU to be 100 MFLOPS (one megaflop is equal to 1,000,000 floating point operations per second). The RISC System/6000 runs the AIX version 3.2 (Unix) operating system. The University Computing and Network Services' (UCNS) Workstation Cluster at The University of Georgia consists of 13 RISC/6000 POWERStations (actually configured as one cluster of 10, and a second one of 3): the specific Models are the 390, 550, 560 and 590. We used four in parts of our experiments.

3.2. The IBM Scalable PowerParallel System (SP-2)

The University of Georgia's IBM SP-2 parallel processing system has 8 nodes connected by a high performance switch. Each of these nodes consists of a RISC/6000 Model 390 processor chip with 64 MB of memory, 2 gigabytes of disk space and has a peak MFLOP rating of 266. The switch has a 40 MB/sec bidirectional bandwidth between node pairs and an application uniform latency less than 40 microseconds. The University Computing and Networking Services' IBM SP-2 runs under the AIX version 3.2 (Unix) operating system. There are two communication channels or switches called: 1) the High Speed Switch; and 2) the Low Speed Switch. The High Speed Switch option allows for faster communications, but only one computing process may run on each hardware processor. The Low Speed Switch, though slower, allows multiple processes per processor and is more appropriate for our research, where a Master Process has minimal work (about 0.1% of the effort) once the optimization begins. The Low Speed Switch uses the TCP/IP communication protocol. Further, the SP-2 is divided into two 4-processor segments, of which only one may be activated at a time by an individual user, thus limiting our tests to 4 processors.

4. A Parallel Multiperiod Assignment Algorithm and Implementation

Branch and bound methods are essentially the only tool available to solve hard combinatorial problems to optimality (de Bruin et al. 1988). There are several parallel branch and bound algorithms in the literature (Kindervater and Lenstra 1988, Gendron and Crainic 1994), parallelized at a *low* or *high* level. We use the branch and bound method of Aronson (1986) and develop a high-level parallel algorithm, that is, several iterations of the main loop are performed in parallel and the work is partitioned and assigned dynamically during execution.

The algorithm uses a master-and-slave processes configuration implemented under PVM. The Master Process spawns all the Slave Processes. These processes are now activated and are waiting for further instructions from the Master Process. The idea is that all processes are initialized and read problem data simultaneously. Otherwise, if a Slave Process were to be activated during the branch-and-bound operation, each Slave Process would need to be initialized by reading the bulk of the problem data, thus reducing the effectiveness of parallelization. The Master Process decides how the work should be divided among Slave Processes, broadcasts requisite information and maintains the level of synchronicity. Therefore, the Master Process makes all important decisions. The Slave Processes perform the actual subproblem solving, which includes branching, computing lower bounds and fathoming the subproblems generated. Each maintains 'local' incumbent solutions (its best one found) and the global incumbent objective value (shared by all processes). The number of Slave Processes spawned depends upon the size of the problem and the resources available.

The Master Process maintains the information obtained from Slave Processes and coordinates the information exchange among them. The Master Process spawns the Slave Processes, and keeps a list of processes that are currently busy working on a subproblem, in a *process busy queue*. It also maintains a list of idle Slave Processes, i.e., the Slave Processes currently available for work in the *process idle queue*, and the list of subproblems waiting for a Slave Process assignment in *subproblem wait queue*. The Master Process obtains the global upper bound (UB_{global}) from the heuristic solution and executes the shortest path algorithm to obtain the lower bound (LB) at node 0. It then dynamically activates the first generation Slave Processes to start the branch-and-bound algorithm.

Initially, the Master Process assigns two Slave Processes for the first infeasible assignment found, introducing a dichotomy in the search tree. For example, if in period 5, persons 2 and 7 each perform job 3, we branch using a depth-first rule and assign the left branch (person 2 assigned job 3) to Slave Process one and the right branch (person 2 is not assigned job 3) to Slave Process two. The partial tree assigned to a Slave Process has flags indicating which portion of the tree 'belongs' to it.

