PARALLEL SIMPLEX FOR LARGE PURE NETWORK PROBLEMS: COMPUTATIONAL TESTING AND SOURCES OF SPEEDUP

Richard S. Barr[1]
Betty L. Hickman[2]

March, 1993
(Revised)

[1]*Southern Methodist University*
Department of Computer Science and Engineering
Dallas, Texas 75275
(214) 768-2605
barr @ seas.smu.edu

[2]*University of Nebraska at Omaha*
Department of Mathematics and Computer Science
Omaha, Nebraska 68182-0243
(402) 554-2831
hickman @ unocss.unomaha.edu

ABSTRACT

This paper reports on a new parallel implementation of the primal simplex method for minimum cost network flow problems, that decomposes both the pivoting and pricing operations. The self-scheduling approach is both flexible and efficient; its implementation is close in speed to the best serial code when using one processor, and is capable of substantial speedups as parallel computing units are added. An in-depth computational study of randomly generated transportation and transshipment problems verified the effectiveness of this approach, with results on a 20-processor 80386-based system that are competitive with—and occasionally superior to—massively parallel implementations using tens of thousands of processors. A micro-analysis of the code's behavior identified unexpected sources of (the occasionally superlinear) speedup, including the evolutionary topology of the network basis.

The past two decades have seen dramatic advances in the development and implementation of algorithms for solving the minimum-cost network flow problem. Large-scale models that previously required days to optimize using general linear programming methods now require only hours with a special-purpose network code. Even with such achievements, many applications require more rapid results, since the value of a model's solution may be either short-lived—as with stock arbitrage opportunities—or otherwise time-sensitive—as with evacuation models, aircraft rerouting; missile targeting, or human interaction problems (see Ahuja, Magnanti, and Orlin, 1993; Aronson, 1989; Bertsekas, 1991; Glover, Klingman, and Phillips, 1989, 1992; Murty, 1992). Large pure network models also arise as subproblems in the context of integer, nonlinear, relaxation, and stochastic optimization algorithms (see survey by Amini and Barr, 1990). In many cases, such as Lagrangean relaxation, algorithms require that subproblems be solved sequentially, hence time improvements must come from faster solutions to individual problems. Such needs have led researchers to explore advances in computing technology for new means of further reducing the real time to optimize a program.

One promising advance is application-level parallel processing, whereby the power of multiple processors can be brought to bear on a single problem. If the work associated with an algorithm can be properly subdivided and scheduled to separate processors for simultaneous execution, opportunities for dramatic new model solution times arise. As with traditional, serial machines, solution efficiencies are directly tied to how well the algorithmic steps match the architecture of the underlying machine. Therefore, with the evolution in computing machinery comes a corresponding evolution in algorithms and their implementations.

The objective of this research was to design and implement a new primal-simplex-based parallel algorithm for efficiently solving the minimum-cost, or *pure*, network flow problem and test its performance on medium- and large-scale problems. The results, obtained on a shared-memory multiprocessor, not only show that substantial improvements in solution time are indeed possible but also that such improvements are due in great measure to temporal characteristics of the basis topology.

The sections that follow give a brief background on parallel processing and network flow problems; describe our parallel algorithm, its implementation, and the results of computational testing; and analyze the evolutionary characteristics of the simplex basis.

1. BACKGROUND

Parallel processing is the simultaneous manipulation of data by multiple computing elements working to complete a common body of work. The key objective of parallel processing is the reduction of real (“wall clock”) time required to complete the work. Hence the motivation for such new

machine architectures springs from the need to solve existing problems faster or to make tractable larger and more difficult problems.

The most common measure of the effect of parallelism on the solution of a given problem is *speedup*, S(p). While several definitions of speedup have been proposed (see Barr and Hickman, 1993a), we use: the ratio of the problem's solution time using the fastest serial code to the time using a parallel code and p processors on the same machine. (Some authors simply report *relative speedup*, which compares the parallel code with itself using one processor.) *Linear speedup*, with $S(p)=p$, is considered an ideal application of parallelism, although *superlinear* results, with $S(p)>p$, are possible in some instances.

1.2 Parallel Computer Architectures

While most computers have some degree of parallelism, only recently have systems become commercially available that allow an applications programmer to control several processing units. Such parallel computers are as varied in design as they are many, but the two main categories—as defined by Flynn (1966)—are: single-instruction, multiple-data (SIMD), a parallel machine design wherein all processors execute the same instruction in lockstep, and apply it to different pieces of data; and multiple-instruction, multiple-data (MIMD), a computing system containing multiple, independently executing processors which can operate on different datasets.

Processors in SIMD and MIMD parallel computers communicate either via a common shared memory accessed through a central switch, or by messages passed through an interconnection network in a *distributed* system. Shared-memory multiprocessors are called *tightly coupled* if the time required to access a particular memory location is the same for all processors, as opposed to being proximity dependent or *loosely coupled*. Our research was carried out in a shared-memory MIMD multiprocessing environment, the most prevalent commercial parallel computer architecture.

Since automatic parallelization of serial programs is in the embryonic stage of development, implementations of parallel algorithms must be coded for a parallel processing environment. Work must be decomposed into a series of tasks which may be assigned to separate processors for simultaneous execution. Our parallel network algorithm was implemented using both functional and domain decomposition (see Osterhaug, 1992), with a prioritized, self-scheduled synchronization and work allocation scheme, as described in Section 3.

1.3 Previous Parallel Research on Pure Network Problems

Two previous parallel implementations of the primal simplex for pure networks have been reported. Miller, Pekny, and Thompson (1990) used a 14-processor BBN Butterfly Plus—a tightly coupled, heterogeneous MIMD computer—to solve large dense uncapacitated transportation problems. They execute only the simplex pricing step in parallel. Peters (1990) performed in parallel the

pricing step and parts of the pivot operation; decomposition of the pivot was deemed "not competitive." His code was tested on a Sequent S81, hence direct comparisons are possible with our implementation and are described below.

Bertsekas and Castañon (1990) implemented Ford and Fulkerson's (1957) primal-dual method for pure networks, based on earlier ideas by Balas, et al. (1989) and their own work for assignment problems. This was implemented and tested on an Encore Multimax using one to four processors.

Li and Zenios (1991) implemented an ε-relaxation network algorithm on a massively parallel Connection Machine CM-2 with 16,384 processors. Nielsen and Zenios (1991) applied two different algorithms—quadratic proximal point (QPP) and entropy proximal point (EPP)—to large pure networks on the loosely coupled SIMD CM-2 with 32,768 processors. In these last two papers, test problems were used that are equivalent to some we tested, so that comparisons can be made.

2. SOLVING THE PURE NETWORK FLOW PROBLEM

2.1 Problem Statement

The minimum cost network flow problem, PN, may be formulated mathematically as follows:

PN: Minimize $\sum_{(i,j)\leftrightarrow A} c_{ij}\, x_{ij}$ (1)

subject to: $\sum_{j:\,(i,j)\leftrightarrow A} x_{ij} - \sum_{j:(j,i)\leftrightarrow A} x_{ji} = b_i\,, \qquad \text{for all } i \leftrightarrow N\,,$ (2)

$0 \le x_{ij} \le u_{ij}\,, \qquad \text{for all } (i,j) \leftrightarrow A\,,$ (3)

where A is the set of all arcs and N is the set of all n nodes in the network. Associated with each arc (i,j) is a variable x_{ij} representing the number of units of flow through the arc from node i to node j, and the constants c_{ij} and u_{ij} representing, respectively, the cost per unit of flow and upper bound. Associated with each node i is a dual variable π_i, the *node potential* and b_i, the *requirement* at node i, called the *supply* at *source node i* if positive or *demand* at *sink node i* if negative.

2.2 Primal Simplex Algorithm for Pure Networks

The most widely used method for solving PN is a specialization of the primal simplex algorithm which capitalizes on the triangularity property of the problem's bases. In this setting, a simplex basis corresponds to a spanning tree on the nodes. The set B denotes such a set of $n-1$ *basic arcs*, and L and U respectively denote the sets of *nonbasic arcs* with flow at zero and their upper bounds. If the assignment of flows denoted by the triplet (B,L,U) satisfies (2) and (3), it is termed a *basic feasible so-*

lution. The *reduced cost*, $\bar{c}_{ij} = c_{ij} + \pi_i - \pi_j$, of each basic arc (i,j) must equal zero. A feasible basis is also an *optimal basis* if it is possible to identify a set of π_i such that:

$$\bar{c}_{ij} = 0, \qquad \text{for each arc } (i,j) \in B, \tag{4}$$

$$\bar{c}_{ij} \geq 0, \qquad \text{for each arc } (i,j) \in L, \text{ and} \tag{5}$$

$$\bar{c}_{ij} \leq 0, \qquad \text{for each arc } (i,j) \in U. \tag{6}$$

It is useful to organize a network basis B as a *rooted* spanning tree, by selecting one node r to be the *root*, with the remainder of the tree hanging below it. Each pair of nodes i and $j \in N$ has a unique connecting path in the basis tree, $P(i,j)$; every node k such that $i \in P(k,r)$ is called a *successor* of i. The basis subtree containing node i and all of its successors is $T(i)$.

The network primal simplex algorithm proceeds as follows.

- *Step 0: Initialization*. A basic feasible solution, (B,L,U), is identified, possibly containing artificial arcs, and a set of dual values, π, are determined from (4).
- *Step 1: Pricing.* A nonbasic arc (k,l) which violates (5) or (6) is selected to be the *incoming arc*. If no such arc exists, the algorithm terminates with the current basis being optimal if it contains no artificial arcs with positive flow; otherwise, the problem is infeasible.
- *Step 2: Ratio test*. Adding arc (k,l) to the basis forms a unique *basis cycle*, consisting of (k,l) and $P(k,l)$, the *basis-equivalent path* of the incoming arc. The algorithm changes the flow in this cycle by δ, the maximum possible improving amount that does not violate the bound constraints (3). This step identifies a *blocking* or *outgoing arc* (p,q) which blocks further change in flow.
- *Step 3: Basis and dual updates*. The flows in the basis-equivalent-path arcs are adjusted by $\pm\delta$, depending on orientation. If the incoming arc is the blocking arc, it remains nonbasic, but changes *L/U* set membership. Otherwise, a *pivot* is performed whereby arc (k,l) becomes a member of B, arc (p,q) becomes a member of L or U, and a subset of the duals π are adjusted by a constant. In either case the algorithm returns to Step 1.

2.3 Implementing the Network Simplex Method

Our parallel network code was built from one of the fastest serial network codes, NETSTAR, written by Barr, which is a descendent of the ARC-II code of Barr, Glover, Klingman (1979). We will use this code to illustrate an efficient implementation of the network simplex algorithm in the serial environment.

The network basis is represented and maintained by the Extended Threaded Index (XTI) Method described in Barr, Glover, and Klingman (1979). The basis data consists of a set of labels associated with each problem node. For node $i \leftrightarrow N - r$, the predecessor label, *p(i)*, identifies the basic variable *(i,p(i))* or *(p(i),i)* whose flow is the label value x(i). The potential for node i is the label $\pi(i)$.

