An Efficient Parallel Algorithm for the All Pairs Shortest Path Problem using Processor Arrays with Reconfigurable Bus Systems

Technical Report B-13-99
July 30, 1999

Rajeev Wankar[1]
Elfriede Fehr
Freie Universität Berlin
Institut für Informatik
wankar@inf.fu-berlin.de
fehr@inf.fu-berlin.de
and

N.S.Chaudhari*
* School of Computer Science, DAVV Indore, India

Abstract

The all pairs shortest path problem is a class of the algebraic path problem. Many parallel algorithms for the solution of this problem appear in the literature. One of the efficient parallel algorithms on W-RAM model is given by Kucera[17]. Though efficient, algorithms written for the W-RAM model of parallel computation are too idealistic to be implemented on the current hardware. In this report we present an efficient parallel algorithm for the solution of this problem using a relatively new model of parallel computing, Processor Arrays with Reconfigurable Bus Systems. The parallel time complexity of this algorithm is O($\log_2$ n) and processors complexity is $n^2 \times n \times n$.

Keywords: Splitting, SPP, PARBS.

[1] This work is supported by the German Academic Exchange Services (DAAD) under the “Sandwich Model” fellowship with the author.

Introduction:

The all pairs shortest path problem is a class of the algebraic path problem. One of the sequential methods for solving this problem is based on dynamic programming [13]. In the past, a lot of research has been carried out to solve this and related problems[4,9,10]. One of the methods for the solution of the all pairs SPP has been given by Kucera[17]. The algorithm presented by Kucera takes O($\log^2$ n) time on P-RAM model and O($\log_2$ n) time on the W-RAM model of computations. Another method to solve the same problem has been given by G.H. Chen et al.[9]. Their method is based on the Processor Arrays with Reconfigurable Bus System (PARBS) model of parallel computations. They used iterative procedure, requiring $log_2 n$ iterations and each consisting of one matrix addition and two multiplications. The matrix multiplication can be done in constant time if the "+" operator in the innermost loop is MAX(MIN). This method solves the problem in O(log n) time using $n^2 \times n \times n$ processors. In this paper we have used the standard approach of dynamic programming to solve the problem. The report consists of three sections: Section 1 gives the problem definition and states the standard algorithm given by Kucera. In section 2 we explain the reconfigurable architecture, its variants and present some basic results. In Section 3 we present the PARBS implementation of the problem.

Section 1:

In this section we present the definition of the problem, the standard algorithm by Kucera.

Definition: Let $G = (V, E)$ be a directed graph with *n* vertices. let *M* be a cost adjacency matrix for *G* such that *M(i, i) = 0,* $1 \le i \le n$. *M(i, j)* is the length or cost of edge $\langle i, j\rangle$ if $\langle i, j\rangle \in E(G)$ and *M(i, j)* $= \infty$ if $i \neq j$ and $\langle i, j\rangle \notin E(G)$. The all pairs shortest path problem is to determine a matrix *A* such that *A(i, j)* is the length of the shortest path from *i* to *j*.

The following algorithm given by Kucera[17], finds the shortest path between every pair of vertices of a weighted graph. The algorithm takes the edge weighted matrix *M* of a directed graph *G* and outputs a matrix *A* defined as follows:

$$A(i, j) = 0,\ i = j$$

$$A(i, j) = \min\{M(i_0, i_1) + M(i_1, i_2) + \dots + M(i_{k-1}, i_k)\},\ i \neq j$$

Where the minimum is taken over all possible sequences $i_0, i_1, i_2, \dots, i_k$ for which $i = i_0$ and $j = i_k$. Obviously, *A(i, j)* is the length of the shortest path from *i* to *j*. The iterative algorithm works in stages. In each iteration the search for the possible shortest path between each pair of vertices is extended to consider all paths utilizing up to twice the number of currently considered edges. Thus a logarithmic number of iterations are sufficient. In the W-RAM model of computation the algorithm uses matrices m_{ij} and q_{ijk} where *i, j* and *k* ranges from 1 to *n*.

Algorithm:

```
1. for all i, j in parallel do $m(i, j) \leftarrow M(i, j)$
2. repeat $\log_2$ n times
       begin
          for all i, j, k in parallel do $q(i, j, k) \leftarrow m(i, j) + m(j, k)$
          for all i, j in parallel do $m(i, j) \leftarrow min\{m(i, j), q(i, 1, j), q(i, 2, j), \ldots, q(i, n, j)\}$
       end
3. for all i, j in parallel do
       if $i \neq j$ then $A(i, j) \leftarrow m(i, j)$ else $A(i, j) \leftarrow 0$
```

Section 2: Computational Models

The basic idea of processor arrays with reconfigurable architecture was given by Miller R., V. K. Prasanna Kumar [18]. They described it as: “Meshes with reconfigurable bus consists of a VLSI array of processors overlaid with a reconfigurable bus systems”.

In the mid 90’s PARBS has drawn a lot of attention in the scientific community for its high performance computing with general purpose processors. Various models of PARBS appeared in the literature[4]. Common models are Bus automaton, Polymorphic torus network[14], Reconfigurable Meshes[18,19,15], Bypass capabilities etc.

An $N_1 \times N_2 \times \ldots \times N_r$ PARBS consist of an array of $N_1 \times N_2 \times \ldots \times N_r$ processors that is connected to a r-dimensional grid shaped reconfigurable bus system. The processing elements are connected to a bus through a fixed number of I/O ports. The ability to change the configuration of the bus system dynamically by adjusting local or global switches makes this architecture interesting to obtain various computational configurations like row, zig-zag, staircase & diagonal at run time.