Each Slave Process works independently on its part of the search tree. A Slave Process may send part of its tree to the Master Process, thus requesting that the Master Process assigns another Slave Process to work on it (simultaneously, it marks the portion of the tree transmitted to the Master Process so it can be ignored when exploring the tree). This Slave Process then works on the remainder of its subproblem until it is completely explored (portions may be fathomed). This ensures that the tree is spread out breadth-wise early in the algorithm, so that each Slave Process has a reasonable amount of work while executing a depth-first search. This also ensures that algorithm achieves near perfect load-balancing. Note, our early tests indicated that the most effective way to partition the search was for a Slave Process to send a portion of the search tree back to the Master Problem as high in the tree as is possible, say the first three right branches. The depth of the tree, where the Slave Processor starts, indicates the number of right branches to send to the Master Process; fewer right branches are sent as we move deeper in the tree.

The Master Process keeps a list of subproblems that are waiting to be assigned to a Slave Process. A Slave Process, after it finishes work on its subproblem, indicates to the Master Process that it is available for further work. The Master Process then assigns a waiting subproblem (the partial tree information) from the subproblem wait queue to this Slave Process (at random). However, if the subproblem wait queue is empty, the Master Process assigns any Slave Process becoming idle to the process idle queue, for later use.

When a Slave Process fathoms a subproblem by finding a new candidate for the incumbent solution, it computes the new current UB, updates the local upper bound, Z^* and global upper bound, UB_{global}, and broadcasts it to all currently active Slave Processes for future fathoming operations, and to the Master Process. Each currently active Slave Process thus receives the updated UB_{global} as soon as it becomes available. A Slave Process checks the message for the updated UB_{global} after it evaluates a node in the tree (checking is as expensive as a simple logical IF statement). Thus, each of the Slave Processes can use the updated UB_{global} to conduct the lower bound fathoming tests on the subproblems it generates. These tests determine whether to prune its part of the tree or to continue branching. If the result of this test is such that the entire subproblem is fathomed, then the Slave Process indicates to the Master Process that it is now free for more work. If no more work is available, i.e., the *subproblem wait queue* is empty, then it joins the *process idle queue* maintained by the Master Process. The branch-and-bound algorithm terminates when all Slave Processes are in the process idle queue and no subproblems are in the *subproblem wait queue*. Then the Master Process sends a done signal to the Slave Processes, which compare their local incumbent objective value to the global upper bound value. If equal, the Slave Process transmits its solution, which is an optimum, to the Master Process.

Communication among the Master Process and the Slave Processes is an integral part of the algorithm. The Master Process can receive four types of messages from a Slave Process:

1. that the status of the Slave Process is idle (message type = Idle),
2. a new *subproblem* to be assigned to another Slave Process (message type = Subproblem), and
3. a *new* UB_{global} (message type = NewUB) (also sent to all active Slave Processes).
4. an optimal solution (message type = OptSol).

A Slave Process can receive the following types of messages:

1. a *new* UB_{global} from another Slave Process or the Master Process (message type = NewUB),
2. a *done* message from the Master Process (message type = Done) implying it should transmit its solution if $UB_{local} = UB_{global}$, and
3. a *new* subproblem to be assigned to this idle Slave Process (message type = New).

We use the depth-and-breadth-first combined strategy while exploring the branches. A Slave Process, while branching, always takes the left branch subproblem (depth-first). The first few right branches are assigned to idle Slave Processes by the Master Process from the *process idle queue* (breadth-first). Thus, many subproblems can be explored simultaneously, depth-wise as well as breadth-wise. The Master Process selects subproblems (tree segments) from the subproblem wait subproblem wait queue at random. For additional details on the algorithm see Gupta (1995).

5. Computational Results

We report the computational results of the parallel branch-and-bound algorithm described in Section 4. The code, PMAST, was implemented in Fortran 77 and runs under PVM. It was tested on the IBM RISC System/6000 POWERStation Cluster and the IBM Scalable PowerParallel System (SP-2) of the UCNS at The University of Georgia. The RISC/6000 Cluster consists of machines of the same architecture but different MFLOPS ratings. The UGA Cluster consists of one Model 590, two Model 560's, seven Model 550's, and three Model 390's. There were actually two clusters, one consisting of three Model 390's, and the second consisting of the 500 series machines. Each cluster had dedicated communication channels within the cluster. The

Model 590 is the fastest machine in the cluster, faster by a factor of 1.80 compared to the Model 560; a factor of 2.18 compared to the Model 550; and a factor of 13.45 compared to the Model 390. Recall that the SP-2 consists of eight RISC/6000 Model 390 processors with some additional, speed enhancements. Each processor is 14% faster than the Model 590.

