Permutations on Superposed Parallel Buses

(Version 2)

Bruce W. Arden

College of Engineering and Applied Science
University of Rochester
Rochester, N.Y. 14627

Toshio Nakatani

Department of Computer Science
Princeton University
Princeton, N.J. 08544

CS-TR-024-86

May, 1986

Index Terms: Area-time trade-off, bus-connected arrays, interconnection networks, mesh-connected computer, parallel processing, permutations, VLSI complexity.

Permutations on Superposed Parallel Buses

Bruce W. Arden

College of Engineering and Applied Science
University of Rochester
Rochester, N.Y. 14627

Toshio Nakatani

Department of Computer Science
Princeton University
Princeton, N.J. 08544

ABSTRACT

This paper presents two schedules for arbitrary permutations on the square grid of *superposed parallel buses.* In the first, called *a self-pipelined* schedule, row-column broadcasts are pipelined for a single permutation. Because broadcast-ordering involves the control algorithms similar to those for setting a Benes rearrangeable network, this schedule is only applicable for static permutations. *A multi-pipelined* schedule is based on fixed-order row broadcasts of one permutation and fixed-order column broadcasts of another permutation. More than one permutation is in process in the pipeline. Because of the fixed broadcast order, this schedule is applicable for dynamic permutations. These two schedules are area-efficient and almost optimal for permutations using the AT^2 measure. They take a shorter cycle time than previously reported interconnection schemes using almost linear area. They are especially suitable for highspeed but area-limited technologies because of the simple hardware control algorithm and the regular, area-efficient interconnections. Descriptions of these two schedules and a summary of their VLSI complexities and minimum I/O bandwidths in comparison with other linear area interconnections are included.

May, 1986

Permutations on Superposed Parallel Buses

Bruce W. Arden

College of Engineering and Applied Science
University of Rochester
Rochester, N.Y. 14627

Toshio Nakatani

Department of Computer Science
Princeton University
Princeton, N.J. 08544

1. Introduction

Area-efficient permutation is one of the most important issues in parallel supercomputing (Gottlieb and Kruskal[1984], Johnsson[1985], Leiserson[1983], Schwartz[1980], and Ullman[1984]). Although a crossbar (Pippenger[1975]) is the most powerfull interconnection scheme, it requires $O(N^2)$ area for interconnecting N processors because of $O(N^2)$ switches. A rearrangeable, multistage interconnection network (Benes[1965]) has a reduced number of switches, $O(NlogN)$, but it still requires $O(N^2)$ area because of the quadratically growing wire area of shuffle connections (Franklin[1981], Rosenberg[1983], and Wise[1981]). The interconnection schemes based on the perfect shuffle (Stone[1971]) or the hypercube share the same disadvantages.

On the other hand, a time-multiplexed linear bus takes $O(N)$ area for static permutation and $O(N\log^2 N)$ area for dynamic permutation, but a single permutation takes $O(N)$ time. Tree and Ring interconnections have the same area and time performance as a linear bus. A mesh connection (Orcutt[1976]) also takes the same area but has better time performance than those using almost linear area. For static permutation, Thompson[1977] has presented a $(7\sqrt{N}-8)$ time routing method by simulating the binary Benes permutation

network (Waksman[1968]) on a mesh-connected parallel computer. For dynamic permutation, Thompson and Kung [1977] have suggested a $4(\sqrt{N}-1)$ routing method by simple sweeps on a mesh-connected computer. For bit-permuted-complement permutations, Nassimi and Sahni[1980] have also shown an optimal routing algorithm on a mesh-connected computer using $O(log^2 N)$ preprocessing time.

Recently, the authors have found a faster routing algorithm on the interconection schemes based on *superposed parallel buses (SPB)* (Arden and Nakatani[1985]), which use $O(N\log^2 N)$ area for static permutation. It is essentially a simulation of the $N=n^2$ three-stage Benes permutation network (Benes[1962]). This approach takes $3\sqrt{N}$ time and $\sqrt{N}$ time if it operates in pipelined fashion for multiple permutations. Here, the basic unit of time, or cycle, is the time to broadcast one packet on a bus. In essence, this is the unit delay assumption (Thompson[1979]).

