ICS 89

FOURTH INTERNATIONAL CONFERENCE
ON SUPERCOMPUTING AND
THIRD WORLD SUPERCOMPUTER EXHIBITION

Santa Clara Convention Center, Santa Clara, CA USA
April 30 - May 5, 1989

PROCEEDINGS

SUPERCOMPUTING '89:

SUPERCOMPUTING STRUCTURES & COMPUTATIONS

Editors
Prof. Lana P. Kartashev
and
Dr. Steven I. Kartashev

VOLUME I

International Supercomputing Institute, Inc.

1989

Algorithms for Shortest Paths Problems on an Array Processor

J. Yadegar
S. El-Horbaty

AMT,INC.,IRVINE,CA.,USA
UAE UNIV.,AL AIN.

D. Parkinson
A.M. Frieze

C 1C,LONDON UNIV.,UK
CMU,PITTSBURGH,USA.

ABSTRACT

The innovation of parallel computers has added a new dimension to the design of algorithms. Parallel programming is not a simple extension of serial programming. We describe and implement parallel algorithms for the well known problem of finding shortest paths in a network. We present our computational experience using the massively parallel processor DAP.

Keywords: Shortest paths problems, network, grid graph, parallel algorithms, distributed array of proceessors, SIMD computer.

1. INTRODUCTION

One of the most fundamental problems in network theory is to find the shortest and/or longest path(s) in a network. These problems are major components of numerous quantitative transportation and communication models which seek to improve efficiency by reducing travel time, minimizing congestion, increasing capacity, lowering the cost of transportation service, or improving vehicle routing. Such models usually use a network to represent the transportation system where it is required to find the *minimum* time, cost, distance, or *maximum* capacity between several pairs of points in the network. The former are often called *shortest path problems* and the latter are called *longest path problems*.

The longest path problem is the central component of critical path analysis which is applied for the economical scheduling, planning and controlling of large and complex projects. It is often known by acronyms such as CPS, CPM and PERT. It is noteworthy that the longest path problem is mathematically identical to a shortest path problem and thus, the algorithms described in this paper apply to all such problems.

There has, therefore, been a large number of papers and reports published on this subject. To mention but a few, the survey by Dreyfus [10], the bibliographies by Golden and Magnanti [19] and Pierce [27], and the taxonomy and annotation by Deo and Pang [6]. Many papers have also been written to examine the relative merits of various shortest path algorithms, such as Dial et al. [8], Gilsinn and Witzgall [16], Glover and Klingman [17], Golden [18] and Hulme and Wisniewski [20].
The innovation of parallel computers has added a new dimension to the design of algorithms. *Parallel programming* is not a simple extension of serial programming.

Our motivation in this research has been to develop parallel algorithms for shortest path problems; namely, the *single-source problem* and *all-pairs problem*.

Although most networks arising in practical problems (e.g. road and project networks) have only non-negative arc lengths, there are, however, networks which have both negative and positive arc lengths (e.g. a negative arc length may represent cost and a positive arc length may show income). Our algorithm deals with such networks (the most general case), and a specialized version of it is used for grid networks, modeling "perfect" cities and VLSI systems. We have also described a parallel algorithm for the all-pairs problem.

This research differs from most other works in this area - see, for example, Chandy and Misra [5], Deo, Pang and Lord [7], Frieze [15], Mateti and Deo [23] and Resta [29] - in that all the algorithms described in this paper have been implemented and executed on a real SIMD computer, called the Distributed Array of Processors (DAP). With this in mind, the emphasis of the paper is on implementation rather than the well known complexity results on the shortest path problem - see, for example, Lawler [22].

The rest of the paper is organized as follows. In Section 2, we outline very briefly the DAP. Section 3 contains the relevant definitions and notations. In Section 4, we describe a parallel algorithm for finding shortest paths for the single-source problem and present our computational results in Section 4.2. The parallel algorithm to deal with the rectangular grid network is described in Section 5, followed by some results in Section 5.2. In Section 6, we present a parallel algorithm for the all-pairs problem and give computational results in Section 6.2. We conclude the paper in Section 7.

2. THE DAP ARCHITECTURE

The general principle of the DAP - see Parkinson [25, 26] and Quinn [28] - is that of a SIMD machine as defined by Flynn [12]. On a p x p DAP, one can perform up to p^2 operations (of the same type) simultaneously. This parallel processing capability of the DAP is achieved by a p x p matrix of processors, called Processing Elements, each of which may operate independently on its own local store. Thus, it is convenient to think of the DAP as a square array of processors placed on a 2-dimensional grid in which each processor can communicate directly with its four neighbors. Specifically, we have an array of p^2 (p=32 or 64) processors P_{ij}, $1 \leq i, j \leq p$, where P_{ij} can communicate directly with its four neighbors $P_{i-1,j}$, $P_{i+1,j}$, $P_{i,j-1}$, and $P_{i,j+1}$, with $1 - 1 \equiv p$ and $p + 1 \equiv 1$. In addition, processors are connected via row and column highways to a set of edge registers in such a way that in unit time data can be selected from any set of processors, one per row (or column), into the corresponding register; or data can be broadcast from a register to some or all processors in the same row (or column). The row and column highways coupled with the bit serial nature of the processors, allows the DAP to exhibit many properties of associative or content addressable processors [14]. For further details on the DAP and its programming language, Fortran-Plus, we refer the reader to [1 , 2].