On the cluster, the maximum CPU time available to any user was 660 CPU seconds (11 CPU minutes). The wall clock time = CPU time limit on the SP-2 was 21,600 CPU seconds, or 6 CPU hours. Both the serial code, MAST (Aronson 1986) and the parallel code, PMAST, were tested on the same processor to ensure a fair comparison of results.

We use the measure of absolute speedup which measures the reduction in time due to the parallelism, and can be used to compare two computing platforms, given that the serial and the parallel algorithms are the same on the different platforms. The absolute speedup is calculated by dividing the time required by the serial algorithm, MAST, on a platform by the time required by the parallel algorithm, PMAST.

The xlf (f77) Fortran compiler was used with the optimization feature enabled at the highest level (-O3). The number of people, n, ranged from 5 to 10, and number of time periods, T, ranged from 4 to 9, when it was possible to solve a problem within the allowed maximum execution time. Multiperiod assignment problems were randomly generated from a uniform probability distribution with assignment arc costs ranging from 1 to 100, and transfer arc costs ranging from 1 to 1000. These problems are extremely difficult to solve because the paths of each network are approximately of the same length. We had to work with two sets of problems for the two computational platforms, the cluster and IBM SP-2, due to practical limitations. Owing to the time limitation of 11 CPU minutes available to any user on the cluster, we could not solve large problems on the cluster. The same set of eight problems that we could solve on the cluster, however, ran so fast on SP-2 (in a few CPU seconds) that one could not effectively measure any speedup. Hence we solve a different set of four very large problems on the SP-2. This, however, does not detract from the merit of the study, as we make speedup comparisons of the two platforms based on solving a problem in serial and parallel algorithms on a particular platform. In other words, speedups are a ratio, and therefore can be compared regardless of the problems solved, so long as the algorithms, MAST and PMAST, themselves are the same on the two platforms.

Three runs were made for each of the eight problems tested on IBM RISC/6000 Cluster. The computational results obtained on the RISC/6000 Cluster are reported in Tables 1 and 2. We ran the parallel implementation using four Slave Processes. The Master Process (called the PVM host program) of the PMAST code ran concurrently on processor one as the fifth process on the four (hardware) processors, so, only four processors were used overall. The Master Process typically requires about 0.10% or less computing effort.

Table 1 shows the results when all four Slave Processes in PMAST were started on four different machines in the cluster: two Model 390's, one Model 550 and one Model 560. Since we have a CPU time limit of 11 CPU minutes for any job on the cluster, we could not solve very large problems. MAST, the serial implementation, was run on the Model 590, the fastest machine in the cluster. The average speedup (solution time of the serial implementation divided by the solution time of the parallel implementation) for this set of tests was a very poor 0.612 with a standard deviation of 0.183. The cluster is shared by many users. Heavy cluster usage and network communication traffic has significant impact on the synchronization and communication time. Our parallel algorithm, though not communication intensive, does require timely computation of local upper bounds and their data transmission, and therefore, heavy network use and traffic affect the overall solution time of PMAST. Thus, our tests of running four Slave Processes on four different processors of differing hardware ratings, communication specifications and different loads, due to time sharing with other users on the cluster, indicated that we were pursuing a fruitless line of empirical research.

We then ran the parallel implementation with all four Slave Processes running on a *single processor*, the Model 590. For the eight problems solved, the average speedup (the solution time of the serial implementation divided by the solution time of the "parallel" implementation)

					PMAST on four different cluster processors (NPROC=4)				
Prob-lem	Run 1 Time	MAST Run 2 Time	Run 3 Time	MAST Average Time	Run 1 Time	PMAST Run 2 Time	Run 3 Time	PMAST Average Time	Speedup MAST /PMAST
558	9.74	10.11	13.45	11.1060.31	18.29	24.51	34.37	0.323	
665	46.92	46.16	44.61	45.90	101.24	88.88	105.29	98.47	0.466
666	78.99	77.29	77.83	78.04	112.67	137.12	113.52	121.10	0.644
775	46.77	51.78	50.40	49.65	91.72	72.62	72.27	78.87	0.630
884	83.24	81.57	92.80	85.877	105.28	109.27	112.68	109.08	0.787
885	208.70	209.11	219.18	212.33	336.32	425.72	419.75	393.93	0.539
895	456.93	458.46	456.46	457.28	457.33	456.60	516.17	476.70	0.959
993	13.64	13.64	15.60	14.29	25.29	26.36	26.28	25.98	0.550
							Average Speedup =		0.612
							Std Deviation =		0.183

Table 1. Computational Runs of the parallel branch-and-bound algorithm implementation, PMAST, on the RISC/6000 Cluster on four different processors, two RISC/6000 Model 390's, one Model 550 and one Model 560, running four Slave Processes concurrently, i.e., NPROC = 4.