In this paper, we present two schedules for arbitrary permutations on the square grid of *superposed parallel buses* (Figure 1.1). In *a self-pipelined schedule,* row-column broadcasts are pipelined for a single permutation. Because broadcast-ordering involves the control algorithms similar to those for setting a Benes rearrangeable network, this schedule is only applicable for static permutations. On the other hand, *a multi-pipelined schedule* is based on fixed-order row broadcasts of one permutation and fixed-order column broadcasts of another permutation. More than one permutation is in process in the pipeline. Because of the fixed broadcast-orper, this schedule is applicable for dynamic permutation.

For a single permutation, *a self-pipelined* schedule takes $\sqrt{N}+1$ cycles and *a multi-pipelined* schedule takes $2\sqrt{N}$ cycles. For sequential or multiple permutations, both schedules take $\sqrt{N}$ cycles in pipelined fashion. These two

schedules are especially suitable for highspeed but area-limited technology because of the simple hardware control algorithm and the regular, area-efficient interconnections. In this case, the broadcasting chips and the buses operate at a much faster cycle than the processors. The pipelined grid interconnection (Figure 1.2) and the design of the broadcasting chips (Figure 2.4 and 3.2) to implement the two schedules are considered.

In section 2 and 3, the two schedules of the SPB are described in detail. In section 4, VLSI complexities of the SPB, such as area, time, and AT^2, are discussed. In section 5, other linear area interconnections, such as the mesh (Figure 1.3) and the torus (Figure 1.4), are described in terms of the performance of permutations. In section 6, minimum I/O bandwidth is discussed and compared for those interconnections. In section 7, comparison tables are summarized including area, time, AT^2, and minimum I/O bandwidth of three linear area interconnections described in this paper for both static and dynamic permutation.

2. A Self-pipelined Schedule of the SPB for Static Permutation

In this schedule (the hardware control algorithm 2.1), the broadcast-order must be precalculated in such a way that all the row and the column buses are continuously busy during the given permutation. This mode of operation is called *self-pipelined* in this paper, since row broadcasts and column broadcasts are pipelined for a single permutation on superposed parallel buses (Figure 2.1). The following theorem states that it is always possible to consecutively select a packet with distinct target column addresses from each row and it takes $n+1$ cycles to complete an arbitrary permutation of $N=n^2$ data items on a two dimensional $n \times n$ square grid of superposed parallel buses.

Theorem 2.1: There exists a schedule for an arbitrary permutation of $N=n^2$ data items in $n+1$ cycles on a two dimensional $n\times n$ square grid of superposed parallel buses.

Proof: Consider $n\times n$ array of column destinations. There are exactly n occurences of each of the n column destinations in the array. This satisfies the hypothesis of Hall's theorem on "distinct representatives" (Hall[1935]). That is, for any k $(1\leq k\leq n)$, there are at least k distinct column destinations on any k rows. Therefore, there exists a selection of distinct column destinations from each row. After selection and removal of distinct representatives, Hall's theorem is again satisfied with the n rows now containing $n-1$ column destinations. Therefore, there continues to exist a selection of distinct column destinations from each row until exausted. Thus, all the packets on the rows can be broadcast in n cycles, and all the packets reach their destinations in $n+1$ cycles by interleaving operations of row and column broadcasts (Figure 2.1).□

However, it is a time-consuming operation to select a set of distinct representatives (or equivalently to find a maximum matching on the bipartite multigraph). The best known sequential algorithm for the case of n sets with n elements each takes $O(n^{5/2})$ time (Papadimitriou and Steiglitz[1982]). Therefore, by using this algorithm repeatedly, the whole broadcast-order can be obtained for a single permutation in a total of $O(n^{7/2})$ time. On the other hand, *a self-pipelined* schedule is regarded as the minimum edge-coloring of the bipartite multigraph. That is, this schedule is considered to be equivalent to a time-division-multiplexing (Andresen and Harrison[1972] and Marcus[1972]) of a n^2 Benes-Clos rearrangeable network (Figure 2.2) and also to be equivalent to a time-multiplexed crossbar (Figure 2.3). Therefore, the related control algorithm can be used to select the sets of distinct representatives. However, the best known sequential algorithm to set the switches of a n^2 Benes-Clos

rearrangeable network takes $O(n^2 \log n)$ (Opferman and Tsao-Wu[1971] and Andresen[1977]). An $O(log^4 n)$ time parallel algorithm (Nassimi and Sahni[1982] and Lev, Pippenger, Valiant[1981]) has also been described. Therefore, this schedule is primarily applicable for static permutations.