A two dimensional processors array with a reconfigurable bus system of size N^2, consisting of identical processors, connected to a $N \times N$ rectangular mesh system, is called reconfigurable mesh. In figure 1, we see processing elements connected to a grid of buses and a PE with its four I/O ports and connection patterns.

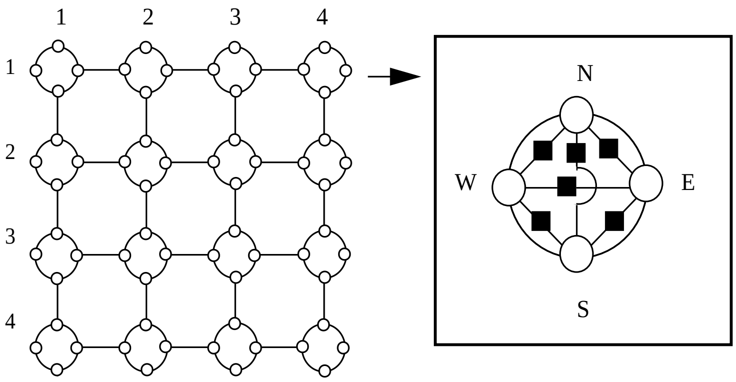


Figure 1. Reconfigurable Mesh of size 4x4

The basic computational unit of the reconfigurable mesh is the Processing Element (PE) which consists of switches, small storage and an ALU. A PE is capable of performing the following operations in one unit of time

1. Setting up a connection pattern
2. Read from or write onto a sub bus or memory
3. Performing logical and arithmetic operations
4. Disconnecting itself from the bus.

Reconfigurable bus models are characterized by the following parameters: [7]

- ***Width:*** It refers to the data width of the PE. There are two models which differ in the length of the operand of the PE.

  1. Bit model

  2. Word model

- ***Delay:*** It is the time needed to propagate a signal on the buses. The two models of PE’s are

  1. Unit delay model: “No matter how far signal has travelled”

  2. Logarithmic delay model: “O($\log_2 N$) time is needed”

- ***Bus Access:*** Each PE is connected to the bus through its port and will either read or write to it. There are two common models:

  1. ER Model (Similar to CREW PRAM)

  2. CR Model (Similar to CRCW PRAM)

- ***Connection pattern:*** Each PE can set the connection between its four ports based on local data or global instruction. There are 15 different connection patterns possible. Models differ in the number of connection patterns (a subset of 15) which they allow.

Various Models:

Based on these classifications, various models of reconfigurable bus system appear in the literature. Most of these models are synchronous in nature and permit unconditional global switch setting in addition to the local switch setting. Unconditional switch setting is performed by broadcasting a global instruction from a central controller.

These models differ in the way they are allowed to make internal connections, a few to note are:

PARBS (Processor Arrays with Reconfigurable Bus System)

RMESH (Reconfigurable Mesh)

RN (Reconfigurable Network)

Polymorphic Torus Network

- **PARBS:** The most general and most powerful model is PARBS**.** No restriction is placed on allowed connections. All **15** patterns of internal connections at each node (notation {xy} to mean that port x and y are connected to each other) are possible: [20]

•no Connections - { }

•two-port connections - {NS}, {EW}, {NW}, {NE}, {SW}, {SE}

•three-port connections - {EWS}, {EWN}, {SNE}, {SNW}

•four-port connections - {EWSN}

•two-pair connections - {EW, SN}, {EN, WS}, {ES, WN}

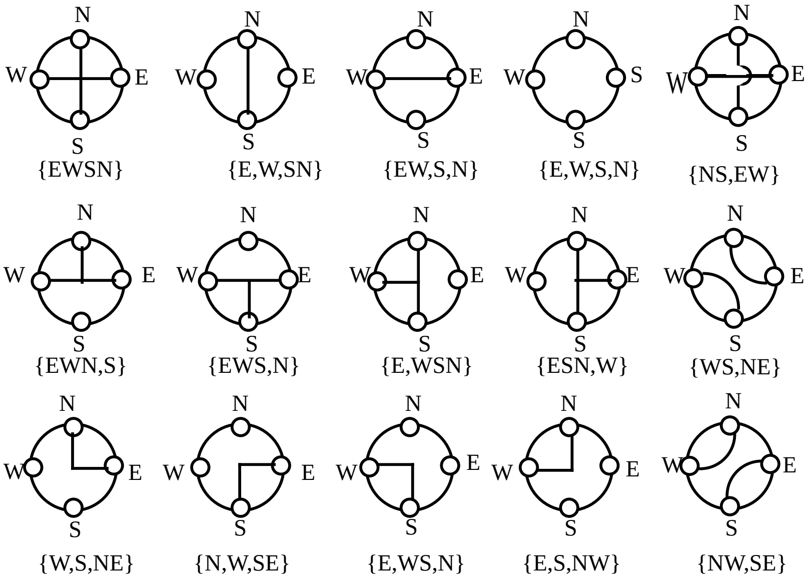


Figure 2. Connection patterns in PARBS

- **RMESH:** It is a two dimensional mesh where the PEs are located on the intersection of the grid lines of reconfigurable bus.