We measure the complexity of the algorithms in terms of number of "DAP operations", each one is executable very "efficiently" on the DAP. Some of these are:

(i) Compute the resultant matrix (a_{ij} o b_{ij}) where o=+, - ,* or/. The p x p matrices (a_{ij}) and (b_{ij}) are stored one element per processor.

(ii) Overwrite a row or column of a p x p matrix with a given p-vector.

(iii) Given a p-vector V, construct a p x p matrix MATR (V), where each row is identical to V. Similarly, MATC (V) has each column identical to V.

(iv) Compute the position(s) of the minimum value of a p x p matrix or a p-vector.

On the DAP, one is able to execute these operations in O(1) time. To get a feeling of the actual computation times for these operations on the existing DAPs with p=32 and p=64, Table 2-1 gives the ratio of the time for an operation compared to that for a 32 bit floating point matrix (element by element) multiplication as defined in (i) above. The time for matrix (element by element) multiplication is 155.7 micro seconds, in which time 1024 multiplications are performed when p=32 and 4096 multiplications are performed when p=64.

TABLE 2.1: Ratio of the DAP computation time for an operation compared to that of a 32 bit floating point matrix (element by element) multiplication.

Function	Operation Time / Multiply Time
*	1.0
+	0.67
/	1.36
A(i,)=V or A(,j)=V	0.13
MATR(V) or MATC(V)	0.11
MINP(A)	0.14
MINP(V)	0.14

In Table 2.1, A is a p x p matrix and V is a p-vector. The assignment $A(i,)=V$ $(A(,j)=V)$ overwrites the i^{th} row (j^{th} column) of A by V. Functions MATR and MATC have already been defined in (iii) above. Function MINP takes a matrix (or vector) argument and returns a logical matrix (or vector) with .TRUE. value(s) corresponding to the position(s) of the minimum value of its argument. For more detail on these functions and other Fortran-Plus functions we refer the reader to [3].

3. DEFINITIONS AND NOTATIONS

Before describing our algorithms, it is appropriate to summarize some of conventions and terms we shall use in this paper.

We shall deal with a *directed graph* or digraph $G=(V,A)$ consisting of a finite set $V = \{1, 2, ..., n\}$ of elements called *nodes* or *vertices* and a finite set $A \subseteq V \times V$ of $m(\leq n)$ ordered pairs of nodes called *arcs* or *edges*. An edge (u,v) of A is said to be directed from u to v, where v is called a *successor* of u, and u is called a *predecessor* of v.

A *directed path* or *path* from node s to node t (called an *s-t path*) is a finite sequence of arcs $(s, v_1), (v_1, v_2), ..., (v_{k-1}, t)$, sometimes written as $(s, v_1, ..., t)$, in which all the vertices are distinct. Note that such a path has no *loops*. An s-t path is a *cycle* if s=t.

By a *network*, we mean a digraph $G=(V,A)$ together with a real valued function $l: A \to R$ defined as follows:

$$
\begin{array}{lll}
l_{ij} & = \text{ the (finite) length of arc } (i,j) & \text{if } (i,j) \in A \\
 & = \infty & \text{otherwise} \\
l_{ii} & = 0 & \text{for } i \in V
\end{array}
$$

As mentioned earlier, in a (general) network, G, the arc lengths may be either positive or negative. However, we assume that G contains no directed cycle with a strictly negative length.

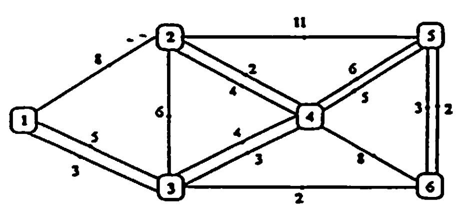
It is noteworthy that an undirected shortest path problem can be reduced to a directed one, if all arc lengths are non-negative, by replacing each undirected arc $\{i,j\}$ by a symmetric pair of directed arcs (i,j) and (j,i) with $l_{ij} = l_{ji}$. However, if the length of arc $\{i,j\}$ is negative, then such a transformation would introduce a negative directed cycle into the network, and this approach is invalid. In such cases, the problem can be reduced to that of a weighted non-bipartite matching problem [21].