Only a portion of the node potentials must be updated during a pivot. Specifically, when the blocking arc is removed from the basis, two subtrees result and it is necessary to update the duals in only one of them. Four node labels, or functions, facilitate this time-consuming process. The *cardinality*, *s(i)*, of node $i \in N$ is the number of nodes in T(i) and is used to identify the smaller subtree. The set of *thread* labels, *t(i)*, may be viewed as placing a top-to-bottom, left-to-right ordering on the nodes and provides an efficient means of identifying all nodes in T(i) when used with the cardinality function. The *reverse thread* function, $\bar{t}(i)$, expedites the updating of the thread function and is defined so that $t(\bar{t}(i)) = i$. Finally, the *end-node* label, *e(i)*, identifies the last node in T(i), when considered in thread order, and facilitates the updating of the other labels. These labels are important for parallelization of the pivot operation, and we later statistically analyze the data structures to describe the evolving basis topology in a parallel setting.

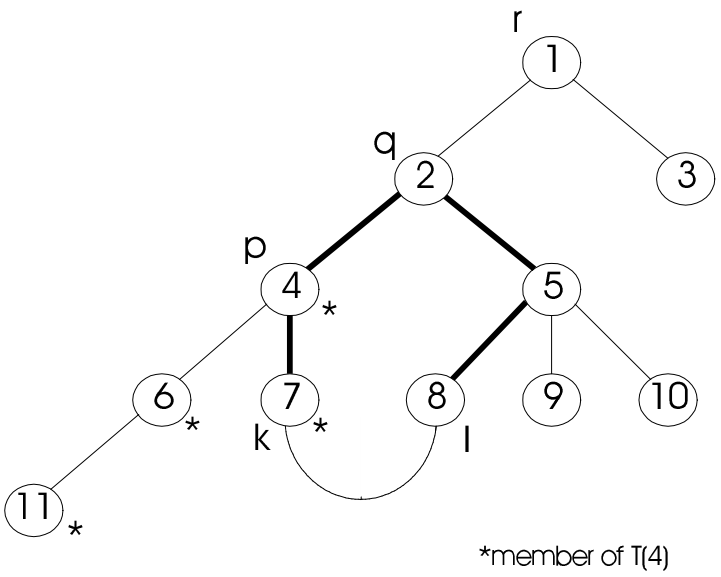
An example network basis and associated labels are shown in Figure 1. The flows are not given and the duals reflect a cost of five for all basic arcs. For the nonbasic incoming arc (k,l), the basis-equivalent path is emphasized. Note that, for the blocking arc (p,q), the potentials must be updated for either those nodes in T(p), flagged with a "*", or those not in T(p), unflagged.

3. PARALLEL CODE DESIGN AND IMPLEMENTATION

3.1 Design Issues

Since pricing and pivoting are the primary simplex operations, their concurrent execution would provide an excellent basis for any parallelization scheme. Unfortunately the operations are not independent, since pivoting modifies the duals used in the pricing procedure. In dealing with this dependency, our design used a single shared set of node potentials, rather than maintain multiple datasets. Since the number of potentials which change at each pivot is relatively small (ranging from 0.2% to 17% in our tests), the probability of selecting an incorrectly priced candidate arc is low. And since all candidates are repriced with correct duals prior to incoming arc selection, the pivoting operation is not compromised. Our parallel code simultaneously executes the pricing and pivoting steps.

Further parallelism is achieved by decomposing each of these operations. In the pricing step, the arc data is partitioned for multiple pricing units. The intensification of the pricing



Node i	Node Label $p(i)$	$\pi(i)$	$s(i)$	$t(i)$	$\bar{t}(i)$	$e(i)$
1	–	0	11	2	3	3
2	1	5	9	4	1	10
3	1	-5	1	1	10	3
4	2	0	4	6	2	7
5	2	10	4	8	7	10
6	4	5	2	11	4	11
7	4	5	1	5	11	7
8	5	5	1	9	5	8
9	5	15	1	10	8	9
10	5	15	1	3	9	10
11	6	0	1	7	6	11

Figure 1. Example Network Basis and Node Labels

effort with multiple processors should produce better candidates for basis entry and accelerate progress towards optimality.

Decomposition of the pivot operation is more involved. A pivot consists of three steps: the ratio test, the basis update, and the dual update. Studies of our serial code indicated that the total time spent pivoting during the solution of a problem is roughly divided among these steps as follows: ratio test, 17%; basis update, 23%; and dual update, 60%. Clearly decomposition of the dual update would have the greatest potential for reducing the real time spent pivoting.

Our approach permits division of the dual update step into two tasks, each resetting half of the nodes in the appropriate (smaller) subtree. The XTI data structure made this decomposition practical. For a given subtree, say T(i), a first task updates the $s(i)/2$ potentials found in thread order beginning at node i; the remaining duals are updated by a "delegated" second task that starts at node $e(i)$ and proceeds in reverse-thread order. This was found to be beneficial if the subtree was of sufficient size.

3.2 Parallel Network Simplex Implementation

An elegant and efficient means of synchronizing parallel processes is via a programming construct called a *monitor* (Hoare, 1974). A monitor is a self-scheduled work allocation scheme that consists of: (1) a critical section of code (i.e., one which can be executed by only one processor at a time); (2) a shared work list, accessible only within the critical section; and (3) a delay queue for idle processes. Access to the critical section is controlled by an associated lock. Idle processes enter the common critical section one-at-a-time and update the work list, select a task from the list, exit the section, perform the task, and return for additional work. If work is not available from the monitor, the process is placed in a delay queue, to be released when additional tasks become available. Termination occurs when all processors are in the delay queue. Note that this design permits participation of one or many processors.

The steps of our monitor-based parallel algorithm for the network simplex are given informally in Figure 2. Note that this logic is executed concurrently by all processes participating in solving the problem, although only one may be in the critical section at a given point in time. With only a single process, the procedure alternates between pricing and pivoting tasks.

Such a self-scheduled approach is effective when the number and time requirements of tasks vary widely or are unknown, hence was particularly apropos for our network simplex implementation. Our design maintains both an implicit work list of tasks and a candidate list as shared datasets accessible only within a monitor, with all problem and basis data in shared, globally accessible memory.

```
algorithm Parallel_Network_Simplex
begin
    NewDuals = true      {Boolean, shared, global variable}
    loop forever
        Enter monitor              {Begin critical section}
        if last task was pricing and favorably priced arc was found then
            Add arc to candidate list
        endif
        if previous pivot is complete then
            if NewDuals then
                Reset arc pricing list
                NewDuals = false
            endif
            Reprice candidate list
            if favorable arc is found on candidate list then
                Assign pivot as task, with most favorable arc as incoming
                NewDuals = true
            else
                if unpriced arc group remains then
                    Assign arc group for pricing as task
                else
                    Enter delay queue
                    if all processes delayed then exit loop
                endif
            endif
        else          {previous pivot is in progress}
            if parallel dual update was requested by pivot task then
                Assign dual update as task
            else
                if unpriced arc group remains then
                    Assign arc group for pricing as task
                else
                    Enter delay queue
                    if all processes are delayed then exit loop
                endif
            endif
        endif
        if work is available and a process is delayed then
            Release all processes from the delay queue
        endif
        Exit monitor          {End critical section}
        Perform assigned task
    endloop
    stop    {Solution optimal or problem infeasible}
```

Figure 2. Algorithm for Parallel Network Simplex Monitor

Tasks scheduled by the monitor are: (1) select an eligible arc from the candidate list and perform a pivot; (2) perform any delegated portion of the dual update; and (3) price a group of arcs and return the most pivot-eligible, if any, to the candidate list. Since, in this implementation, pivots cannot be executed concurrently and only a portion of each pivot may be decomposed, this operation tends to be the bottleneck in a parallel environment. Hence priorities were established to minimize the time between pivots. Pivot execution is given the highest priority and, with pivot completion depending on updated node potentials, secondary priority is given to any delegated dual update task; pricing is given the lowest priority.

A parallel pure network code, PPNET, implementing the above algorithm was constructed from the NETSTAR code's network simplex routines. Figure 3 illustrates the parallel network-simplex monitor's operation and participating processes' data usage.

Performance of the code is strongly affected by the pricing and pivot decomposition strategies. Using a parallel variant of the Mulvey (1978) *candidate list* approach, each pricing task consists of pricing all arcs leaving m_1 nodes, and identifying the most attractive one, if any; such an arc replaces any less attractive nonbasic on the m_2-length candidate list. Decomposition of the dual update portion of the pivot is performed only if the subtree to be updated is of sufficient size to justify incurring the associated overhead. This minimum subtree size was a third user-specified parameter, m_3. Hence a *pricing and pivoting strategy* is specified by the triplet (m_1, m_2, m_3).

The other user-specified parameter is the "Big-M" value assigned to artificial arcs in the all-artificial initial basis. Computational testing shows that the smallest value for M that still eliminates all artificials is superior to a larger M value or "Phase I-II" approach. Our implementation used M=4(max$\{c_{ij} \mid (i,j) \in A\}$), so as to gradually drive artificials from the basis as a natural byproduct of the pivoting process.

In the computational testing reported below, substantial effort was invested to determine reasonable values for the pricing and pivot decomposition parameters, a task complicated by the fact that "good" values vary not only with problem characteristics but also with the number of processors used to solve a given problem. Rather than specify parameter values for each test problem individually, they were computed using simple rules or fixed to values that were observed to work reasonably well on a variety of test problems. The rules are based solely on the number of processors and are therefore problem independent. Although this approach yielded inferior results on some individual cases, it was used in order to avoid tuning the code to each problem. Of course, dramatically better times are possible with such tuning.

4. COMPUTATIONAL TESTING

PPNET was tested on medium-and large-scale problems, as would be encountered in practical applications. All problems were generated using the most current version of the