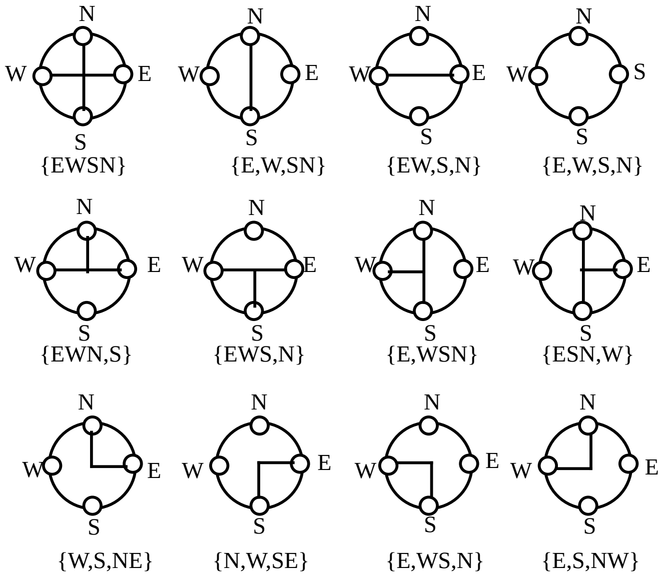


Figure 3. Connection patterns in RMESH

- **RN:** The Reconfigurable Network is a general model in which PEs may not lie at the grid point and a bus segment may join an arbitrary pair of PEs. [15]

“In RN model, each I/O port of a PE is connected to at most one other port”

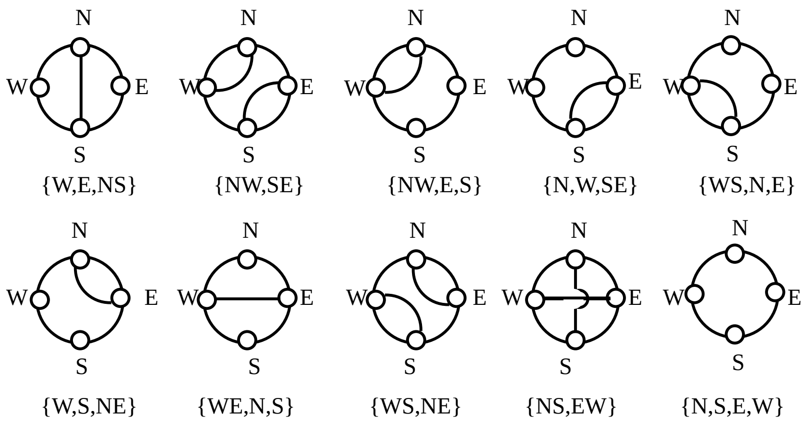


Figure 4. Connection pattern in RN

- **Polymorphic Torus:** It is identical to the PARBS except that the rows and columns of the underlying mesh wrap around.[14]

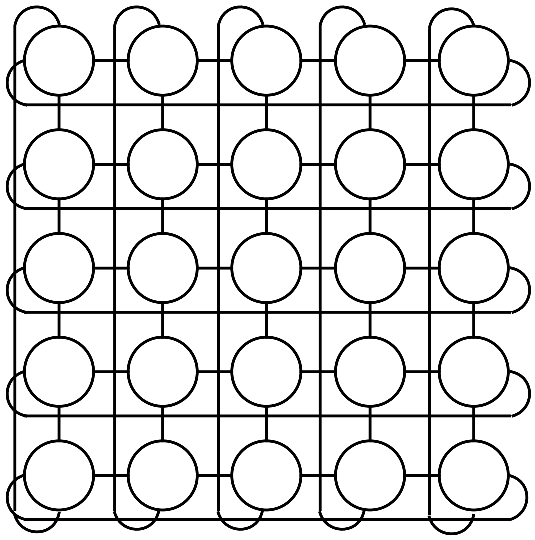


Figure 5. The 5 x 5 Torus

Higher dimensional PEs can be formed in the same way. Figure 6 (b) shows a PE in 3-dimension with six ports labelling (U)pper, (L)ower, (F)ront, (B)ack, (R)ight and (L)eft.

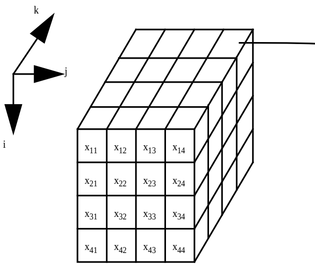


Figure 6(a). A 3-D configuration.

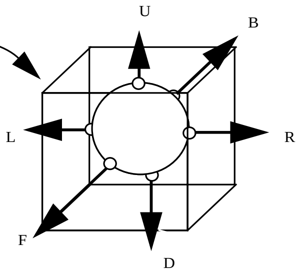


Figure 6(b). A PE in 3-D.

We assume that constant time is needed to broadcast values through the established buses irrespective of their distance from the first processor. Although it is a theoretical assumption and somewhat unrealistic on the current architecture, with the advancement in the fibre optics communication technology, this architecture is expected to gain wide popularity.

In designing an algorithm for the all pairs SPP, we use the unit delay model of 3-D PARBS. First we present the basic algorithms on PARBS and later use some of them in designing the final algorithm.

Before presenting lemmas we present a concept of splitting.

Splitting a bus is a technique which shows how the processors can exploit the ability to locally control the effective size of the subbuses.[19]

Lemma 1: Logical OR or AND of N bits can be obtained in constant time on a linear PARBS of size N. [19].