3.1. Computer Representation of a Network

A network may be represented in a computer in several ways and the manner in which it is represented directly affects the performance of algorithms applied to the network. There is no single overall best data structure for networks. The choice depends on the size of the network, its density and on the computer as well as the language being used.

The *length-matrix* in which the function l is represented as an n x n matrix, and the *arc-lists* in which for each node i, there is a list of those arcs (i,j) which are incident to node i, together with their lengths l_{ij}, are perhaps the most used data structures for networks - see Syslo, Deo and Kowalik [30].

Below in Figure 3.1, we give an example of a network with its length-matrix as well as arc-lists representations.



$$
\begin{bmatrix}
0 & 8 & 5 & \infty & \infty & \infty \\
\infty & 0 & 6 & 4 & 11 & \infty \\
3 & \infty & 0 & 4 & \infty & 2 \\
\infty & 2 & 3 & 0 & 6 & 8 \\
\infty & \infty & \infty & 5 & 0 & 3 \\
\infty & \infty & \infty & \infty & 2 & 0
\end{bmatrix}
$$

Tails = (1 1 2 2 2 3 3 3 4 4 4 4 5 5 6)

Heads = (2 3 3 4 5 1 4 6 2 3 5 6 4 6 5)

Arc lengths = (8 5 6 4 11 3 4 2 2 3 6 8 5 3 2)

Figure 3.1: A network with its length-matrix and arc-lists representations.

Notes:
1) The diagonal entries in the matrix representation are usually set to 0, or to some other value depending on the application and algorithm.
2) In the arc-lists representation, we have listed arcs in ascending order of their initial nodes (tails). However, they could equally have been listed in any other order, say in ascending order of their arc-lengths or simply they may be listed randomly.
3) If the network is sparse, i.e., $m << n(n-1)$, then it would be wasting memory space to store the ∞'s in the matrix representation. In such a case, it is often more efficient to use the arc-lists approach, which requires 3m memory space.

On a p x p DAP (p=32 or 64), the length-matrix and the arc-lists representations are evidently suitable for dealing with problems with p nodes and p^2 arcs respectively. A number of applications require to solve much larger problems. In this case the data sets should be mapped into the DAP store in such a way restricted by the number of processors of the DAP. There are many different ways for mapping the data sets into the DAP store - see Parkinson [24] - and the optimum mapping strategy may depend on the operations required to perform. This is discussed fully in Section 4.1.

4. SINGLE-SOURCE SHORTEST PATH PROBLEM

The most commonly encountered shortest path problem is to find the shortest paths (or the lengths of shortest paths) from a specified node, called the source, to all other nodes in a network. This problem is usually known as the "single-source problem". In this section we describe a parallel algorithm for (the most general case) of this problem. That is, when the arc lengths can be positive or negative.

4.1. Algorithm SP

This algorithm finds the length of shortest paths from the source to every other node in a directed network in which the arc lengths can be positive or negative. Of couse, as mentioned earlier, we do, however, assume that there are no negative cycles.

Algorithm SP solves the problem by successive approximations. That is, given an approximation of a shortest path from the source to node j, we try to improve this approximation by considering paths via predecessor nodes of j. Formally, let:

$d_j^{(r)}$ = the length of a shortest path from the source (say node 1) to node j, such that the path contains no more than r arcs (4.1)

There are two cases to be considered. One for problems smaller than or equal DAP size, and one for problems larger than DAP size.

<u>CASE 1</u>: $n^2 \le$ number of processors p^2

Each processing element, PE_{kj}, keeps the value of l_{kj} and also the value of $d_k^{(r)}$. Let $\underline{d}^{(r)}$ be the vector holding the values $(d_1^{(r)}\ d_2^{(r)} \dots d_j^{(r)} \dots d_n^{(r)})$. Initially, we let:

$$\underline{d}^{(1)} = (l_{11}\ l_{12} \dots l_{1i} \dots l_{1n})$$

Then, we compute $(r+1)^{st}$ order approximations from the r^{th} order for all $j \in V$ simultaneously as follows:

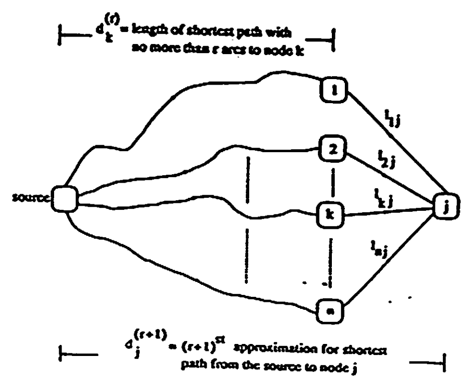
$$\underline{d}^{(r+1)} = \underline{d}^{(r)} \text{ O } L$$

Where $L = (l_{ij})$ is the arc length matrix, and O is defined by:

$$d_j^{(r+1)} = \text{Min}_j \{d_k^{(r)} + l_{kj}\} \quad \text{for all } j \in V \qquad (4.2)$$

Here, we are assuming that $l_{jj}=0$ for all $j \in V$.