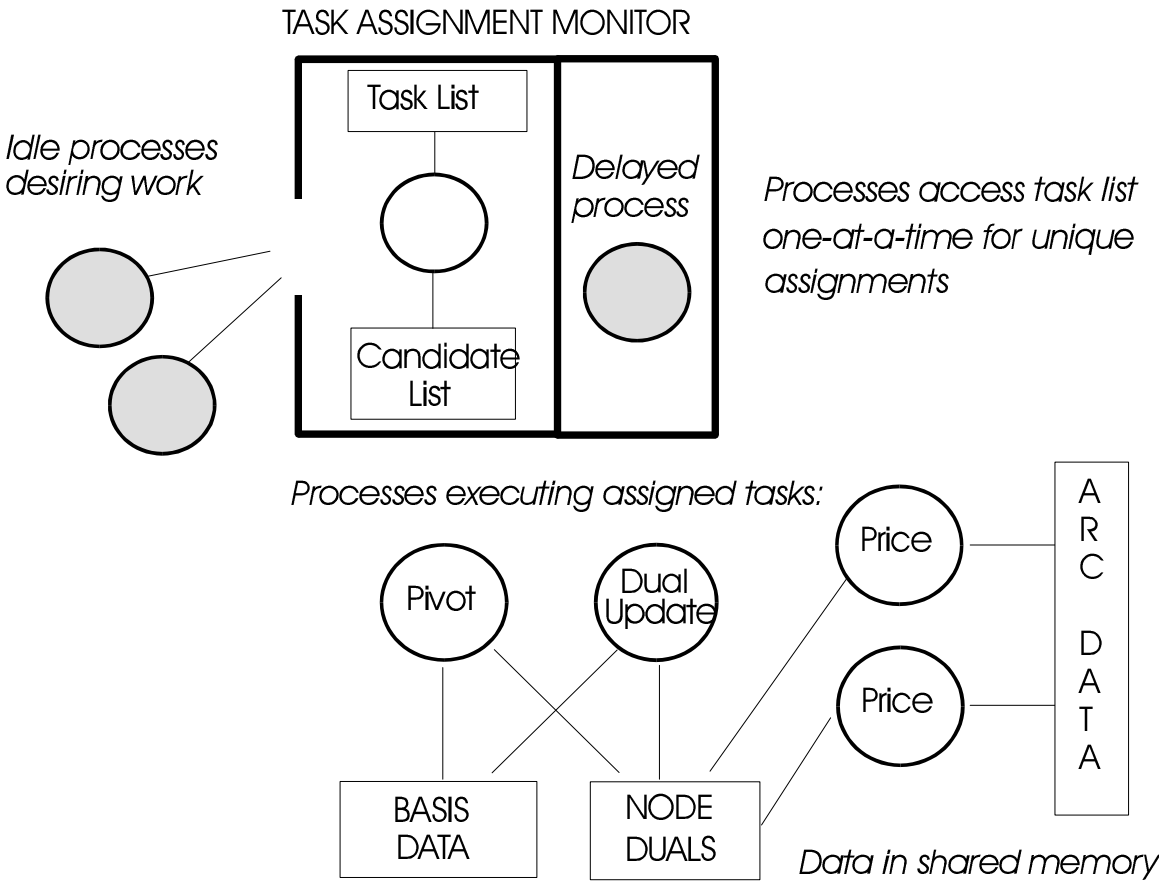


Figure 3. Parallel Network Simplex Monitor & Data Structures

NETGEN random problem generator by Klingman, Napier, and Stutz (1974). The medium-scale set consists of 50 problems defined in a NETGEN problem suite designed by Klingman and Mote (1987), as described in Table I. (Although this problem suite has been used by numerous researchers, it has not been previously documented in a technical report or publication.)

Testing was performed at Southern Methodist University on a Sequent Symmetry S81, a tightly coupled MIMD machine with 20 16-MHz Intel 80386 processors and 32 megabytes of sharable memory. Due to a system upgrade during this research project, we were able to test PPNET on both the Sequent Symmetry "Rev. A" and the upgraded "Rev. B." The primary difference between systems is in their memory updating schemes for maintaining cache coherency (see Dubois, Scheurich, and Briggs, 1988), that have the effect of increasing individual processor throughput from three to four million instructions per second. This permitted computational comparisons with an earlier paper and to note the effect, if any, of processor speed on speedup.

4.1 Problem Set A: Medium-Scale Problems

Problem set A consisted of 17 problems from the Klingman and Mote (1987) NETGEN problem suite described in Table I, selected for the availability of comparable parallel results by Peters (see Table II). The test set consists of transportation and transshipment problems typically having 5000 nodes and 25,000 arcs, with a variety of cost and capacitation ranges, total supply, etc.

These problems were solved on a dedicated Rev. A system using one to ten processors. The real, or "wall clock" solution times were recorded, exclusive of input and output time. For all problems, PPNET used the pricing strategy $m_1=\max\{40/p,5\}$, $m_2=5$, and $m_3=80/p$, where p is the number of participating processors. The resulting one-and three-processor results are shown in Table II.

To provide benchmarks with which to compare our one-processor results, test set A was also solved using the well-known serial code NETFLO (Kennington and Helgason, 1980) and NETSTAR. The NETFLO times are included in Table II—as a point of reference—and are seven to eight times longer than the serial PPNET code.

One-processor PPNET times were used in problem set A testing as the best serial case for the computation of speedups. (Testing performed on the Rev. A machine indicated that NETSTAR and PPNET results were indistinguishable. Testing on the updated Rev. B machine indicated that NETSTAR was somewhat faster than the one-processor PPNET code, hence NETSTAR times were used for speedup computation in all Rev. B testing.) The mean speedups from Table II show that, on average, PPNET achieves near-linear speedup on three processors. The occasional, and surprising, superlinear speedups are explored in section 5.

Prob.	No.	No.	No.	No.	Arc Costs		Total	Transshipment		%Hi		% Capacity		Random	Objective
No.	Nodes	Source	Sinks	Arcs	Min	Max	Supply	Src	Sinks	Cost	Cap	Min	Max	No. Seed	Function
101	5000	2500	2500	25000	1	100	250000	0	0	0	100	1	1000	13502460	6191726
102	5000	2500	2500	25000	1	100	2500000	0	0	0	100	1	1000	4281922	72337144
103	5000	2500	2500	25000	1	100	6250000	0	0	0	100	1	1000	44820113	218947553
104	5000	2500	2500	25000	-100	-1	250000	0	0	0	100	1	1000	13450451	-19100371
105	5000	2500	2500	25000	101	200	250000	0	0	0	100	1	1000	14719436	31192578
106	5000	2500	2500	12500	1	100	125000	0	0	0	100	1	1000	17365786	4314276
107	5000	2500	2500	37500	1	100	375000	0	0	0	100	1	1000	19540113	7393769
108	5000	2500	2500	50000	1	100	500000	0	0	0	100	1	1000	19560313	8405738
109	5000	2500	2500	75000	1	100	750000	0	0	0	100	1	1000	2403509	9190300
110	5000	2500	2500	12500	1	100	250000	0	0	0	100	1	1000	92480414	8975048
111	5000	2500	2500	37500	1	100	250000	0	0	0	100	1	1000	4230140	4747532
112	5000	2500	2500	50000	1	100	250000	0	0	0	100	1	1000	10032490	4012671
113	5000	2500	2500	75000	1	100	250000	0	0	0	100	1	1000	17307474	2979725
114	5000	500	4500	25000	1	100	250000	0	0	0	100	1	1000	4925114	5821181
115	5000	1500	3500	25000	1	100	250000	0	0	0	100	1	1000	19842704	6353310
116	5000	2500	2500	25000	1	100	250000	0	0	0	0	1	1000	88392060	5915426
117	5000	2500	2500	12500	1	100	125000	0	0	0	0	1	1000	12904407	4420560
118	5000	2500	2500	37500	1	100	375000	0	0	0	0	1	1000	11811811	7045842
119	5000	2500	2500	50000	1	100	500000	0	0	0	0	1	1000	90023593	7724179
120	5000	2500	2500	75000	1	100	750000	0	0	0	0	1	1000	93028922	8455200
121	5000	50	50	25000	1	100	250000	50	50	0	100	1	1000	72707401	66366360
122	5000	250	250	25000	1	100	250000	250	250	0	100	1	1000	93040771	30997529
123	5000	500	500	25000	1	100	250000	500	500	0	100	1	1000	70220611	23388777
124	5000	1000	1000	25000	1	100	250000	1000	1000	0	100	1	1000	52774811	17803443
125	5000	1500	1500	25000	1	100	250000	1500	1500	0	100	1	1000	22492311	14119622
126	5000	500	500	12500	1	100	125000	500	500	0	100	1	1000	35269337	18802218
127	5000	500	500	37500	1	100	375000	500	500	0	100	1	1000	30140502	27674647
128	5000	500	500	50000	1	100	500000	500	500	0	100	1	1000	49205455	30906194
129	5000	500	500	75000	1	100	750000	500	500	0	100	1	1000	42958341	40905209
130	5000	500	500	12500	1	100	250000	500	500	0	100	1	1000	25440925	38939608
131	5000	500	500	37500	1	100	250000	500	500	0	100	1	1000	75294924	16752978
132	5000	500	500	50000	1	100	250000	500	500	0	100	1	1000	4463965	13302951
133	5000	500	500	75000	1	100	250000	500	500	0	100	1	1000	13390427	9830268
134	1000	500	500	25000	1	100	250000	500	500	0	100	1	1000	95250971	3804874
135	2500	500	500	25000	1	100	250000	500	500	0	100	1	1000	54830522	11729616
136	7500	500	500	25000	1	100	250000	500	500	0	100	1	1000	520593	33318101
137	10000	500	500	25000	1	100	250000	500	500	0	100	1	1000	52900925	46426030
138	5000	500	500	25000	1	100	250000	500	500	0	100	1	50	22603395	60710879
139	5000	500	500	25000	1	100	250000	500	500	0	100	1	250	55253099	32729682
140	5000	500	500	25000	1	100	250000	500	500	0	100	1	500	75357001	27183831
141	5000	500	500	25000	1	100	250000	500	500	0	100	1	2500	10072459	19963286
142	5000	500	500	25000	1	100	250000	500	500	0	100	1	5000	55728492	20243457
143	5000	500	500	25000	1	100	250000	500	500	0	0	1	1000	593043	18586777
144	5000	500	500	25000	1	10	250000	500	500	0	100	1	1000	94236572	2504591
145	5000	500	500	25000	1	1000	250000	500	500	0	100	1	1000	94882955	215956138
146	5000	500	500	25000	1	10000	250000	500	500	0	100	1	1000	48489922	2253113811
147	5000	500	500	25000	-100	-1	250000	500	500	0	100	1	1000	75578374	-427908373
148	5000	500	500	25000	-50	49	250000	500	500	0	100	1	1000	44821152	-92965318
149	5000	500	500	25000	101	200	250000	500	500	0	100	1	1000	45224103	86051224
150	5000	500	500	25000	1001	1100	250000	500	500	0	100	1	1000	63491741	619314919

Table I. NETGEN Problem Suite (Klingman and Mote, 1987)

NETGEN Problem Number	1-Processor Times			3-Processor Times		3-Processor Speedups	
	NETFLO	PARNET	PPNET	PARNET	PPNET	PARNET	PPNET
101	672.94	na	65.30	58.97	26.38	1.11	2.48
104	569.45	na	67.55	65.33	25.13	1.03	2.69
106	308.40	na	39.61	29.98	14.01	1.32	2.83
110	363.59	na	39.98	28.19	13.99	1.42	2.86
115	486.53	na	76.38	50.11	22.89	1.52	3.34
116	473.47	na	65.01	65.99	25.30	0.99	2.57
117	249.05	na	39.41	24.54	12.14	1.61	3.25
121	668.34	na	88.48	81.62	35.23	1.08	2.51
122	629.91	na	79.91	72.28	32.84	1.11	2.43
126	357.32	na	52.38	31.63	16.82	1.66	3.11
130	411.14	na	54.97	32.26	16.35	1.70	3.36
134	71.32	na	64.94	23.94	15.42	2.71	4.21
138	1163.80	na	134.30	101.33	47.52	1.33	2.83
142	479.12	na	80.39	48.21	25.66	1.67	3.13
144	662.07	na	85.22	71.03	32.44	1.20	2.63
147	2436.37	na	164.16	116.40	64.42	1.41	2.55
150	855.85	na	84.93	71.62	34.49	1.19	2.46
Mean	638.75	na	75.47	57.26	27.12	1.41	2.90

na = not available

Table II. Solution Times and Speedups on Problem Set A

Our code was also compared with the multiprocessor code PARNET whose published three-processor times, shown in Table II, were also obtained on a Rev. A Symmetry (see Peters, 1990). (Because of its design, this code requires a minimum of two processors to operate.) Although the PARNET times included input and output operations, while ours did not, our testing of its binary problem-input method and abbreviated output report indicated that the input/output times were negligible in comparison with solution times. A comparison of three-processor times shows PPNET to be roughly twice as fast as PARNET.