Lemma 2: Given a reconfigurable mesh of size N, in which no more than one processor in each column stores a data value, maximum(minimum) of these $(N^{1/2})$values can be determined in $\Theta(1)$ time using the unit delay model and in O(log$_2$n) time using the log-time delay model. [19].

Algorithm:

We assume that the *n* values, of which the minimum is to be obtained, are placed at the row 0 of the 2-D mesh.

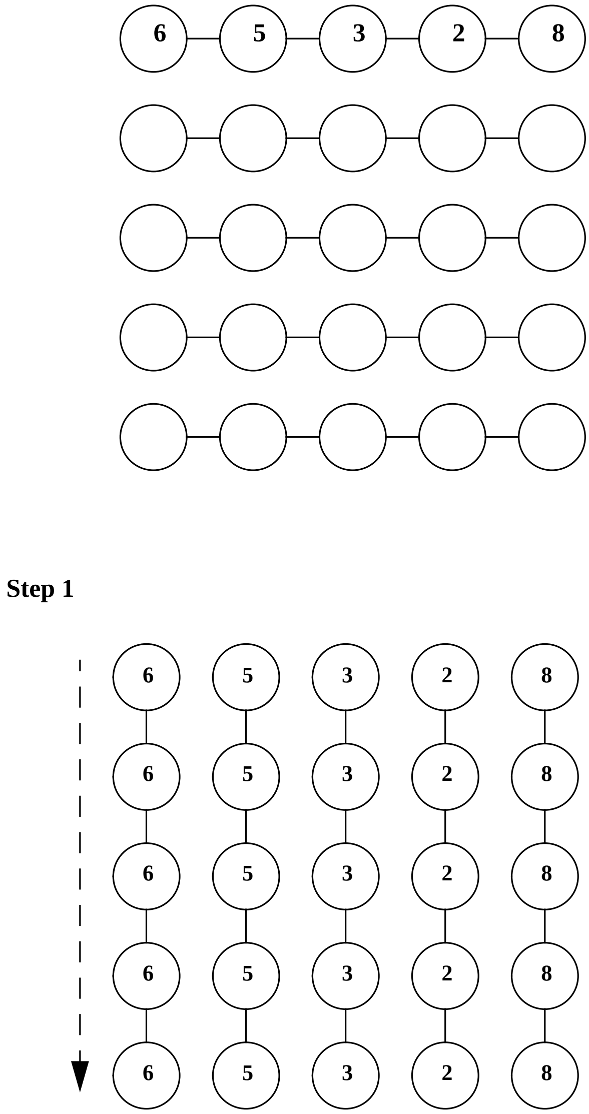
Step 1: A column broadcast is used so that every PE$_{i,j}$ contains entry $x_{j.}$

Step 2: Within each row i, processor $PE_{i,i}$ uses a row broadcast to inform all processors $PE_{i,j}$ the value of x_i, $0 \le i,\ j \le n-1$.

Step 3: Every processor computes the boolean result of “$x_j > x_i$”.

Step 4: In every column j, the logical OR of these values (In constant time) can be obtained to decide whether or not x_j is the minimum.

There may be more than one column having “0”, bus splitting on a row can be used to inform $PE_{0,0}$ the minimum value.



Step 1

Figure 7(a) and 7(b)