Pictorially, we have:



Now, the ability of DAP to broadcast using the row and/or column high ways means that in O(1) time operation, all the processors PE_{kj} will have simultaneously the corresponding value of $d_k^{(r)} + l_{kj}$, for all k, j. It only remains to find the minimum across each column, and this can be done on the DAP simultaneously very efficiently.

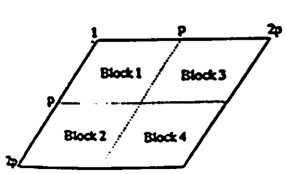
It is perhaps interesting to observe a certain similarity between (4.2) and algebraic multiplication of a row vector $\underline{d}^{(r)}$ by a matrix L; that is, $d_j^{(r+1)} = \Sigma_{k=1}\, d_k^{(r)} * l_{kj}$ for all j, when replacing the summation and multiplication by taking minima and addition in (4.2) respectively.

It is not difficult to see, given the definition of $d_j^{(r)}$ in (4.1), that equation (4.2) does converge to the correct value of the shortest path from the source to node j. Indeed, we can be assured that $d_j^{(n-1)} = d_j$ for all $j \in V$, where d_j is the length of a shortest path from the source to node j.

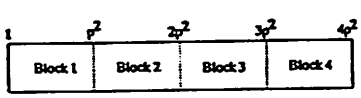
It is important to notice that the computation may be terminated whenever $\underline{d}^{(r+1)} = \underline{d}^{(r)}$; a test which can be executed on the DAP extremely fast. This stopping criterion is usually reached long before r=n, and this is what makes the method so efficient on average.

CASE 2: n^2 > number of processors p^2

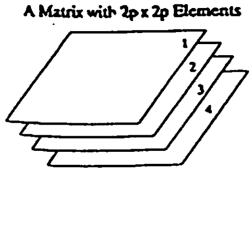
In this case we need to find a way to fit the problem into the DAP store. One of the most common mappings is called *slicing*. To demonstrate this, let us consider, for example, the task of dealing with a network having n=2p nodes(and a maximum of $4p^2$ arcs), where p^2 is the number of processors on the DAP. The strategy is to simply cut the length matrix $L=(l_{ij})$ into four blocks of DAP matrices, and store them in a matrix array as follows:



A Matrix with 2p x 2p Elements



A Vector with $4p^2$ Elements



A DAP Matrix Array having 4 DAP frames

The DAP declaration for L is a real (or integer) matrix array ARC-LENGTH (, ,NB, NB), where NB=[n/p]* (for our assumption, NB=2). The first comma in the declaration, defines a frame of the DAP data holding p^2 values which can be processed simultaneously.

Note: The slicing mapping can be considered similar to the following Fortran declaration:

REAL L(64, 64, 2, 2)

or

REAL L(64, 64, 4)

when p=64 say.

Let us now define $\underline{d}_1^{(r)}$ and $\underline{d}_2^{r}$ to be the corresponding r^{th} approximation of the lengths of shortest paths from the source to the set of nodes in $V_1=\{1, 2, ...,p\}$ and $V_2 = \{p+1, p+2, ...,2p\}$ respectively. Then:

$$(\underline{d}_1\ \underline{d}_2)^{(r+1)} = (\underline{d}_1\ \underline{d}_2)^{(r)} \boxtimes \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{bmatrix}$$

where L_{ij} is the frame of DAP data stored in ARC-LENGTH (,, i, j), and $\boxtimes$ is defined by:

$$\begin{aligned} \underline{d}_1^{(r+1)} &= \min\{\underline{d}_1^{(r)} \otimes L_{11}, \underline{d}_2^{(r)} \otimes L_{21}\} \\ \underline{d}_2^{(r+1)} &= \min\{\underline{d}_1^{(r+1)} \otimes L_{12}, \underline{d}_2^{(r)} \otimes L_{22}\} \end{aligned} \tag{4.3}$$

*[x] is the ceiling function; that is, [x] = x if x is integer
= int(x+1) otherwise

Here, we notice that once we have computed $\underline{d}_1^{(r+1)}$, this was subsequently used in calculating $\underline{d}_2^{(r+1)}$. This is because, to calculate $\underline{d}_1^{(r+1)}$, one needs only the blocks L_{11} and L_{21} which represent the length of the arcs in $V_1 \times V_1$ and $V_2 \times V_1$ respectively. Similarly, to compute $\underline{d}_2^{(r+1)}$ we only need the data in blocks L_{22} and L_{12}.