4.2 Problem Set B: More Medium-Scale Problems

Problem set B consists of all 50 problems in the test suite, which were solved on a "Rev. B" machine by PPNET and NETSTAR. In all NETSTAR runs, the pricing strategy m_1=40, m_2=10 was employed. For all problems and all numbers of processors, the pricing strategy m_1=20, m_2=5, m_3=100 was implemented in the PPNET runs. Problems 137 and 138 required a 50% larger M value for feasibility.

Three PPNET runs using one to ten processors and three NETSTAR runs were made on each problem. The NETSTAR average times, the PPNET average speedups, and summary statistics are shown in Table III. Figure 4 summarizes this data with the high, low, and mean speedups across all set B problems. In general, the code's times continue to improve as processors are added, with a mean speedup of 4.44 on ten processors.

4.3 Problem Set C: Million-Variable Problems

The large-scale test set consists of five additional problems, each having one million arcs and from 10,000 to 50,000 nodes, randomly generated with NETGEN and the parameters in Table IV. Each problem was solved with PPNET using 5, 10, 15, and 19 processors and with NETSTAR. The same pricing strategies used on Set B were employed on this set.

To see the processor effect on this problem set, Figure 5 graphs the speedups versus number of processors for all instances tested, and Table IV shows both the serial solution times and the best parallel time and corresponding speedup. For these problems, the mean best speedup was 8.26, with an overall best of 14.41 and a minimum best of 4.75. On problem 4, a 50,000-node transshipment problem, solution time was reduced from over ten hours to just 42 minutes with the application of parallelism. Even the 20,000-constraint transportation problem 3, which exhibited the smallest speedup, resulted in a reduction of solution time from approximately 41 to 8.7 minutes, and million-arc problem 1 was solved in under five minutes with the application of parallelism.

On three of the five problems, the addition of processors improved solution time without exception. On problem 5 speedup continued to improve through 15 processors and then leveled off. On problem 4 however, the effect of additional processors topped out at ten, with

Prob. No.	Seconds P=1	Speedup P=2	P=3	P=4	P=5	P=6	P=7	P=8	P=9	P=10
101	55.08	1.80	2.40	3.54	5.14	4.86	5.12	5.05	4.78	4.61
102	72.13	1.56	1.98	2.95	4.75	4.52	4.51	4.40	4.33	3.96
103	88.39	1.64	2.19	3.12	4.45	4.49	4.50	4.47	4.23	3.97
104	56.92	1.97	2.39	3.73	5.32	4.89	5.27	5.11	5.09	4.97
105	56.11	1.81	2.30	3.64	4.88	4.67	5.04	4.86	4.69	4.41
106	32.40	1.70	2.72	3.63	3.73	3.80	3.39	3.13	2.94	2.75
107	75.89	1.93	2.20	2.83	5.50	4.26	5.62	5.63	5.42	5.27
108	131.96	1.60	2.53	2.97	5.30	4.08	6.17	6.33	6.83	7.12
109	166.29	2.28	2.69	3.05	4.27	3.47	5.66	6.72	7.57	7.78
110	33.25	1.67	2.72	3.95	4.06	4.33	3.70	3.55	3.20	3.04
111	76.76	1.97	2.19	2.93	5.41	4.41	5.60	5.95	5.68	5.48
112	102.19	2.05	2.51	3.01	5.13	3.48	5.82	6.22	6.43	6.62
113	149.63	2.19	2.94	3.10	4.93	3.70	6.49	7.86	8.02	8.37
114	111.22	3.31	5.30	7.13	8.88	7.85	11.11	12.69	14.11	14.23
115	63.89	2.20	3.15	3.51	6.21	4.99	6.53	6.68	6.42	6.31
116	54.53	1.79	2.15	3.28	4.86	4.61	5.01	4.89	4.73	4.55
117	32.79	1.77	2.94	4.30	4.57	4.63	4.25	4.05	3.65	3.26
118	69.87	1.79	2.14	2.87	4.90	3.85	5.62	5.29	5.44	5.26
119	106.14	1.59	2.43	2.77	5.24	3.78	5.48	5.74	6.10	6.20
120	148.91	1.84	2.71	2.99	4.36	3.57	5.86	6.58	7.17	7.42
121	73.27	1.54	2.38	3.47	3.71	3.60	3.49	3.29	3.03	2.86
122	67.00	1.64	2.33	3.64	4.12	4.22	4.31	4.24	4.06	3.74
123	79.18	1.62	2.53	3.53	4.47	4.34	4.46	4.65	4.41	4.25
124	90.64	1.49	2.07	2.97	3.69	3.52	4.02	3.65	3.84	3.78
125	123.63	1.59	2.36	3.08	3.65	3.57	3.73	3.81	3.52	3.53
126	44.17	1.76	2.81	3.50	3.69	3.64	3.43	3.32	3.19	2.86
127	109.84	1.54	1.96	3.05	4.64	4.26	4.98	5.11	5.00	4.98
128	140.49	1.59	1.80	2.49	4.29	3.48	4.68	5.03	4.83	4.80
129	217.53	1.64	1.92	2.37	4.41	2.91	4.74	5.33	5.37	5.69
130	45.93	1.71	2.86	3.51	3.48	3.91	3.51	3.17	3.06	2.96
131	97.05	1.61	2.08	3.18	4.66	4.20	5.35	5.18	5.36	5.27
132	119.47	1.72	2.15	2.90	4.84	3.85	4.91	5.52	5.30	5.53
133	159.52	1.80	2.24	2.71	4.54	3.57	5.29	6.05	6.10	6.27
134	53.14	2.51	3.84	4.72	6.28	5.47	7.09	8.27	8.49	8.94
135	55.79	1.79	2.23	2.85	4.63	3.67	4.70	5.11	4.86	5.50
136	104.45	1.69	3.27	4.55	4.84	4.88	4.82	4.65	4.40	3.96
137	143.60	2.03	3.75	5.03	5.21	5.19	4.64	4.61	4.34	4.20
138	113.71	1.68	2.53	3.76	3.84	3.78	3.55	3.41	3.23	2.99
139	96.14	1.45	2.25	3.34	4.29	3.96	4.06	3.96	3.74	3.63
140	84.36	1.51	2.19	3.66	4.39	4.27	4.27	4.23	4.12	3.81
141	66.00	1.75	2.43	3.89	4.67	4.48	4.76	4.66	4.72	4.46
142	67.27	1.76	2.61	3.94	4.88	4.83	5.10	4.77	4.52	4.40
143	52.72	1.80	2.65	3.72	4.89	4.55	4.79	4.49	4.71	4.19
144	70.71	1.58	2.33	3.42	4.11	4.26	4.21	4.43	4.22	4.15
145	76.10	1.64	2.27	3.30	4.06	3.74	4.13	4.10	4.11	3.86
146	75.20	1.57	2.18	3.48	4.48	4.48	4.54	4.62	4.30	4.22
147	137.91	1.43	2.32	4.12	5.31	5.30	5.14	4.74	4.48	4.27
148	91.14	1.25	2.06	3.38	4.29	4.49	4.07	3.81	3.70	3.41
149	78.13	1.69	2.37	3.34	4.17	4.03	3.97	3.97	3.88	3.94
150	72.42	1.69	2.35	3.07	3.50	3.45	3.63	3.67	3.46	3.49
Mean	89.82	1.77	2.51	3.47	4.24	4.68	4.90	5.02	4.98	4.91

Table III. Serial Times, Parallel Speedups for Problem Set B

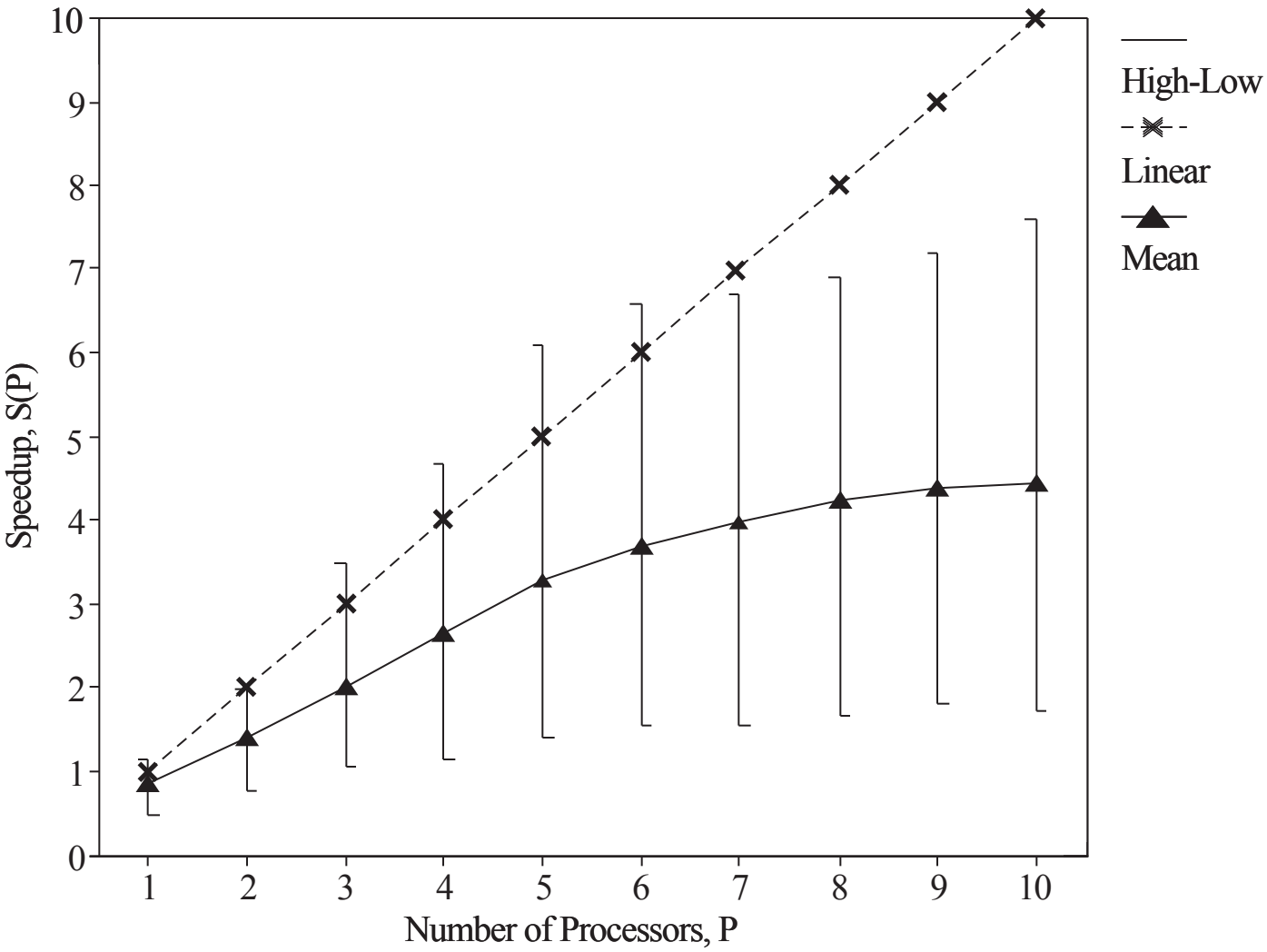


Figure 4. Test set B high, low, and mean speedups

Problem Characteristic[1]	Problem 1	2	3	4	5
Type[2]	TP	TS	TP	TS	TP
Nodes	10,000	20,000	20,000	50,000	50,000
Sources	5,000	4,000	10,000	10,000	25,000
Sinks	5,000	4,000	10,000	10,000	25,000
Arcs	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000
Supply	2,500,000	2,500,000	10,000,000	10,000,000	2,500,000
Cost range	1-100	1-100	1-100	1-10,000	1-100
Capacity range	1-1000	1-1000	1-1000	1-500	1-1000
% Capacitated	100	100	0	100	100
Seed	13502460	75578374	13502460	63491741	13450451
NETSTAR Time (sec)	1448.14	3629.13	2489.89	37166.6	9113.21
Best Parallel Time (sec)	298.76	573.76	526.12	2586.05	834.58
Best Speedup	S(19)=4.96	S(19)=6.22	S(19)=4.75	S(10)=14.41	S(15)=10.95

Table IV. Problem Set C Specifications and Test Results

the superlinear results up through ten processors. The sources of this behavior are explored in Section 5.