The Problem number, *nmt*, means *n* people, *m* jobs and *t* time periods (558 = 5 people, 5 jobs, 8 time periods). Each problem is solved three times (Run 1, Run 2, and Run 3).

Column MAST is the CPU time (in seconds) of the serial implementation on a RISC/6000 Model 590, the fastest in the cluster.

Column PMAST is the wall clock time (in seconds) of the parallel implementation.

using four Slave Processes was 1.825 with a standard deviation of 0.382 (see Table 2). As we can see, the average speedup is much lower than the desired perfect 4.000 (the case of a linear speedup). However, using a single processor, a speedup exceeding 1.0 was attained. Firstly, running the four Slave Processes on a single processor is a time-shared type of processing environment rather than true parallel processing. In effect, we are broadening the branch and bound search, where four searches are running in time slices. Thus, letting the

					PMAST on a single cluster processor (NPROC=4)				
Prob-lem	Run 1 Time	MAST Run 2 Time	Run 3 Time	MAST Average Time	Run 1 Time	PMAST Run 2 Time	Run 3 Time	PMAST Average Time	Speedup MAST /PMAST
558	9.74	10.11	13.45	11.10	4.74	5.37	5.64	5.25	2.114
665	46.92	46.16	44.61	45.90	33.33	29.95	31.66	31.65	1.450
666	78.99	77.29	77.83	78.04	47.78	53.63	43.66	48.36	1.614
775	46.77	51.78	50.40	49.65	25.62	47.67	29.00	34.10	1.456
884	83.24	81.57	92.80	85.87	43.13	32.55	36.57	37.42	2.295
885	208.70	209.11	219.18	212.33	149.50	150.86	114.96	138.44	1.534
895	456.93	458.46	456.46	457.28	166.73	226.97	159.75	184.48	2.479
993	13.64	13.64	15.60	14.29	8.87	8.00	9.04	8.64	1.655
							Average Speedup =		1.825
							Std Deviation =		0.382

Table 2. Computational Runs of the parallel branch-and-bound algorithm implementation, PMAST, on the RISC/6000 Cluster. PMAST is running 4 Slave Processes (NPROC = 4) on a single processor, a Model 590 machine.

The Problem number, *nmt*, means *n* people, *m* jobs and *t* time periods (558 = 5 people, 5 jobs, 8 time periods). Each problem is solved three times (Run 1, Run 2, and Run 3).

Column MAST is the CPU time (in seconds) of the serial implementation on a RISC/6000 Model 590, the fastest in the cluster. Column PMAST is the wall clock time (in seconds) of the parallel implementation.

operating system and PVM control the data structures, solution management and the search through the communication system, a speedup of 1.825 was attained. Secondly, the effects of heavy network traffic and time-sharing with other users explain why the speedup was worse when PMAST ran on four different machines in the cluster than when it ran on a single processor.

To test the above premise and the efficacy of our parallel algorithm, we solved four large problems on the IBM Scalable PowerParallel System (SP-2), a dedicated parallel machine (no other users or time sharing). We solved problems with both MAST and PMAST on the SP-2. The set of problems that we solved on the cluster was small. They ran so fast on the SP-2 (0.5-9.0 CPU seconds) that we could not observe any speedup factors (an SP-2 node is 14% faster than a Model 590 machine, as our early empirical tests indicted). Therefore, we decided to solve a different set of problems on the SP-2. These problems were much larger and much more difficult to solve; in fact, due to time limitations we cannot solve them on the cluster. The computational results are shown in Table 3. The average speedup with four Slave Processes on the SP-2 is 4.726 with a standard deviation of 0.906. Thus, we obtain a *superlinear speedup* when our parallel implementation of the multiperiod assignment method runs in a dedicated parallel environment for these test problems.