Computationally, (4.3) was implemented more efficiently in the sense that the minimum of the two operands, which are DAP matrices, was carried out first (and this is very efficient on the DAP) and then the minimum across each column was computed. Also, there are two ways in storing the length of the paths on the DAP. One, as a vector array, say PATH-LENGTH (,NB) or as a matrix, say PATH-LENGTH (,). Storing it as a matrix means that we can treat all its components simultaneously and this becomes more efficient when NB is large.

The idea behind Algorithm SP is very close to the dynamic programming type algorithm attributed to Bellman and Ford [4, 13], having complexity of O(n) on an n x n DAP as compared to the $O(n^3)$ of the serial algorithm.

We implemented Algorithm SP on the DAP 610 with 4096 processors, using the length-matrix representation to store the network. For dense and semi-dense networks, the algorithm is very efficient (see Table 4.1). However, for very sparse graphs, the results are not as encourging as for dense and semi-dense networks. Note that the improvements suggested by Yen - see [32] - are not applicable in a parallel environment.

4.2. Computational Results

All of the test problems were generated in a random fashion. A random (connected) network with n nodes and m arcs were generated as follows:

First, a spanning tree rooted at the source node (node 1) is created by randomly joining a node in the current (non-spanning) tree to a node not in the tree yet. Then, the remaining arcs were added by joining two distinct nodes (not yet connected) chosen randomly from {1, 2 ..., n}, until a total of m arcs have been generated. We note that, the resulting network has no multiple arcs and loops.

The above procedure was used to generate random networks having from 20 to 640 nodes, with arc densities, defined as $\delta = m/(n^2 - n)$, ranging from 1.0 to 0.0025. The arc-lengths were randomly generated values from a uniform distribution on the interval [1,10000]. Table 4.1 contains the computation times of Algorithm SP. These timing become more attractive for more dense networks, confirming that the run time of Algorithm SP is proportional to the maximum number of arcs in the shortest paths in the network.

The results given in Table 4.1 compare very favorably with those of serial algorithms as reported in the literature (See for example, Hulme and Wisniewski [20] and Dial et al.
[8]).

It is worth mentioning that when we implemented a parallel version of Dijkstra algorithm
[9], using both the length-matrix and arc-lists representations, no substantial speed up was obtained. This was due to the fact that such an algorithm will process every node sequentially and evaluate the successor nodes in parallel.

TABLE 4.1: Computation Times for Algorithm SP in Millisecs on the AMT DAP 610.

N \ δ	1.0	0.5	0.25	0.1	0.05	0.025	0.01	0.0075	0.005	0.0025
20	0.305 380	0.30 190	0.394 95	0.301 38	0.394 19					
32	0.301 992	0.394 496	0.394 248	0.394 99	0.490 49					
64	0.423 4032	0.423 2016	0.490 1008	0.523 403	0.622 201	0.722 100				
100	1.074 9900	1.074 4950	1.074 2475	1.721 990	1.721 495	2.368 247	2.368 99			
128	1.074 16256	1.074 8128	1.074 4064	1.721 1625	2.044 812	2.368 406	2.692 162			
300	4.594 89700	4.594 44850	6.004 22425	8.250 8970	10.428 4485	10.428 2242	10.262 897	10.262 672	10.575 448	
320	4.594 102080	4.594 51040	6.004 25520	8.250 10208	10.364 5104	10.514 2552	10.365 1020	10.429 765	10.429 510	
500	10.404 249500	10.404 124750	13.722 62375	20.398 24950	22.720 12475	25.211 6237	25.211 2495	27.161 1871	27.161 1247	30.393 623
600	15.541 359400	15.538 179700	20.555 89850	30.585 35940	30.603 17970	38.032 8985	46.176 3594	46.176 2695	46.491 1796	46.176 898
640	17.349 408960	17.349 204480	20.540 102240	22.12 40896	30.606 20448	42.491 10224	46.176 4089	46.176 3067	46.491 2044	48.893 1022

Upper rows are the times, under that are number of acrs in graph.

Upper rows are the times, under that are number of acrs in graph.

5. GRID NETWORKS

A grid network is one which can be embedded in a rectangular plane grid. Vertices are situated at lattice points and each vertex is directly connected to at most four neighbors in the N, S, E, W direction. Therefore, a pxq grid network has pq nodes and 4pq - 2p - 2q arcs. It is important to note, however, that the arc lengths are randomly generated. Thus, arc lengths are not necessarily symmetric and the triangle inequality may not hold. Interest in these networks stems from models of "perfect" cities and VLSI. Because the structure of grid networks closely resembles that of the DAP, by using the nearest neighbors connections, we are able to process these type of networks very rapidly.

5.1. Algorithm G

In this section we describe a parallel algorithm for solving the single-source problem in rectangular grid networks. There is no restriction on the arc lengths, they can be positive or negative. However, there are no negative cycles.