4.4 Comparisons with Other Parallel Codes

Of the other codes, Peters' (1990) PARNET is the most similar in design to PPNET. Unlike our generic processes, his are assigned specific tasks: one process for pivoting with the rest for pricing. A minimum of two processors are required to operate, hence the code cannot execute serially. As indicated earlier, PPNET solved Problem Set A twice as fast as PARNET.

Miller, Pekny, and Thompson (1990) efficiently solve completely dense uncapacitated transportation problems through parallel pricing alone. By focusing on problems with a high ratio of arcs to nodes, the required pricing effort is emphasized over pivoting, which favors their code and machine design. Their relative speedups on a 14-processor BBN Butterfly ranged from about 3 on two 1000-node 250,000-arc problems to about 7 on 6000-node, 9-million-arc problems.

The Bertsekas and Castañon (1990) code is not competitive with the other approaches. Based on results from the two small NETGEN problems reported, although the serial times seem slow, the best speedup obtained was S(4)=1.82, with an average speedup of 1.72 for two to four processors.

The massively parallel codes of Li and Zenios (1991) and Nielsen and Zenios (1991) for transportation problems were tested on equivalents to problems 1, 3, and 5 from Problem Set C (the random number generator and seed values differed, but identical problem specifications were used). Their reported results and the comparable PPNET times are shown in Table V. Surprisingly, PPNET—running on a relatively inexpensive Sequent Symmetry (under $100,000 used)—was highly competitive with all massively parallel codes. PPNET outperformed the ε-relaxation approach with 32,768 processors and the QPP code, and was 36% slower than the EPP code with 16,384 processors on problem 5 but over twice as fast on problem 1.

4.5 Summary of Computational Testing

The computational testing clearly underscores the effectiveness of parallelism in solving medium- and large-scale transportation and transshipment problems with our approach. The code is both flexible and efficient; its implementation runs slightly slower than the best serial code when using one processor, but is capable of substantial speedups as parallel computing units are added. It also appears to be the fastest available parallel code for pure network problems.

Code	PPNET	Li and Zenios		Nielsen and Zenios	
Algorithm	Primal Simplex	ε-Relaxation		QPP	EPP
Computer	Sequent S81B	CM-2	CM-2	CM-2	CM-2
# Processors	20	32,768	65,536	16,384	16,384
Memory	32MB	1,000MB	2,000MB	512MB	512MB
List price	$0.6M	$2.5M	$5M	$1.25M	$1.25M
Time (sec):					
Problem 1	298.76	760.45	447.32[1]	2,850.18	726.42
Problem 3	526.12	956.26	562.51[1]	n.a.	n.a.
Problem 5	834.58	890.76	523.97[1]	1,018.32	613.34

[1]Estimated, not observed.

Table V. Comparative Results, Problem Set C

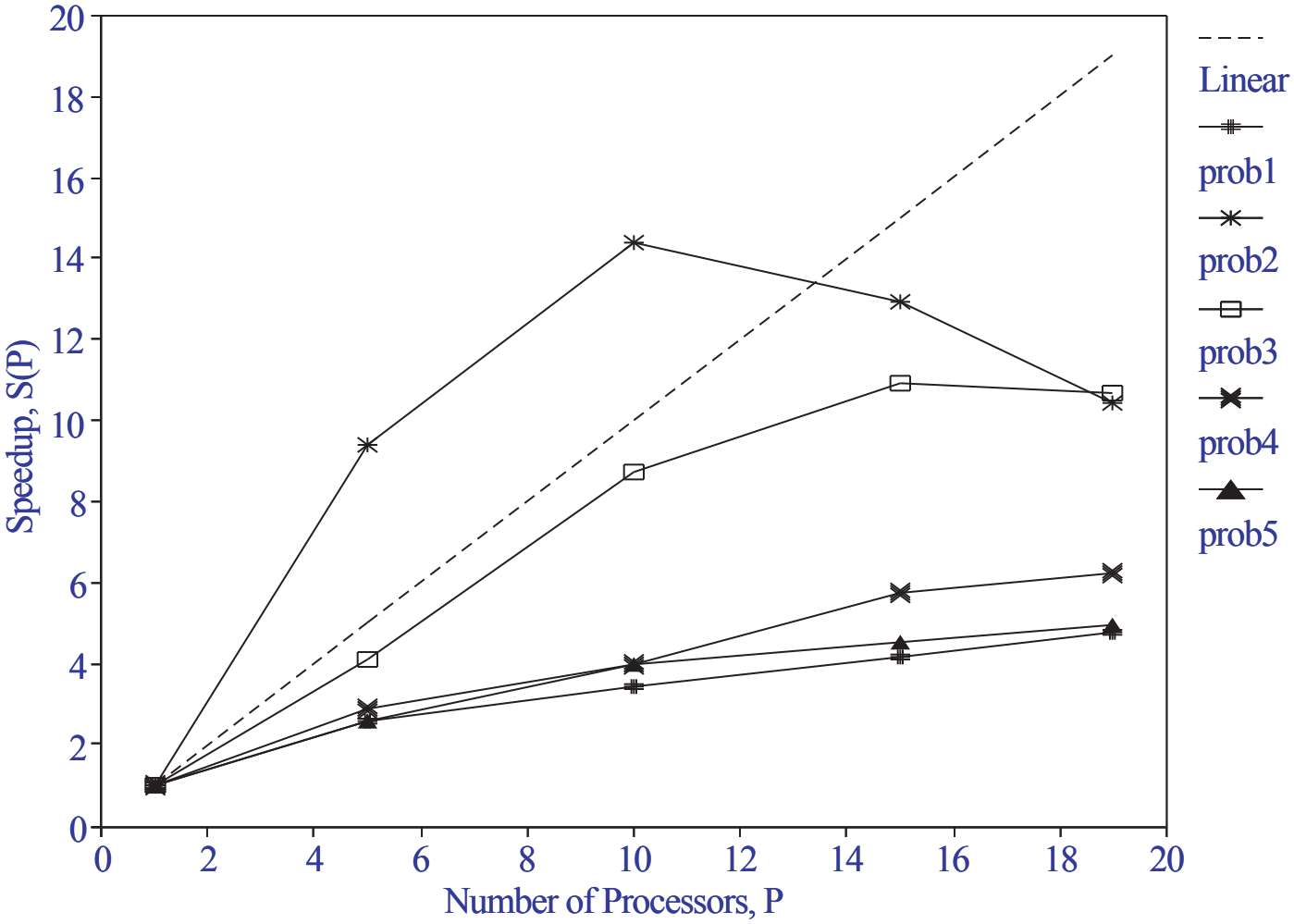


Figure 5. Test set C speedups by problem

5. AN EMPIRICAL STUDY OF THE ALGORITHM'S TEMPORAL BEHAVIOR

Superlinear results were encountered in each of the problem sets and, because the notion of superlinearity is controversial, we felt that an explanation for this behavior was required. To this end, we performed a micro-analysis of the code's performance, the behavior of individual algorithm steps, and the nature of the basis over the solution time. We were particularly interested in how the addition of processors would affect the various solution stages, and wished to answer such questions as: What are the sources of speedup? How is superlinear speedup possible?

The study uncovered three key speedup factors that resulted in occasional superlinearity; in addition, the statistics gathered provide insight into the temporal nature of network bases that has not been explored previously in the literature. Below we describe the data collection process, example statistics, and the model of algorithm behavior that emerged.

5.1 The Data Collection Process

Before undertaking the study, several measures of algorithm behavior were identified. In some cases intermediate calculations were required—whose inclusion in the code would affect its behavior—therefore the data was collected in stages. The first stage consisted of executing the program and recording each pertinent event and its time of occurrence (the overhead involved in this operation is negligible). Using this data, time-dependent statistics—such as the number of microseconds spent waiting for access to the monitor—were computed after the fact. In addition to logging events, the incoming arc for each pivot was recorded, permitting a second serial execution with the same pivot sequence to collect data regarding basis structure and pivot characteristics.

This process was applied to a variety of problems, all of which yielded similar results. While our findings are based on an analysis of a large number of problems, in the sections below we illustrate those findings with data collected during runs with one through ten processors on NETGEN problem 108, whose behavior was representative of all problems examined.

5.2 Temporal Basis Structure

One part of the study focused on the evolution of the network basis structure. With PPNET, an initial basis is constructed by connecting all problem nodes to an added root node with artificial arcs whose flow and orientation accommodate the node requirements. Hence, initially, the basis tree is wide and shallow. This structure is amenable to very fast pivoting since the paths to be traced are short and the number of duals to be updated is minimal.

As the basis evolves though pivoting, a more “mature” topology should develop whereby the basis tree becomes narrower and more vertical, thus increasing both the ex-

pected length of the basis-equivalent paths and the expected size of the updated subtree. We hypothesized that, as a result, pivoting should require increased processing over time, at least up to a point, and that the structure and parallel solution times are related.

Several metrics are used to characterize the basis topology. One such measure is the *mean cardinality of the basis tree*. As before, the cardinality, *s(i)*, of node *i* is the number of nodes in the subtree of which *i* is the root. The average of the cardinalities of all *n* nodes is an indicator of the "slenderness" of the basis tree. The code's initial basis has the smallest possible mean cardinality, $2 - 1/n$, indicating a "wide" or "bushy" basis tree, while the most slender basis possible has a mean cardinality of $(n+1)/2$.

Another statistic descriptive of the basis tree structure is the *number of singleton subtrees,* an indicator of the tree's "width" or "bushiness." The possible values range from the initial basis' maximum of $n-1$ singleton subtrees, indicating a wide tree, to a minimum of 1 for the most narrow tree possible.