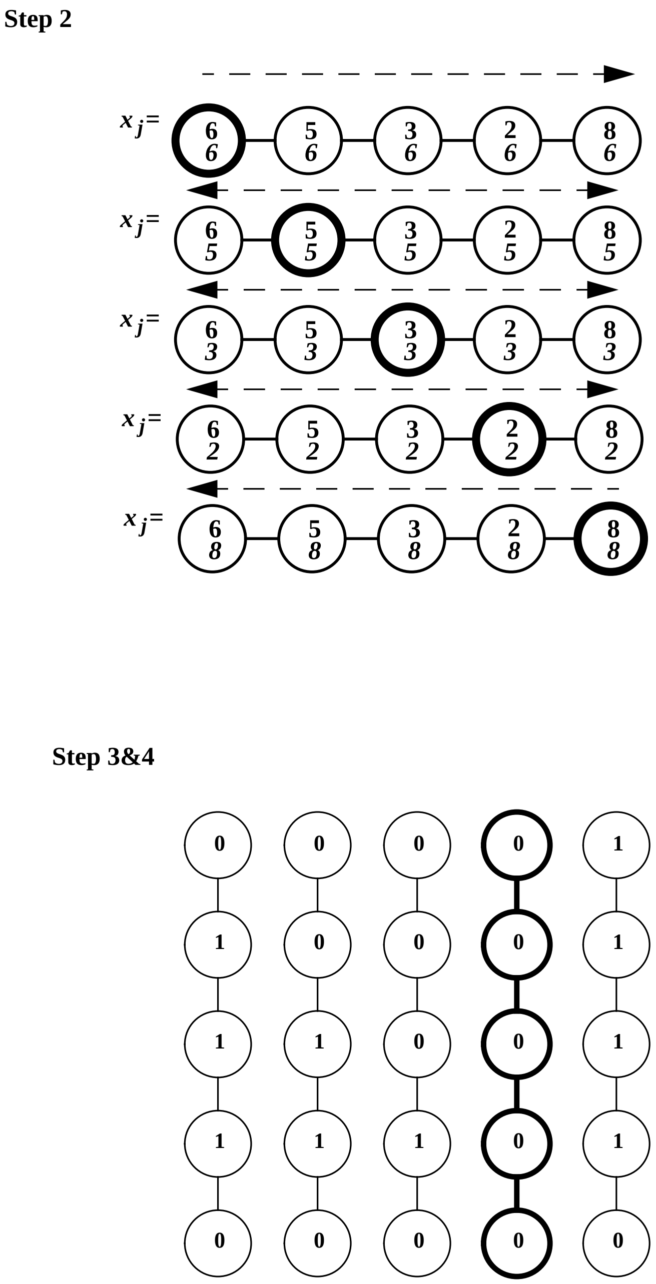

Step 2
x_j= 6 5 3 2 8 / 6 6 6 6 6
x_j= 6 5 3 2 8 / 5 5 5 5 5
x_j= 6 5 3 2 8 / 3 3 3 3 3
x_j= 6 5 3 2 8 / 2 2 2 2 2
x_j= 6 5 3 2 8 / 8 8 8 8 8
Step 3&4
0 0 0 0 1
1 0 0 0 1
1 1 0 0 1
1 1 1 0 1
0 0 0 0 0

Figure 8(a) and 8(b)

Lemma 3: Searching of a given number in a list of *n* numbers can be done in constant time on a linear PARBS of size *n*.

Proof:

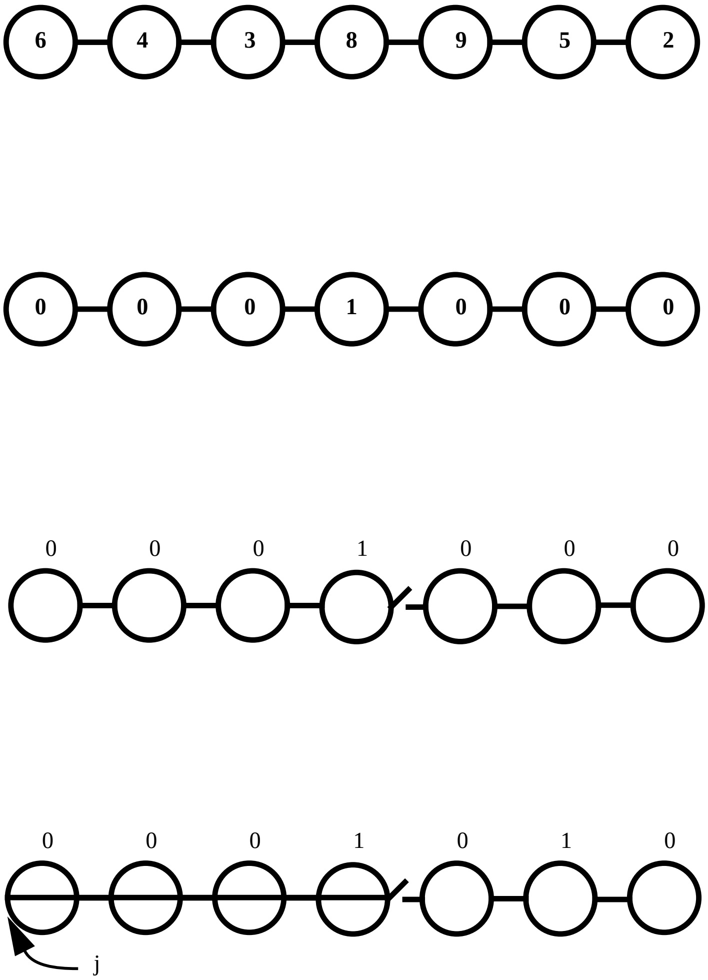
Assume that *n* numbers x_i, $1 \le i \le n$ are stored at *n* processors. The value “p” to be searched is available at PE(1). The steps involved are:

Step 1: Connect port *E* to *W* and broadcast “p” on the established bus, perform the operation “**if p = x(i)**”, which results in either 0 or 1.

Step 2: Each processor PE(i) that has “**1**” as its data value splits its bus by setting its eastern switch to disconnect its row bus.

Step 3: Each processor that has a “**1**” as its data value broadcasts the “**j**” on its sub bus. Processor PE(1) receives the western most “**j**” as search position of “p” in *n* values.

We explain it with one example for n = 7. The value to be searched is “p = 8”



Initially:

After step 1:

After step 2:

After step 3:

Figure 9(a-d)

Section 3:

PARBS algorithm for the all pairs SPP problem:

We assume that the values of m(i,j) are stored at the P((i-1)n+1,j,1), $1 \le i \le n$, $1 \le j \le n$ of $n^2 \times n \times n$ PARBS.

repeat step 1-7, (log$_2$ n) times

Step 1: Establish sub buses in *k*-direction. Processors P((i-1)n+1,j,1) broadcast values of m(i,j) on the buses and then break up the sub buses.

Step 2: Establish sub buses in j-direction (connect port *E* to *W*) and broadcast values of m(i,j) from processors P((i-1)n+1,j,j) to P((i-1)n+1,i,j), $1 \le i \le n$, $1 \le j \le n$.

Step 3: Disconnect sub buses and establish sub buses in *i*-direction (Connect port *U* to *D*). Broadcast values of m's, received in the step 2, from processors P((i-1)n+1,i, j), $1 \le i \le n$, $1 \le j \le n$ on the established sub buses which is received by processors P((k-1)n+1,i,j), $1 \le i \le n$, $1 \le j \le n$, $1 \le k \le n$. To end this, values of m(i,j) and m(j,k) are available at P((i-1)n+1, j, k), $1 \le i \le n$, $1 \le j \le n$, $1 \le k \le n$.