The algorithm is based on successive approximation as described in Section 4.1 for Algorithm SP. The $(r+1)^{st}$ order approximation of shortest distance from the source, say node (1,1), to any node (i,j) is only affected by the r^{th} order approximation of shortest distance to its four neighbors, plus the arc lengths directed from these neighbor (N, S, E, W) nodes into node (i,j), as shown in Figure 5.1.

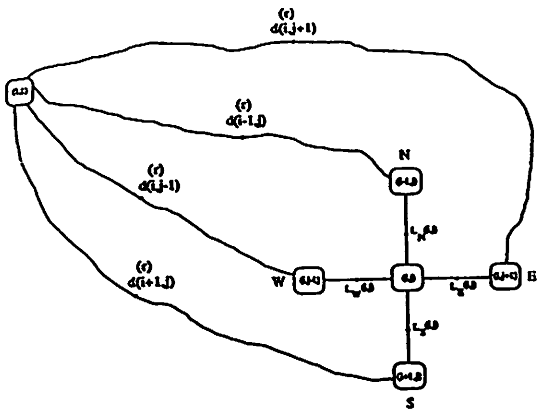


Figure 5.1

To see how Algorithm G works more clearly, let us, without loss of generality, assume we have a pxq grid network and a pxq processing element DAP (e.g. in DAP 610, p=q=64). Each processing element PE_{ij} handles the following information:

(i) The current (r^{th} order) shortest distance from node (1,1) to node (i,j) – $d^{(r)}(i,j)$.

(ii) The length of the incoming-arc from North to node (i,j) – $l_N(i,j)$.

(iii) The length of the incoming-arc from West to node (i,j) – $l_W(i,j)$.

(iv) The length of the incoming-arc from South to node (i,j) – $l_S(i,j)$.

(v) The length of the incoming-arc from East to node (i,j) – $l_E(i,j)$.

Now, if we only consider arcs coming from, say, north; that is, arcs of the form ((i-1,j), (i,j)) for all i,j, then we can compute the $(r+1)^{st}$ order approximation of shortest distances from node (1,1) to <u>all</u> nodes in <u>parallel</u> as follows:

$$d^{(r+1)}(i,j) = \min\{d^{(r)}(i,j),\ d^{(r)}(i-1,j) + l_N(i,j)\} \quad \text{for all } i,j \tag{5.1}$$

In a similar fashion we consider in turn each of $l_W(i,j)$, $l_S(i,j)$ and $l_E(i,j)$. The advantage of this algorithm is that at the second, third and fourth turn of computing (5.1) with the appropriate arc lengths, we are (always) using the improved r^{th} order approximation of the shortest distances. The algorithm terminates whenever $\underline{d}^{(r+1)} = \underline{d}^{(r)}$, a very fast check (one bit operation) on the DAP.

We like to point out that to do (5.1) on the DAP is extremely simple and fast, using the built-in shift functions which employ the nearest neighborhood connections. In fact, the piece of the Fortran-Plus to compute (5.1) is as below:

```
WM=SHSP(D) + LN
D(WM. LT. D) = WM
```

Where D and LN are DAP matrix objects, holding the pq values of $d^{(r)}(i,j)$ and $l_N(i,j)$, respectively, for i=1,...,p and j=1,...,q. WM is a temporary DAP matrix. SHSP is one of the built-in shift functions, which shifts every element in D south by one place using Planar geometry. This means values are dropped out from the south edge and zeros are feed in from the north edge. The expression (WM. LT. D) produces a logical mask, causing inhibit write.

5.2. Computational Results

It is clear that in a pxq grid network, node (p,q) is reachable from the source node (node (1,1)) after performing two shifts, say in the direction of south and east, at least (k-1) times, where k=max(p, q). So, it will be a waste of time to check for termination in Algorithm G prior to the $(k-1)^{th}$ iteration. Also, some of the four shifts tend to have no effect in earlier iterations.

Our experiments show that, it is better first to perform the two shifts for about 1/2 k' iterations (k' = min(p, q)) and then do the four shift for a further of (k - 1/2k') iterations without any check for termination. After that, at each iteration the four shifts are performed with a check for termination.

Table 5.1 describes some of the grid network test problems with number of nodes ranging from 1024 to 4096. These problems are all randomly generated using a uniform distribution with arc lengths belonging to the interval [1,10000].

The results show that the parallel Algorithm G implemented on DAP 610 is much superior to the best serial code - see Dial et al. [8] - and on average is about 25 times faster than the best serial code ran on CDC 6600 machines.

TABLE 5.1: Computation Times for Algorithm G in Millisecs on the AMT DAP 610.