6. Summary and Conclusions

We have presented a computational comparison of a new, parallel branch and bound algorithm for solving the multiperiod assignment problem using the Parallel Virtual Machine (PVM) software system for communication on a cluster of IBM RISC/6000 POWERStations and on an IBM Scalable PowerParallel System (SP-2).

The results indicate that a large-grain parallelization and problem partitioning, which involves transmitting large problem tree segments (with minimal actual data transfer) to be solved independently, is very effective. This is interesting in its own right, because reports in the literature indicate that fine-grain parallelization and problem partitioning where individual nodes of the branch and bound tree are solved on Slave Processes and coordinated by the Master Problem, are less effective and that sublinear performance is typically attained (Gendron and Crainic, 1994).

The results further indicate that it is ineffective to utilize multiple processors in a cluster, yet, parallel code running of multiple processes

				PMAST on a 4 different SP-2 processors (NPROC=4)				
Problem	Run 1 Time	MAST Run 2 Time	MAST Average Time	Run 1 Time	PMAST Run 2 Time	Run 3 Time	PMAST Average Time	Speedup MAST /PMAST
1103**	288	288	288	56	56	57	56.33	5.112
884	167	167	167	50	54	53	52.33	3.191
777	476	475	475.5	92	96	94	94.00	5.059
779	11445	11405	11425	1922	2383	1879	2061.33	5.543
							Average Speedup =	4.726
							Std Deviation =	0.906

$**$ 1103 = 10 people, 10 jobs and 3 time periods.

Table 3. Computational Runs of the parallel branch-and-bound algorithm implementation, PMAST, on the dedicated SP-2. PMAST is running 4 Slave Processes (NPROC = 4) on 4 processors.

The Problem number, *nmt*, means *n* people, *m* jobs and *t* time periods (558 = 5 people, 5 jobs, 8 time periods). Each problem is solved twice for MAST (there was virtually no variation) and three times for PMAST.

Column MAST is the CPU time (in seconds) of the serial implementation on a single SP-2 processor (comparable to a RISC/6000 Model 390, about 14% slower than the Model 590). Column PMAST is the wall clock time (in CPU seconds) of the parallel implementation.

on a single processor can broaden the search tree and attain speedups exceeding 1. However, a dedicated multiprocessor is, by far, the best computational platform to use (e.g., see Kennington and Zhang (1988) who report on solving pure assignment problems in a dedicated, shared memory computing environment).

The computational results demonstrate that superlinear speedups can be achieved (unlike sublinear speedups reported in the literature, see Gendron and Crainic, 1994), given the right implementation of the parallel algorithm, with its solution/search parameters set carefully and matched to the proper parallel architecture. The results may be generalized to any branch and bound method that utilizes a similar tree search strategy.

Further work will focus on more extensive testing of this algorithm and on developing methods for solving similar, difficult large-scale combinatorial optimization problems. Other work can focus on parallel implementations of methods for related problems (e.g., see Hofstra 1990).

Acknowledgement The authors would like to express their appreciation to Dr. Alan M. Ferrenberg for his invaluable assistance in facilitating the use of the hardware platforms and PVM.

References

ARONSON, J. E., The Multiperiod Assignment Problem: A Multicommodity Network Flow Model and Specialized Branch and Bound Algorithm, *European Journal of Operational Research*, **23**, 3 (1986), 367–381.

BEGUELIN, A., J. DONGARRA, A. GEIST, R. MANCHEK AND V. SUNDERAM, A Users' Guide to PVM Parallel Virtual Machine, *Oak Ridge National Laboratory Technical Memorandum*, Oak Ridge, TN, July 1991.

DE BRUIN, ARIE, A. H. G. RINNOOY KAN AND HARRY W. J. M. TRIENEKENS, A Simulation Tool for the Performance Evaluation of Parallel Branch and Bound Algorithms, *Mathematical Programming*, **42**, 2 (1988), 245–271.

ELNIDANI, M. A., The Multicommodity, Multiperiod Assignment Problem, *Doctoral Dissertation*, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, TX, 1986.

ELNIDANI, M. A. AND J. E. ARONSON, The Multicommodity, Multiperiod Assignment Problem I: A Specialized Branch and Bound Algorithm, *Working Paper 89–276*, Department of Management, Terry College of Business, The University of Georgia, Athens, GA, 1990a.

ELNIDANI, M. A. AND J. E. ARONSON, The Multicommodity, Multiperiod Assignment Problem II: Theoretical Results, *Working Paper 89–277*, Department of Management, Terry College of Business, The University of Georgia, Athens, GA, 1990b.