Step 4: Find the sum of m(i,j) and m(j,k) at every processor in constant time.

Step 5: Establish sub buses in *j*-direction (connect port *E* to *W*), find minimum of these *n* values row wise, in constant time. Store the minimum at processor P((i-1)n+1,1,j) in some variable "x", $1 \le i \le n$, $1 \le j \le n$.

Step 6: Breakup sub buses and establish sub buses in *j*-direction, broadcast values of "x" from P((i-1)n+1,1,j) to P((i-1)n+1,j,j), $1 \le i \le n$, $1 \le j \le n$.

Step 7: Disconnect established sub buses and connect sub buses in *k*-direction, broadcast value of "x" from P((i-1)n+1,j,j) to P((i-1)n+1,j,1), $1 \le i \le n$, $1 \le j \le n$ and perform operation m(i,j) = minimum{m(i,j), x} at respective processors.

Lemma 4: The shortest path between each pair of vertices of a directed graph, stored in a 3-D PARBS can be obtained in O(log$_2$ n) time using $n^2 \times n \times n$ processors.

Proof: The proof is trivial as it can be easily seen that the steps (1-7) takes constant time and the loop is repeated (log$_2$ n) times.

Next we present the execution of the algorithm on a directed graph of figure 10(a).

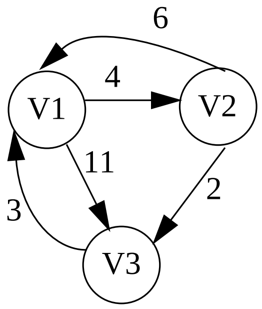


Figure 10(a). Graph G

	1	2	3
1	0	4	11
2	6	0	2
3	3	inf	0

Figure 10(b). Cost matrix M

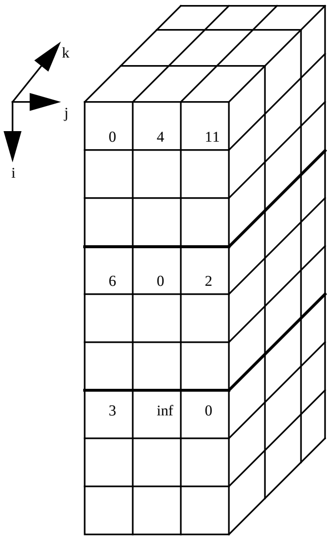
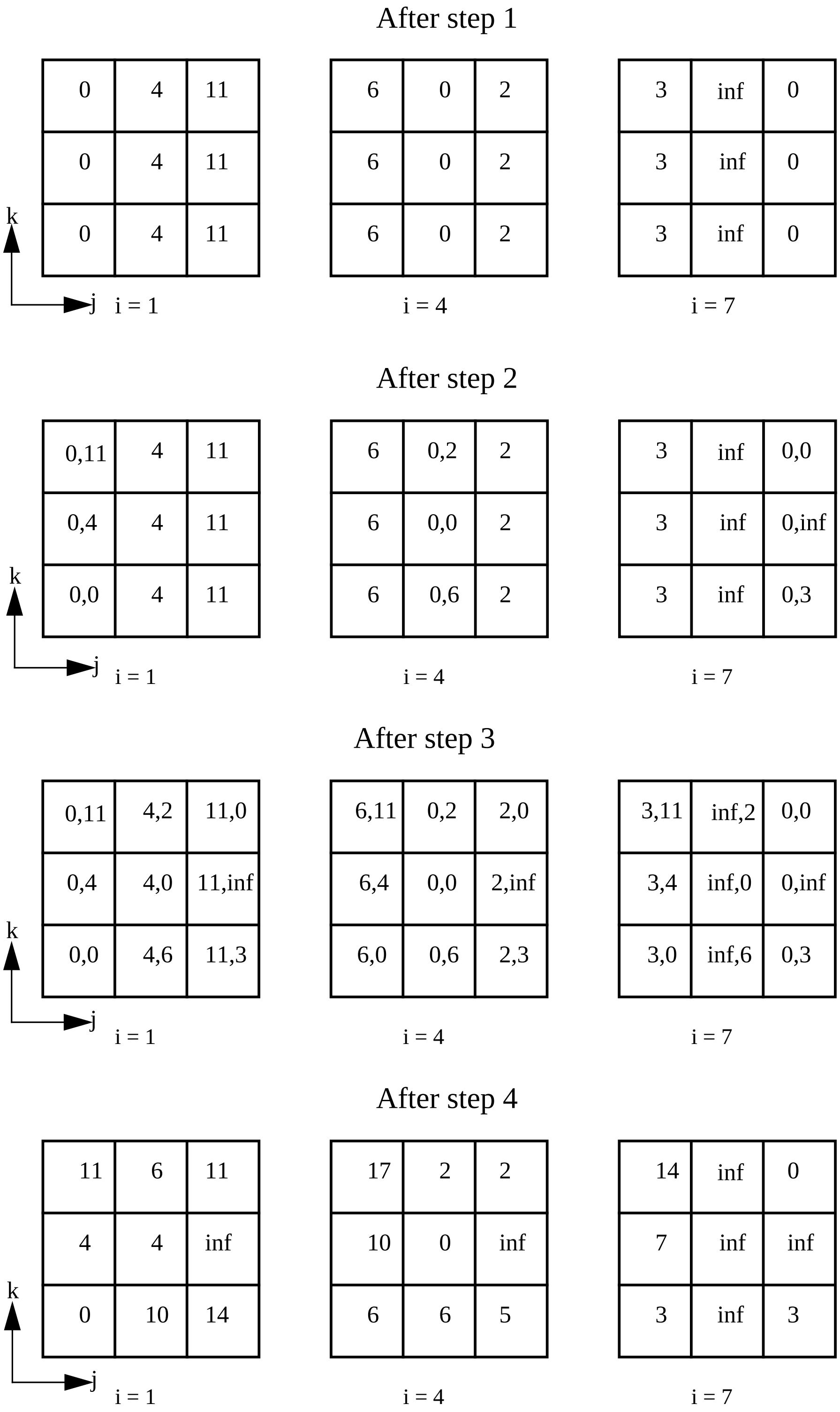