Rectangularity	Number of Nodes	Numbers of Arcs	Run Times in Millesec
32 x 32	1024	3968	6.902
32 x 64	2048	8000	11.824
64 x 32	2048	8000	11.905
40 x 40	1600	6240	8.496
40 x 64	2560	10032	11.879
64 x 40	2560	10032	12.013
50 x 50	2500	9800	10.83
50 x 64	3200	12572	12.343
64 x 50	3200	12572	13.203
64 x 64	4096	16128	13.874

6. ALL-PAIRS SHORTEST PATH PROBLEM

In this section we consider the problem of finding the length of shortest path between every pair of nodes in a directed network. This problem is often called "all-pairs problem". Instead of solving this problem by repeated application of, say, Algorithm SP for solving single-source problem, choosing n separate origins, we will develop a single integrated parallel procedure.

6.1. Algorithm AP

No restrictions are placed on the arc lengths except that there are no negative cycles. Let us suppose we have an n node network, stored in an n x n matrix L using the length-matrix representation, and an n x n processor DAP. Each processing element PE_{ij} keeps the value of l_{ij} (for all i,j) and also the current r^{th} length of the shortest path from node i to node j, $d_{ij}^{(r)}$.

The algorithm works by inserting one or more nodes into paths whenever it is advantageous to do so. After initializing $D^{(1)} = L$, we construct a sequence of n matrices $D^{(2)}, D^{(3)}, ..., D^{(n+1)}$, where the $(i,j)^{th}$ entry of $D^{(r)}$ gives the length of a shortest path from node i to node j, subject to the condition that the intermediate nodes of the path are taken from the set of nodes $\{1,2,...,r-1\}$.

More formally, we need to compute the following:

$$d_{ij}^{(1)} = l_{ij} \qquad \text{for all } i,j \in V$$

$$d_{ij}^{(r+1)} = \min \{d_{ij}^{(r)}, d_{ir}^{(r)} + d_{rj}^{(r)}\} \qquad \text{for all } i,j \in V \text{ and for } r=1,2,...,n$$

Note that all the entries of $D^{(r)}$ are computed simultaneously on the DAP.

Thus, at iteration (r+1), the algorithm tries to improve paths by first checking if $d_{ir}^{(r)} + d_{rj}^{(r)} < d^{(r)}_{ij}$, and if so then the known path from node i to node j is replaced by the known path from node i to node r followed by the known path from node r to node j, see Figure 6.1.

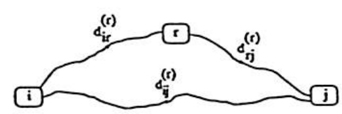


Figure 6.1

Clearly, we must have $d_{ij}^{(n+1)}$ as the length of the shortest path from node i to node j. These are the entries of $D^{(n+1)}$ which are computed simultaneously.

Algorithm AP runs in O(n) time as compared to $O(n^3)$ of the serial algorithm of Floyd and Washall - see [11] and [31] - which is the basis of our algorithm.

It is worth giving the main piece of the Fortran Plus code which will execute Algorithm AP:

```
        D = L
        DO 10 R=1,N
        WM = MATC (D(,R)) + MATR
(D(R,))
        D(WM . LT. D) = WM
10      CONTINUE
```

For problems larger than DAP size, one can use the sliced mapping and the expansion is straightforward.

6.2. Computational Results

The results of running our implementation on DAP 610 for a number of test problems are summarized in Table 6.1. The test problems are randomly generated using a uniform distribution with arc lengths belonging to the interval [1,10000]. Note that the computation time is independent of sparsity of the network.

TABLE 6.1: Computation Times for Algorithm AP in Millisecs on the AMT DAP 610.

Number of Nodes	Run Time in Millisec
20	1.952
32	3.118
40	3.897
50	4.871
60	5.845
64	6.233

7. CONCLUSION

We have demonstrated that the DAP is a powerful machine for solving shortest path problems. The performance is at its peak for more dense networks, as all the processors are active. For extremely sparse networks, which usually have some type of structure, one needs to use that structure in order to achieve worth while speed up.

Contrary to the serial case, the Bellman and Ford type parallel algorithm is faster by an order of magnitude than the Dijkstra's algorithm; a phenomenon which usually is common with other algorithms.

The current work and other similar works on the DAP indicate that the idea of applying parallel programminig techniches to solve combinatorial optimization problems is promising and deserves much more attention, and will likely lead to significant performance improvements.

REFERENCES

[1] AMT, "AMT DAP Series, Technical Overview", Active Memory Technology, Ltd., Reading, U.K. (1988).

[2] AMT, "DAP 500/600: Introduction to FORTRAN-PLUS Programming", Active Memory Technology Ltd., Reading, U.K. (1988).

[3] AMT, "DAP 500/600 FORTRAN-PLUS Language", Active Memory Technology Ltd, Reading, U.K. (1988).