Both the mean cardinality and the number of singleton subtrees in the basis tree were recorded before each pivot. The resulting data was partitioned into intervals of 1000 pivots and averaged over these intervals so as to give a representative view of the basis structure over time. These statistics for the one-processor case are shown as *Avg Card* and *1-Trees*, respectively, in Figure 6. The multiprocessor statistics are not presented, as they are virtually identical: the average correlation between the one-processor case and the nine multiprocessor cases is 0.9924 for mean cardinality and 0.9996 for number of singleton subtrees (see Barr and Hickman, 1990, for details).

Table VI shows, for problem 108, the number of pivots executed as a function of the number of processors. This can be used with Figure 6 to identify the work avoided by decreasing the number of pivots executed. For example, the vertical dashed line reflects the 12,925 pivots executed for ten processors and the corresponding final mean cardinality (72) and number of singleton subtrees (2,186).

The statistics clearly indicate that as the number of pivots increases the mean cardinality increases and the number of singleton subtrees decreases, until a plateau is reached, regardless of the number of processors. The similarity of values across different number of processors implies that *the structure of the basis at a given pivot number is relatively independent of the number of processors used*.

Another important inference to be made from this data concerns the increasing difficulty of pivots. The data indicate that the basis tree evolves from a wide, shallow shape to a more narrow, elongated form. With an increasing mean cardinality, the expected number of duals to be updated each pivot should increase as well. The expected length of the basis-equivalent path should also increase along with the effort required to perform the ratio test, flow updates, and subtree rerooting. Because of this evolving nature, *later pivots are hypothesized to be more difficult to perform than early pivots*, as explored in the next section.

5.3 Evolving Characteristics of the Pivot Steps

The effect that the evolution of the basis tree has on the relative difficulty of pivots may be described in terms of the three basic steps in pivoting: (a) the simultaneous identification of the basis-equivalent path and the application of the ratio test, (b) the updating of arcs in the basis-equivalent path, and (c) the updating of a subset of the duals.

5.3.1 Ratio Test

Pivot step (a) consists of identifying the basis-equivalent path (BEP) by tracing the basis tree from the endpoints of the incoming arc, and simultaneously performing the ratio test on each arc encountered. This comprises roughly 17% of the total pivot time with difficulty depending primarily on the length of the path. Degeneracy also plays a role since, once a degenerate pivot has been identified, the only remaining work is path tracing.

To measure the effort required to perform this step, the length of the BEP for each pivot was averaged over 1000-pivot intervals and the percentage of degenerate pivots in each interval computed. These values are shown for the serial case in Figure 6 as *BE Path Length* and *Degenerate Pivots*, respectively. As before, these same statistics for the parallel cases are nearly the same; the mean correlation between the serial and parallel instances is 0.9872 for the average path lengths and 0.9725 for the percentage of degenerate pivots.

In all cases of one through ten processors, the length of the BEP increases, then plateaus, with the number of pivots, indicating increasing tracing and ratio-test effort. Running counter to this trend is the increasing number of degenerate pivots and the corresponding decrease in pivot effort. Although PPNET contains specialized degeneracy logic, it saved only 3.05 seconds out of 165.57, with only 4616 degenerate pivots in the serial run. However, we shall shortly see that any savings gained because of degeneracy is far outweighed by other factors.

5.3.2 Path Update

The second portion of the pivot, the basis-equivalent path updates, accounts for approximately 23% of total pivot time. The effort required for these updates again depends on the BEP length; hence the above analysis leads us to the same conclusion: since the BEP length generally increases with the number of executed pivots, the amount of work per pivot also increases.

5.3.3 Dual Update

The most time-consuming portion of the pivot is the dual update, accounting for roughly 60% of total pivot time. If the blocking arc for a pivot is removed from the basis, two subtrees result, and the duals for the nodes in one of these subtrees must be updated. The update step consists of adding a constant to the node potentials associated with the smaller subtree, as de-

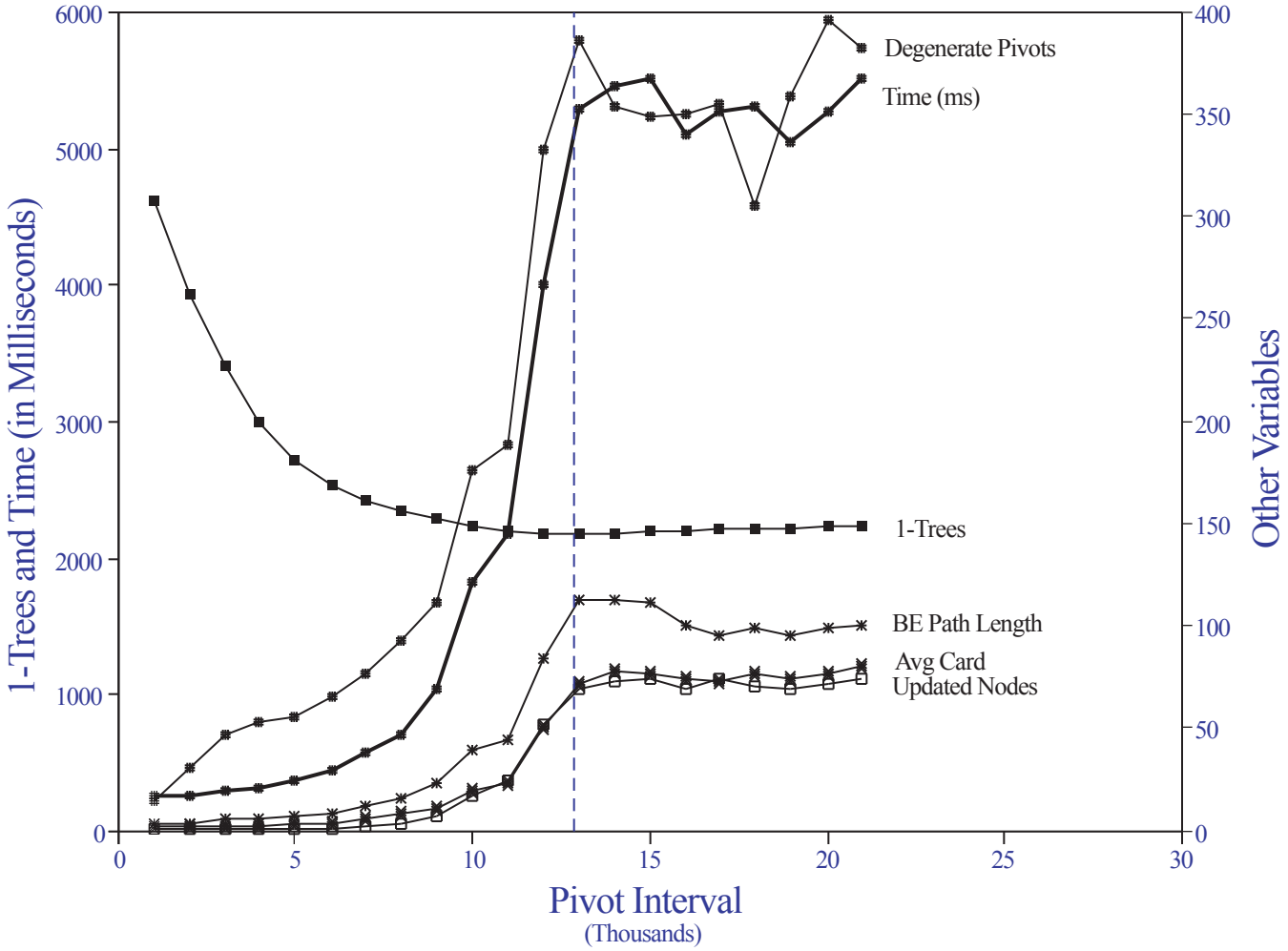


Figure 6. Evolving basis statistics, serial case

Number of Processors	Pivots Executed	Solution time (seconds)
1	21,378	165.597
2	19,622	87.239
3	17,781	56.846
4	16,136	44.290
5	15,048	35.094
6	14,745	31.372
7	13,853	26.044
8	13,761	24.932
9	13,216	21.532
10	12,925	20.230

Table VI. Problem 108 Statistics

termined from the cardinality function. Obviously the amount of work performed depends on the size of the updated subtree.

For our example, the number of nodes in the updated subtree was recorded for each pivot and averaged over 1000-pivot intervals. This statistic is shown as *Updated Nodes* in Figure 6 for the one-processor run; the average correlation between parallel and serial instances is 0.9927. The data indicates that the average size of the updated subtree grows with the number of pivots, then levels off. Again the effect of the maturing basis is seen as the pivots become more difficult as the basis evolves.

5.3.4 The Combined Effect

Although all our data, except for the number of degenerate pivots, indicates that pivots become more difficult as the basis matures, the composite effect is revealed in the time required to perform a pivot as the number of pivots increases. The total time spent performing pivots—summed over 1000-pivot intervals—are shown, in milliseconds, as *Time (ms)* in Figure 6 for the serial case. The mean correlation between the serial times and corresponding parallel times is 0.9907, indicating a strong linear relationship between the one-processor case and each of the multiprocessor cases (see Barr and Hickman, 1990, for detailed statistics).

The increase in the time required to perform a pivot as the number of pivots increases clearly illustrates the effects of the maturing basis. Generally speaking, the mean pivot time increases with pivot number, up to a point, irrespective of the number of processors involved. As with the average cardinality and number of singleton subtrees, the mean pivot times show an independence of algorithm behavior and the number of processors.

In contrast, the total number of pivots performed differs with the number of processors. The ten-processor solution required 8,453 fewer pivots than the one-processor case, and 145 seconds were required to execute those "extra" pivots. Since the structure of the basis is relatively independent of the number of processors, how can such a reduction be obtained?

5.4 Comparison of the Pricing Effort in the One- and Ten-Processor Runs

To answer this question, let us compare the number of arcs which are priced before each pivot in the one-and ten-processor runs. Our data indicated that, on average, it takes .00015 seconds to price all arcs leaving one node. If we examine the 11001-12000 pivot interval, in the ten-processor case it took 4.330 seconds to execute these 1000 pivots. Therefore one process can price 29 nodes during each pivot. Since nodes are priced in groups of 20, we may conservatively estimate that one process actually prices 20 nodes during each pivot. If we again underestimate the pricing effort and assume that two processes are constantly executing the pivot tasks, the remaining eight processors price 160 nodes during each pivot, compared with the 20 (or more if a candidate cannot be identified from 20 nodes) priced in the se-

rial case. Further analysis revealed that, beginning with the 6001-7000 pivot interval, more arcs were priced per pivot in the ten-processor run than in the one-processor run. Prior to this point, pivots are executed so quickly on the immature basis that there is insufficient time for a pricing process to complete its assigned work during only one pivot.

Clearly, if more arcs are examined in the process of identifying a candidate for basis entry, the likelihood of finding a more attractive candidate is increased. We have demonstrated that, given our pricing strategy, if a sufficient number of pricing processes are engaged, once the basis reaches a certain level of maturity more arcs will be priced than in serial execution. Testing also showed that an increased pricing effort reduces the total pivots required to reach optimality, a phenomenon observed in other empirical studies on serial machines (see Glover, et al., 1974, and Mulvey, 1978).

To quantify the importance of being able to price more arcs after the basis has begun to mature, a hybrid code was developed which ran with only one processor for the first v pivots, after which ten processors were utilized. With $v = 6000$, an average of 13,308 total pivots were performed, over 8000 less than the pure one-processor run, and almost equivalent to the pure ten-processor run. This indicates that the increased pricing effort is most significant when the basis is mature.