Figure 11. n^2 x n x n PARBS



After step 1

i = 1

0	4	11
0	4	11
0	4	11

i = 4

6	0	2
6	0	2
6	0	2

i = 7

3	inf	0
3	inf	0
3	inf	0

After step 2

i = 1

0,11	4	11
0,4	4	11
0,0	4	11

i = 4

6	0,2	2
6	0,0	2
6	0,6	2

i = 7

3	inf	0,0
3	inf	0,inf
3	inf	0,3

After step 3

i = 1

0,11	4,2	11,0
0,4	4,0	11,inf
0,0	4,6	11,3

i = 4

6,11	0,2	2,0
6,4	0,0	2,inf
6,0	0,6	2,3

i = 7

3,11	inf,2	0,0
3,4	inf,0	0,inf
3,0	inf,6	0,3

After step 4

i = 1

11	6	11
4	4	inf
0	10	14

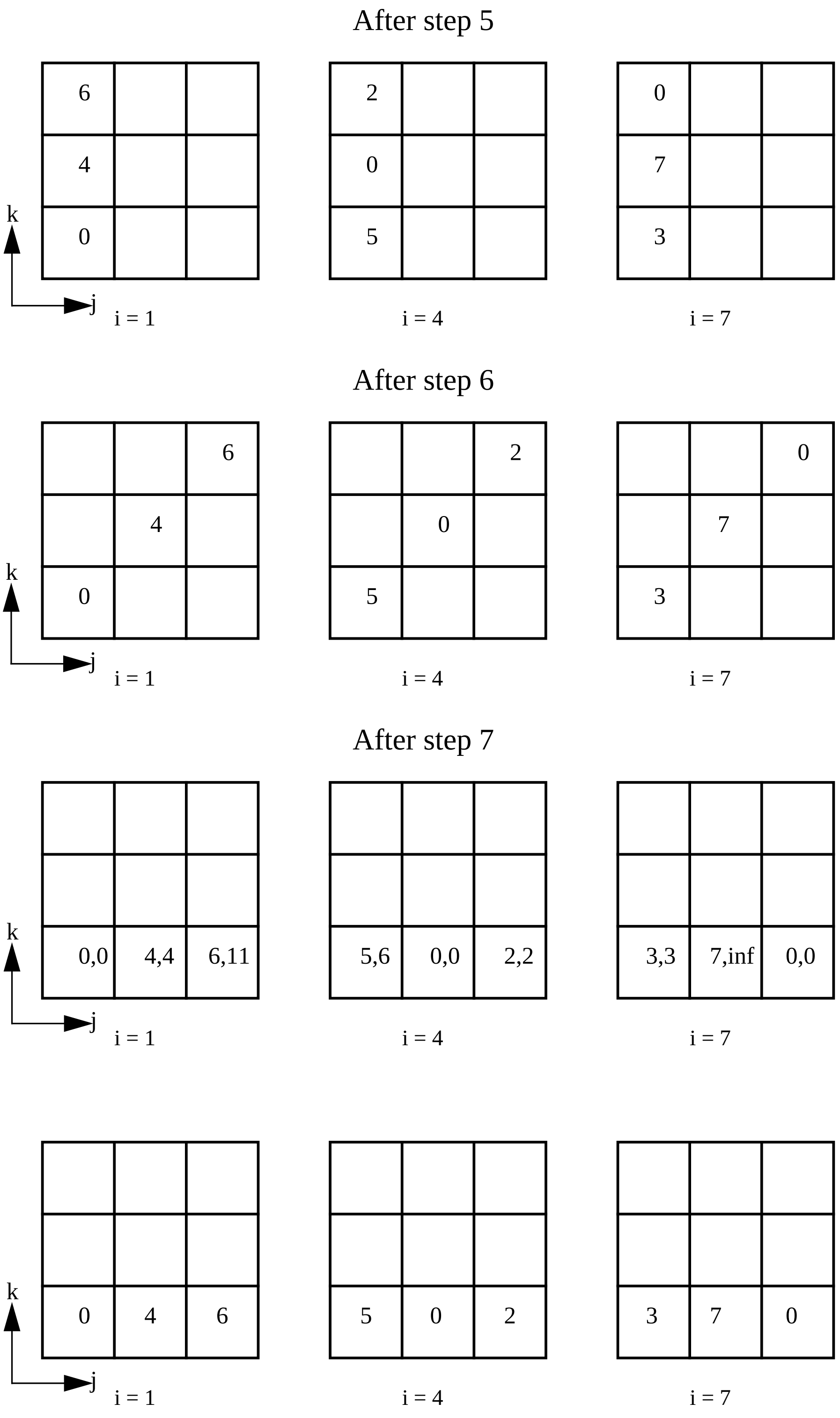
i = 4

17	2	2
10	0	inf
6	6	5

i = 7

14	inf	0
7	inf	inf
3	inf	3

Figure 12 (a-d)


After step 5

After step 6

After step 7

Figure 13(a-d)

Conclusion:

In this technical report we have presented two algorithms on the PARBS namely an optimal constant time searching algorithm and an efficient parallel algorithm for the solution of the all pairs shortest path problem. The SPP algorithm is considered as an efficient algorithm since it belongs to Nick's class [11]. This algorithm has parallel time complexity of O($\log_2$n) and processors complexity is $n^2 \times n \times n$. The algorithm presented in this report has the same time and processor complexity which has been reported in [9]. The work can further be extended to get the solution of this problem in constant time or an optimal parallel algorithm by reducing the number of processors.