ELNIDANI, M. A. AND J. E. ARONSON, The Multicommodity, Multiperiod Assignment Problem III: Variations for Facility Location and Personnel Planning, *Working Paper 89–278*, Department of Management, Terry College of Business, The University of Georgia, Athens, GA, 1991.

FRANZ, L. S. AND J. L. MILLER, Scheduling Medical Residents to Rotations: Solving the Large-scale Multiperiod Staff Assignment Problem, *Operations Research*, **41**, 2 (1993), 269–279.

GENDRON, B. AND T. G. CRAINIC, Parallel Branch-and-Bound Algorithms: Survey and Synthesis, *Operations Research*, **42**, 6 (1994), 1042–1066.

GILBERT, K. C. AND R. B. HOFSTRA, An Algorithm for a Class of Three-Dimensional Assignment Problems in Scheduling Applications, *Institute of Industrial Engineers Transactions*, **19** (1987), 29–33.

GILBERT, K. C. AND R. B. HOFSTRA, Multidimensional Assignment Problems, *Decision Sciences*, **19** (1988), 306–321.

GILBERT, K. C. AND R. B. HOFSTRA, A New Multiperiod Multiple Traveling Salesman Problem with Heuristic and Application to a Scheduling Problem, *Decision Sciences*, **23** (1992), 250–259.

GUPTA, B., A Parallel Multiperiod Assignment Algorithm, *Doctoral Dissertation*, Terry College of Business, The University of Georgia, Athens, GA, 1995.

HOFSTRA, R. B., "A New Multiperiod Multiple Traveling Salesman Problem with Heuristic and Application to a Scheduling Problem," *Proceedings of the Annual Decision Sciences Institute Southeast Regional Meeting* (February 21–23, 1990).

KENNINGTON, J. L. AND Z. WANG, "Solving Dense Assignment Problems on a Shared Memory Multiprocessor," *Technical Report 88–OR–16*, Department of Operations Research and Engineering Management, School of Engineering and Applied Sciences, Southern Methodist University, Dallas, TX (October 1988).

KINDERVATER, G. A. P. AND J. K. LENSTRA, Parallel Computing in Combinatorial Optimization, *Annals of Operations Research*, **14**, 4 (1988), 245–289.

PIERSKALLA, W. P., The Multidimensional Assignment Problem, *Operations Research*, **16**, 2 (1968), 422–431.

SUNDERAM, V. S., PVM: A Framework for Parallel Distributed Computing, *Concurrency: Practice and Experience*, **2**, 4 (1990), 315–339.

SUNDERAM, V. S. AND G. A. GEIST, The PVM System: Supercomputer-Level Concurrent Computation on a Network of IBM RISC System/6000 POWER-stations, Reprinted from *Scientific Excellence In Supercomputing*: The 1990 IBM Contest Prize Papers, Volume 2, Billingsley, K., H. Brown and E. Derohanes (eds.), The Baldwin Press, The University of Georgia, Athens, GA, 1992, 779–804.

Received: June, 1995
Accepted: May, 1997

Contact address:
Babita Gupta
Assistant Professor of Management
Institute for Management
and International Entrepreneurship
California State University–Monterey Bay
Seaside, CA 93955
USA
Phone: 408/582-4186
Fax: 408/582-3585
Email: babita@seal.monterey.edu

Jay E. Aronson
Department of Management
Terry College of Business
The University of Georgia
Brooks Hall
Athens, GA 30602-6256
USA
Phone: 706/542-0991
Fax: 706/542-3743, 706/542-7196
Email: jaronson@blaze.cba.uga.edu

BABITA GUPTA is Assistant Professor of Management Information Systems at California State University — Monterey Bay. Previously she was an Assistant Professor of Business and Economics at Missouri Western State College. She earned her Ph.D. in Management Sciences from The University of Georgia. Her research interests include parallel network optimization, tabu search, expert systems and others.

JAY E. ARONSON is Associate Professor of Management at Terry College of Business at The University of Georgia. Previously, he was Associate Professor of MSIT at UGA, and Assistant Professor at Southern Methodist University. He earned his Ph.D. in Industrial Administration from Carnegie Mellon University. He has published in major journals on neural computing, executive information systems, group support systems, systems analysis and design, and parallel network optimzation.