[4] R. Bellman, " On a Routing Problem", Quarterly of Applied Mathematics, Vol. 16, pp. 87-90 (1958).

[5] K.M. Chandy and J. Misra, "Distributed Computation on Graphs: Shortest Path Algorithms", Communication of the ACM, Vol. 25, pp. 833-837 (1982).

[6] N. Deo and C.Y. Pang, "Shortest Path Algorithms: Taxonomy and Annotation", Technical Report No. CS-80-057, Computer Science Department, Washington State University, Pullman, Washington, U.S.A. (March 1980).

[7] N. Deo, C.Y. Pang and R.E. Lord, "Two Parallel Algorithms for Shortest Path Problems", Proceeding of International Conference on Parallel Processing", pp. 244-253 (1980).

[8] R. Dial, F. Glover, D. Karney and D. Klingman, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees", Networks, Vol. 9, pp. 215-248 (1979).

[9] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, Vol. 1, pp. 269-271 (1959).

[10] S.E. Dreyfus, "An Appraisal of some Shortest Path Algorithms", Operations Research, Vol. 17, pp. 395-412 (1969).

[11] R.W. Floyd, "Algorithm 97, Shortest Path", Communication of A.C.M., Vol. 9, p. 345 (1962).

[12] M.J. Flynn, "Very High-Speed Computing Systems", Proceedings of the IEEE, Vol. 54, pp. 1901-1909 (1966).

[13] L.R. Ford, "Network Flow Theory", pp. 923, The Rand Corporation, Santa Monica (1956).

[14] C.C. Foster, "Content Addressable Parallel Processors", Van Nostrand Reinhold Company, New York (1976).

[15] A.M. Frieze, "A Parallel Algorithm for all Pairs Shortest Paths in a Random Graph", Proceedings of the 22nd Allerton Conference on Communications, Control and Computing, pp. 663-670 (1985).

[16] J. Gilsinn and C. Witzgall, "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees", NBS TN-772, National Bureau of Standards, Washington, D.C. (May 1973).

[17] F. Glover and D. Klingman, "A Computer Study of Efficient Algorithms for Shortest Path Trees and Assignment Problems", Research Report CCS 430, Centre for Cybernetic Studies. The University of Texas, Austin, Texas 78712 (July 1982).

[18] B.L. Golden, "Shortest-Path Algorithms: A Comparison", Operations Research, Vol. 24, pp. 1164-1168 (1976).

[19] B.L. Golden and T.L. Magnanti, "Deterministic Networks Optimization-A Bibliography", Networks, Vol. 7, pp. 149-183 (1977).

[20] B.L. Hulme and J.A. Wisneiwski, "A Comparison of Shortest Path Algorithms Applied to Sparse Graphs," Report NUREG/CR-0293, SAND 78-1411, Sandia Laboratories, Albuquerque, New Mexico 87185 (August 1978).

[21] E.L. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, N.Y. (1976).

[22] E.L. Lawler, "Shortest Path and Network Flow Algorithms", Annals of Discrete Mathematics, Vol. 4 pp. 251-263 (1979).

[23] P. Mateti and N. Deo, "Parallel Algorithms for the Single Source Shortest Path Problem", Computing, Vol. 29, pp. 31-49 (1982).

[24] D. Parkinson, "Multiplication of Matrices Size NXN on P^2 Processors", Technical Report, DAP Support Unit, Queen Mary College, University of London (1981).

[25] D. Parkinson, "Practical Parallel Computers and Their Uses", in Parallel Processing Systems, Edited by D.J. Evans, Cambridge University Press, pp. 215-235 (1982).

[26] D. Parkinson, "The Distributed Array Processor (DAP)", Computer Physics Communications, Vol. 28, pp. 325-336 (1983).

[27] A.R. Pierce, "Bibliography on Algorithms for Shortest Path, Shortest Spanning Tree, and Related Circuit Routing Problems(1956-1974)", Networks, Vol. 5, pp. 129-150 (1975).

[28] M.J. Quinn, "Designing Efficient Algorithms for Parallel Computers", McGraw-Hill Book Company, New York (1987).

[29] G. Resta, "Parallel Evaluation of the Shortest Path", Technical Report No. 148, The Hatfield Polytechnic, Numerical Optimisation Centre, P.O. Box 109, College Lane, Hatfield, U.K. (1984).

[30] M.M. Syslo, N. Deo and J.S. Kowalik, "Discrete Optimization Algorithms with Pascal Programs", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 07632, U.S.A. (1983).

[31] S. Warshall, "A Theorem on Boolean Matrices", Journal of A.C.M., Vol. 9, pp. 11-12 (1962).

[32] J.Y. Yen, "An Algorithm for Finding Shortest Routes from all Source Nodes to a Given Destination in General Network", Quarterly of Applied Mathematics, Vol. 27, pp. 526-530 (1970).