References:

[1] Aggarwal A., "Optimal Bounds for Finding Maximum on Array of Processors with k Global Buses"**, IEEE Transaction on Computers,** pp. 62-64, Vol. c-35, No. 1, January 1986.

[2] Aho A., J. Hopcroft and J. Ullman, **The Design and Analysis of Computer Algorithms,** Addison-Wesley, 1974.

[3] Bauer B. E., **Practical Parallel Programming,** Academic Press Inc., 1992.

[4] Biing-Feng Wang and Gen-Huey Chen, "Constant Time Algorithms for the Transitive Closure and Some Related Problems on Processor Arrays with Reconfigurable Bus Systems", **IEEE Transaction on Parallel and Distributed Systems,** pp. 500-507, Vol. 1, No. 4, October 1990.

[5] Biing-Feng Wang, Gen-Huey Chen and Ferng-Ching Lin, "Constant Time Sorting on a Processor Array with Reconfigurable Bus System", **Information Processing Letters,** pp. 187-192, 34(1990).

[6] Bokhari S. H., "Finding Maximum on an Array Processor with a Global Bus", **IEEE Transaction on Computers,** pp. 133-139, Vol. c-33, No. 2, February 1984.

[7] Bondalpati K. and V.K. Prasanna Kumar, "Reconfigurable Meshes: Theory and practice", **Reconfigurable Architectures Workshop, International Parallel Processing Symposium, April 1997.**

[8] Fragopoulou. P., "On the Efficient Summation of N Numbers on an N-Processor Reconfigurable Mesh", **Parallel Processing Letters, Vol. 3 No.1 (1993) 71-78.**

[9] G-H Chen, B-F Wang and C.J. Lu, "On the parallel computation of the algebraic path problem", **IEEE Transaction on Parallel and Distributed Systems,** pp. 251-256, Vol. 3, No. 2, March 92.

[10] G-H Chen, S. Olariu et al., "Constant Time Tree Algorithms on Reconfigurable Meshes on Size n x n", **Journal of Parallel and Distributed Computing,** pp. 137-150, 26 (1995).

[11] Gibbons A. And Wojciech Rytter, **Efficient Parallel Algorithms,** Cambridge University Press, Cambridge, May 1988.

[12] Golub G.H. and James Ortega, **Scientific Computing: An Introduction with Parallel Computing,** Academic Press Inc., 1993.

[13] Horowitz E. And S.Sahni, **Fundamentals of Computer Algorithms,** Addison-Wesley, Computer Science Press, NY, 1978.

[14] Hungwen Li and Massimo Maresca, "Polymorphic Torus Network", **IEEE Transaction on Computers,** pp. 1345-1351, Vol. 38, No. 9, September 1989.

[15] Ju-Wook J. and V.K. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Mesh", **Journal of Parallel and Distributed Computing,** pp. 31-41, 25 (1995).

[16] K. Li, Y. Pan and S.Q. Zheng, "Fast and Efficient Parallel Matrix Manipulation on a Linear Array with a Reconfigurable Pipelined Bus System", **High Performance Computing Systems and Applications,** J.Schaeffer and R.Unrau eds., Kluwer Academic Press, 1998.

[17] Kucera L., "Parallel Computation and Conflict in Memory Access", **Information Processing Letters Vol. 14, number 2, April 1982.**

[18] Miller R., V.K. Prasanna Kumar, Dionisios Reisis and Quentin F. Stout, "Meshes with Reconfigurable Buses", **Proc. 15th MIT Conference on Advance Research in VLSI,** pp. 163-178, March 1988.

[19] Miller R., V.K. Prasanna Kumar, Dionisios Reisis and Quentin F. Stout, "Parallel Computations on Reconfigurable Meshes", **IEEE Transactions on Computers, Vol. 42, No. 6, June 1993.**

[20] Ten-Hwang Lai, M.J.Sheng, "Triangulation on Reconfigurable Meshes: A Natural Decomposition Approach", **Journal of Parallel and Distributed Computing,** 38-51, 30 (1995).

[21] Vipin Kumar et al., **Introduction to Parallel Computing, Design and Analysis of Algorithms,** The Benjamin Cummings Publishing Company Inc., California, 1994.

[22] Wankar R., "Parallel Algorithms for Solving Symmetric Positive Definite System (SPD) of Equations", **International Journal of Management and Systems,** pp. 311-324, Vol. 11, No. 3, Sept.-Dec 95.

[23] Wankar R., E. Fehr and N. S. Chaudhari, "A Fast Parallel Algorithm for Special Linear Systems of Equations using Processor Arrays with Reconfigurable Bus Systems", **Technical Report B-2-99,** January 29, 1999, Freie Universität Berlin, Germany.