5.5 Pricing versus Pivoting Effect

The pricing strategy has a large effect on the performance of the algorithm. In the serial case, this strategy determines the amount of time spent pricing and thereby also determines the amount of time spent pivoting. The reduced cost of the entering arc is partially a function of the *pricing effort*, or the number of nonbasics considered. While increased pricing effort is likely to produce a more attractive candidate, spending too much time in the pricing phase may result in excessive solution times.

In the parallel case, the same tradeoff applies, and the number of arcs to be priced in order to identify a sufficiently attractive entering arc must be determined. Unlike the serial case, however, during execution with p processors, there are at least $p-2$ active pricing processes during each pivot (one process is executing the pivot and one may be executing the delegated dual update). Our goal is to use these $p-2$ processes to price some targeted minimum number of arcs *during each pivot*, so that when the pivot is complete, one or more attractive candidates have been identified, and the next pivot can begin without any delay. In this manner, the time spent pricing in a serial run is obtained virtually ``free'' in parallel, given enough processors.

To ascertain the performance of our strategy, we determined what fraction of the total solution time that a pivot was actually in progress. Since only one pivot is in progress at a given time, we may measure the total elapsed time spent in the pivot operation and compare this to the solution time, adjusting for the parallel dual update. The percent of solution time during

which a pivot is in progress is shown as *Active Pivoting* in Figure 7. Note that when four or more processors are used, a pivot is in progress over 85% of the time, compared to only 35% of the time in the serial case.

The remaining portion of solution time may be attributed to two things. First, at times a candidate may not be available at the end of a pivot, perhaps because (a) the pivot was very short in duration, as when the basis is immature, (b) changes in the dual variables caused candidates to no longer be attractive, (c) the few attractive arcs have not yet been located, or (d) the solution is optimal. Therefore a new pivot may not begin immediately and all processes would engage in pricing, a scenario analogous to the serial case in which pricing time exacts a real-time cost. Second, even when an attractive candidate is available at the end of a pivot, a minimal amount of monitor overhead is incurred.

It is clear from Figure 7 that, given enough processors, a majority of the pricing effort is obtained "free," that is, the pricing is done *during* pivots, not *between* pivots. This means that solution time is driven primarily by the total pivot time and so the pivot becomes the "bottleneck" operation. Therefore, to reduce solution time, we need to reduce total pivot time. One method for achieving this is the parallel dual update.

5.6 Effect of the Parallel Dual Update

Our study of the algorithm also examined the effect of decomposing the dual update. Since roughly 60% of pivot time is spent in the dual update, a division of this work between two processors could cut the dual update time in half, thereby achieving a 30% reduction in pivot time (disregarding any overhead).

In Figure 7, *Parallel Dual Update* shows the percentage of the total dual update time in which two processors were actually performing the dual update as a function of the number of processors. This data indicates that, given a sufficient number of processors, the time spent in the dual update portion of the pivot can be reduced by nearly 40%.

5.7. Conclusions from the Statistical Study

Our analysis revealed three sources of speedup: (1) a large portion of the pricing is performed concurrently with pivot execution and therefore is "free;" (2) better basis-entry candidates reduce the total number of pivots; and (3) the parallel dual update reduces individual-pivot time. Specifically, our testing revealed that when four or more processors are employed, a pivot is in progress over 85% of the solution time—that is, most of the pricing is done simultaneously with pivoting. With little time spent exclusively on pricing, parallel solution time is determined primarily by the pivot operation. (This is in contrast to the serial case wherein pricing alone comprises over 60% of the solution time.) Hence, in parallel settings, any reduction in pivot time will reduce the overall solution time.

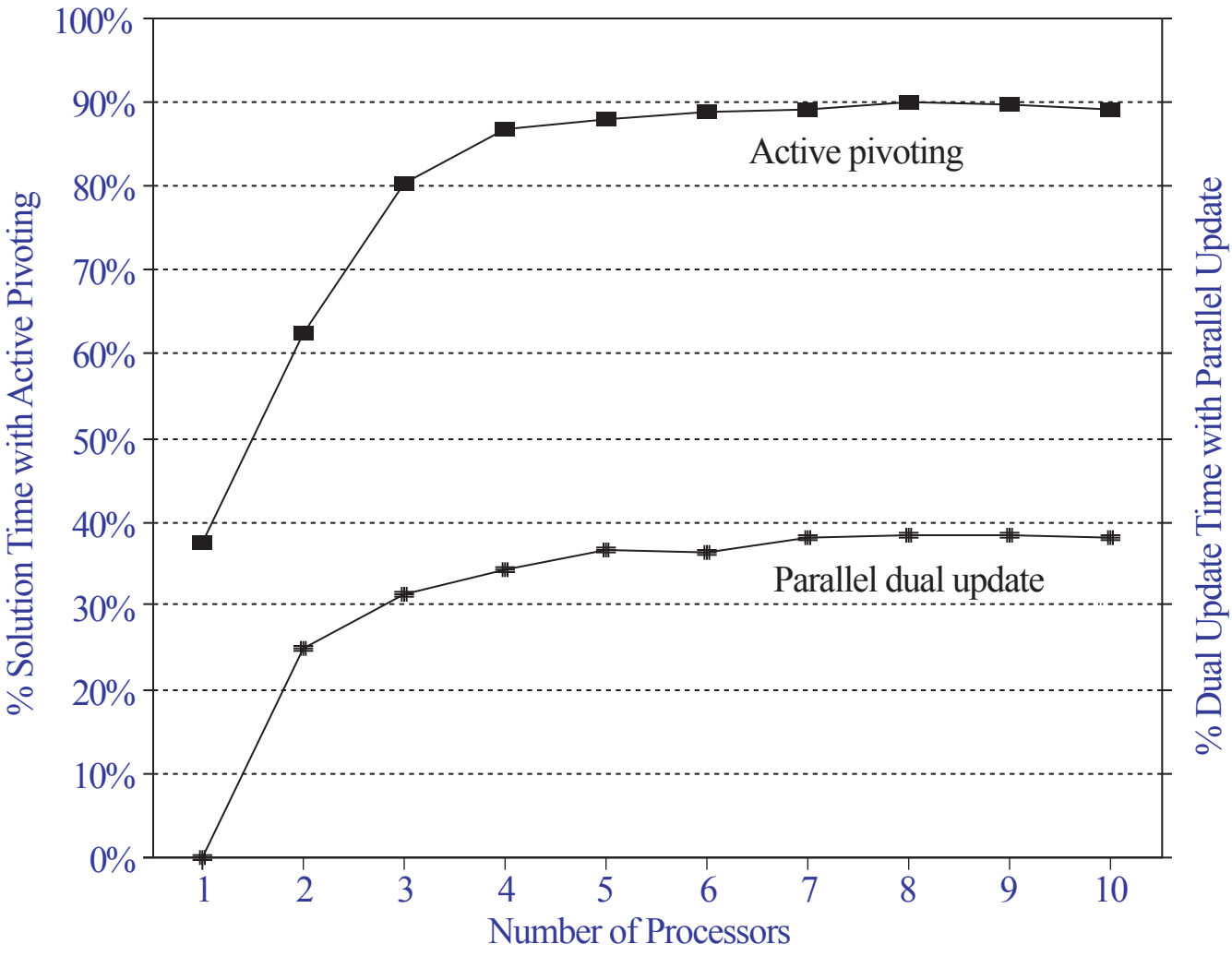


Figure 7. Pivoting effort and parallel dual update times

One way to reduce pivot time is to perform fewer pivots. By employing more processors in the pricing operation, a more exhaustive search for the entering arc can be made prior to each pivot. In this manner, progress toward optimality is accelerated and the number of required pivots—and solution time—is reduced.

Another means of reducing total pivot time is to decompose the work associated with the pivot operation itself so that it can be performed by multiple processors. In our implementation, the most time-consuming portion of the pivot, updating the dual variables, can be executed by two processors. This results in as much as a 25% reduction in pivot time.

6. IN SUMMARY

Although the speed of serial processors is increasing rapidly, the newest systems configure multiples of these computing elements for even higher-speed parallel processing. This study demonstrates the applicability of such parallelism to the solution of large-scale network problems. Results were obtained on relatively slow processors that compare favorably with massively parallel implementations by others. The resultant parallel code appears to be the state-of-the-art for the pure network problem, and we conjecture that its performance on faster MIMD systems would be even more impressive.

The occasional superlinear results of our computational testing spurred an investigation into their sources. A detailed analysis, based on microsecond-level timing of events and post-execution replay of the solution process revealed heretofore unknown temporal characteristics of network bases that were the same in both serial and parallel cases, and were accidentally exploited by the application of parallelism. Specifically, the structure of the network basis at a particular point in time is primarily influenced by the number of pivots executed up to that point, independent of the number of processors used. By utilizing a sufficient number of processors, the pricing effort not only becomes "free" but produces more attractive candidates for basis entry and fewer pivots required to reach optimality. This effort decreases the number of mature, more difficult pivots, resulting in a lower overall solution time. This unexpected finding was another example of a common byproduct of parallel empirical testing: insight into the nature and behavior of algorithms (Barr and Hickman, 1993b).

REFERENCES

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin. 1993. *Network Flows: Theory, Algorithms, Applications*, Prentice-Hall, Englewood Cliffs, New Jersey.

Amini, M. and R.S. Barr. 1990. Applications of Network Reoptimization. In *Proceedings of the 21st Annual Conference of the Southwest Decision Sciences Institute*, Decision Sciences Institute, Atlanta, 77-79.

Aronson, J. 1989. A Survey of Dynamic Network Flows. *Annals of Operations Research* 20, 1-66.

Balas, E., D. Miller, J. Pekny, and P. Toth. 1989. A Parallel Shortest Path Algorithm for the Assignment Problem. Management Science Report MSRR 552, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Barr, R.S., F. Glover, and D. Klingman. 1979. Enhancements of Spanning Tree Labeling Procedures for Network Optimization. *INFOR* 17, 16-34.

Barr, R.S. and B. L. Hickman. 1990. A New Parallel Network Simplex Algorithm and Implementation for Large Time-Critical Problems. Technical Report 89-CS-90, Department of Computer Science and Engineering, Southern Methodist University, Dallas, Texas.

Barr, R.S. and B. L. Hickman. 1993a. Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinion. *ORSA Journal on Computing* 5, 2-18.

Barr, R.S. and B. L. Hickman. 1993b. Using Parallel Empirical Testing to Advance Algorithmic Research. *ORSA Journal on Computing* 5, 29-32.

Bertsekas, D.P. 1991. *Linear Network Optimization*. MIT Press, Cambridge, Massachusetts.

Bertsekas, D. P. and D. A. Castañon. 1990. Parallel Asynchronous Primal-Dual Methods for the Minimum Cost Flow Problem. LIDS Report P-1998, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Bertsekas, D., D. Castañon, J. Eckstein, and S. Zenios. 1991. Parallel Computing in Network Optimization. Report 91-09-02, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania (forthcoming in *Handbook of Operations Research*).

Chalmet, D.G., R.L. Francis, and P.B. Saunders. 1982. Network Models for Building Evacuation. *Management Science* 28, 86-105.

Dubois, M., C. Scheurich, and F. Briggs. 1988. Synchronization, Coherence, and Event Ordering in Multiprocessors. *IEEE Computer* 21, 9-21.

Ford, L. R., Jr., and D. R. Fulkerson. 1957. A Primal-Dual Algorithm for the Capacitated Hitchcock Problem. *Naval Research Logistics Quarterly 4*, 47-54.

Flynn, M. J. 1966. Very High-Speed Computing Systems. *Proceedings of the IEEE* 54, 1901-1909.

Glover, F., D. Karney, D. Klingman, and A. Napier. 1974. A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems. *Management Science* 20, 793-813.

Glover, F., D. Klingman, and N. V. Phillips. 1989. A Modelling/Solution Approach for Optimal Deployment of a Weapons Arsenal. *Annals of Operations Research 20*, 159-177.

Glover, F., D. Klingman, and N.V. Phillips. 1992. *Network Models in Optimization and Their Applications in Practice*, John Wiley and Sons, New York.

Hoare, C. A. R. 1974. Monitors: An Operating System Structuring Concept. *Communications of the ACM* 17, 549-557.

Kennington, J. L. and R. V. Helgason. 1980. *Algorithms for Network Programming*. John Wiley and Sons, New York.

Klingman, D., A. Napier, and J. Stutz. 1974. NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems. *Management Science* 20, 814-821.

Klingman, D. and J. Mote. 1987. Computational Analysis of Large-Scale Pure Networks. Presented at the Joint National Meeting of ORSA/TIMS, New Orleans.

Li, X. and S. A. Zenios. 1991. Data-level Parallel Solution of Min-cost Network Flow Problems Using ε-Relaxations. Report 91-05-04, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania.

Lusk, E. L. and R. A. Overbeek. 1985. Use of Monitors in FORTRAN: a Tutorial on the Barrier, Self-Scheduling Do-Loop, and Askfor Monitors. In Kowalik, J.S., ed., *Parallel MIMD Computation: The HEP Supercomputer and its Applications*, The MIT Press.

Miller, D. L., J. F. Pekny, and G. L. Thompson. 1990. Solution of Large Dense Transportation Problems Using a Parallel Primal Algorithm. *Operations Research Letters* 9, 319-324.

Mulvey, J. M. 1978. Pivot Strategies for Primal-Simplex Network Codes. *Journal of the Association for Computing Machinery* 25, 266-270.

Murty, K.G. 1992. *Network Programming*. Prentice-Hall, Englewood Cliffs, New Jersey.

Nielsen, S. S. and S. A. Zenios. 1991. Proximal Minimizations with D-Functions and the Massively Parallel Solution of Linear Network Programs. Report 91-06-05, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania.

Nielsen, S. S. and S. A. Zenios. 1992. Solving Linear Stochastic Programs Using Massively Parallel Proximal Algorithms. Report 92-01-05, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania.

Osterhaug, Anita, 1992. *Guide to Parallel Programming on Sequent Computer Systems.* Prentice-Hall, Englewood Cliffs, New Jersey.

Peters, J. 1990. The Network Simplex Method on a Multiprocessor. *Networks* 20, 845-859.

Zenios, S. A. and M. Ç. Pinar. 1992. Parallel Block-Partitioning of Truncated Newton for Nonlinear Network Optimization. *SIAM Journal on Scientific and Statistical Computing* 13, 1173-1193.

ADDITIONAL TABLES OF SUPPORTING DATA

Pivot Interval	Number of Processors									
	1	2	3	4	5	6	7	8	9	10
1-1000	2	2	2	2	2	2	2	2	2	2
-2000	2	2	2	2	2	2	2	2	2	2
-3000	2	2	2	2	2	2	2	2	2	2
-4000	2	3	2	2	2	2	2	2	2	2
-5000	3	3	3	3	3	3	3	3	3	3
-6000	4	4	4	4	4	4	4	5	4	4
-7000	6	6	6	6	6	6	6	6	6	6
-8000	7	8	9	8	8	9	8	8	8	8
-9000	11	12	12	13	13	14	13	13	12	12
-10000	16	16	19	19	18	19	18	20	19	21
-11000	28	25	27	24	34	27	32	28	28	30
-12000	42	40	52	66	58	43	46	45	48	51
-13000	60	76	75	83	78	65	65	73	66	70
-14000	78	80	80	82	75	71				
-15000	74	84	83	77	72					
-16000		75	77	82	77					
-17000		72	78	76						
-18000			72	76						
-19000			73							

Table VII. Average Cardinality Over Pivot Interval

Pivot Interval	Number of Processors									
	1	2	3	4	5	6	7	8	9	10
1-1000	4633	4638	4634	4636	4630	4634	4630	4633	4629	4632
-2000	3975	3976	3972	3965	3964	3974	3961	3967	3967	3975
-3000	3477	3470	3456	3462	3446	3469	3457	3470	3473	3481
-4000	3122	3093	3093	3075	3088	3092	3091	3105	3111	3114
-5000	2838	2816	2794	2802	2808	2804	2823	2836	2838	2837
-6000	2641	2617	2595	2630	2595	2610	2612	2631	2629	2633
-7000	2512	2490	2493	2497	2468	2509	2503	2516	2509	2525
-8000	2436	2396	2381	2386	2383	2408	2424	2431	2424	2437
-9000	2350	2355	2325	2314	2331	2345	2359	2363	2366	2365
-10000	2322	2292	2298	2295	2295	2323	2348	2330	2328	2337
-11000	2321	2248	2263	2251	2275	2313	2317	2311	2322	2317
-12000	2301	2250	2240	2247	2246	2295	2293	2272	2278	2284
-13000	2294	2235	2227	2253	2242	2273	2283	2276	2286	2280
-14000	2287	2242	2220	2256	2263	2277				
-15000	2295	2250	2235	2262	2279					
-16000		2250	2243	2269	2287					
-17000		2272	2263	2279						
-18000			2279	2277						
-19000			2279							

Table VIII. Average Number of Singleton Subtrees Per Interval

Pivot Interval	Number of Processors										Overall Mean
	1	2	3	4	5	6	7	8	9	10	
1-1000	4	4	4	4	4	4	4	4	4	4	4.0
-2000	4	4	4	4	4	4	4	4	4	4	4.0
-3000	5	5	5	5	5	5	5	5	5	5	5.0
-4000	6	6	5	6	6	5	6	6	5	5	5.6
-5000	7	7	7	7	7	7	7	7	7	7	7.0
-6000	9	9	9	9	9	10	10	10	10	10	9.5
-7000	13	12	13	12	13	13	12	13	12	13	12.6
-8000	17	19	19	18	17	20	18	16	17	17	17.8
-9000	25	26	25	28	32	30	27	27	27	25	25.0
-10000	37	35	42	40	40	40	37	44	39	43	39.7
-11000	58	56	61	58	62	51	58	53	54	53	56.4
-12000	74	75	83	88	93	62	69	66	68	69	74.7
-13000	93	110	115	122	126	98	95	106	98	103	106.6
-14000	119	113	119	128	115	101					115.8
-15000	106	115	125	121	107						114.8
-16000		105	114	117	112						112.0
-17000		95	108	113							105.3
-18000			93	102							97.5
-19000			94								94.0

Table IX. Average Basis Equivalent Path Length Over Interval

Pivot Interval	Number of Processors									
	1	2	3	4	5	6	7	8	9	10
1-1000	7	5	9	4	5	7	11	9	9	13
-2000	25	26	21	31	27	18	20	23	22	19
-3000	33	33	37	38	36	40	35	36	33	29
-4000	42	41	50	42	36	46	35	41	42	46
-5000	35	41	36	43	35	38	45	35	39	37
-6000	49	48	50	43	47	55	54	47	51	52
-7000	55	66	67	61	60	58	52	59	62	53
-8000	79	67	78	72	79	69	78	67	69	54
-9000	101	98	87	115	104	120	81	89	87	78
-10000	125	103	131	134	109	114	117	120	109	102
-11000	161	170	196	152	175	176	166	172	147	138
-12000	217	220	249	229	276	180	201	183	173	179
-13000	251	294	332	297	331	307	236	260	234	248
-14000	277	338	341	313	277	234				
-15000	270	282	297	270	273					
-16000		243	293	269	268					
-17000		228	271	311						
-18000			280	285						
-19000			259							

Table X. Number of Degenerate Pivots Over Pivot Interval

Pivot Interval	Number of Processors									
	1	2	3	4	5	6	7	8	9	10
1-1000	1	1	1	1	1	1	1	1	1	1
-2000	2	2	2	1	1	1	1	1	1	2
-3000	2	2	2	2	2	2	2	2	2	2
-4000	3	3	3	3	3	3	3	3	3	3
-5000	6	5	6	5	6	6	6	6	6	7
-6000	12	9	11	11	10	11	12	13	14	12
-7000	23	18	20	16	21	21	19	20	19	22
-8000	37	52	49	39	42	50	41	38	39	45
-9000	83	85	89	106	133	108	102	108	104	90
-10000	145	181	222	185	199	206	194	254	211	257
-11000	408	330	376	363	456	320	408	366	390	400
-12000	551	605	564	747	633	503	556	536	564	556
-13000	723	815	792	819	868	711	724	803	789	823
-14000	884	846	862	839	855	810				
-15000	831	876	857	878	754					
-16000		802	835	869	821					
-17000		750	800	858						
-18000			779	823						
-19000			820							

Table XI. Average Nodes in Updated Subtree Over Interval

Pivot Interval	Number of Processors									
	1	2	3	4	5	6	7	8	9	10
1-1000	0.293	0.300	0.242	0.271	0.256	0.264	0.218	0.210	0.209	0.212
-2000	0.311	0.305	0.255	0.271	0.254	0.253	0.217	0.221	0.214	0.233
-3000	0.332	0.335	0.283	0.297	0.276	0.273	0.260	0.251	0.256	0.287
-4000	0.363	0.366	0.319	0.332	0.332	0.307	0.323	0.333	0.322	0.349
-5000	0.421	0.416	0.370	0.383	0.392	0.395	0.400	0.402	0.405	0.429
-6000	0.511	0.499	0.482	0.494	0.484	0.535	0.539	0.545	0.563	0.551
-7000	0.670	0.653	0.636	0.588	0.664	0.693	0.665	0.691	0.661	0.729
-8000	0.829	0.998	0.929	0.904	0.899	1.043	0.949	0.866	0.897	0.947
-9000	1.221	1.377	1.298	1.459	1.750	1.523	1.491	1.439	1.460	1.343
-10000	1.730	2.043	2.221	2.088	2.206	2.143	2.099	2.407	2.172	2.436
-11000	3.255	3.196	3.269	3.150	3.667	2.798	3.394	2.987	3.139	3.179
-12000	4.162	4.674	4.306	5.117	4.962	3.688	4.227	3.946	4.070	4.082
-13000	5.160	6.256	5.742	6.011	6.542	5.168	5.445	5.720	5.459	5.748
-14000	6.402	6.323	6.090	6.159	6.342	5.482				
-15000	6.008	6.513	6.263	6.205	5.736					
-16000		6.107	6.016	6.144	6.216					
-17000		5.740	5.705	5.984						
-18000			5.284	5.661						
-19000			5.384							

Table XII. Total Real Time Per Interval