An implementation of this schedule on the pipelined grid interconnection is described below. First, every processor puts a packet in its broadcasting chip (Figure 2.4). A packet (Figure 2.5) is composed of the broadcast order, the destination address in two parts, and the data. The broadcasting sequence for the row buses is prescheduled and known by the broadcast order of a packet. Whenever the content of the broadcast order is matched by the modulo n clock counter, a chip will broadcast a packet on the row bus after stripping the broadcast order from the packet. Whenever the first half of the destination address is matched by its own location, a chip moves a packet from the row bus into the input and output register. In the following cycle, the chip strips the first half of the destination address from the packet and broadcasts the remainder on the column bus. A chip on the column bus moves a packet to the input & output register, whenever the latter part of the destination address of a packet is matched by its own location. After the first broadcast, the row and column buses are used simultaneously. Finally, every processor receives its data specified by the given permutation after $n+1$ cycles.

Finally, we note that the self-pipelined schedule is optimal for arbitrary static permutations on a grid of superposed parallel buses (Arden and Nakatani[1986]).

3. A Multi-pipelined Schedule of the SPB for Dynamic Permutation

In this schedule (the hardware control algorithm 3.1), the broadcast-order is fixed so that the broadcast operations on the row and the column buses for one permutation are not overlapped and thus not pipelined in itself. Instead the column broadcasts of one permutation are overlapped with the row broadcasts of the following permutation (Figure 3.1). Thereby, all the buses on the rows and the columns are kept busy during consecutive permutations. This mode of operation is called *multi-pipelined* in this paper, since multiple permutations are pipelined. This schedule can be used for dynamic permutations since the broadcast-order is fixed and independent of permutations. However, n buffers on each broadcasting chip (Figure 3.2) are needed for the pipeline operation of this schedule. The following theorem states that n buffers are necessary and sufficient to implement an arbitrary permutation of $N=n^2$ data items on a two dimensional $n\times n$ square grid of superposed parallel buses.

Theorem 4.1: There exists a dynamic schedule for an arbitrary permutation of $N=n^2$ data items in $2n$ cycles on a two dimensional $n\times n$ square grid of superposed parallel buses and n buffers are necessary and sufficient.

Proof: Since there are n packets on each row, all the packets can reach their destination columns by n row broadcasts (that is, in n cycles). There are exactly n packets on each column after n row broadcasts. Therefore, n column broadcasts (that is, in n cycles) can convey all the packets to their destinations. Row and column broadcast-orders are both fixed, independent of permutations. That is, it takes $2n$ cycles to permute n^2 data items dynamically on $n\times n$ square grid of superposed parallel buses. In the worst case of permutations, as many as n packets must be received by the buffer of a single processor on a destination column. Therefore, n buffers are necessary. On the other hand, for any k $(1\leq k\leq n)$, after k column broadcasts, at most $n-k$ buffers remain filled for

transmissions on columns and at most k new packets have been received from row transmissions. Therefore, n buffers are sufficient.□

An implementation of this schedule on the pipelined grid interconnection is described below. First, every processor puts a packet in its broadcasting chip. A packet (Figure 3.3) is composed of the destination address and the data. Whenever the modulo n clock counter is matched by its own location in the row, a chip broadcasts a packet on the row bus. Whenever the first half of the destination address of the broadcast packet is matched by its own location on the row, the chip moves a packet to the input and output buffer. After n cycles, the row broadcast operations are completed and some chips may have at most n packets. Then, the chips start broadcasting on the column bus. Whenever the modulo n clock counter is matched by the last half of the destination address of the packets in the buffer, the chip broadcasts the packet on the column bus after stripping the destination address of the packet. Whenever the modulo n clock counter is matched by its own location in the column, a chip moves a packet to the input and output register. After n cycles, the column broadcast operations are completed. Every processor receives a packet specified by the given permutation after $2n$ cycles. Since the row and the column operation can be pipelined for multiple permutations, every processor can send and receive a new packet every n cycles after all the row and the column buses are busy.

Finally, we note that a buffering approach has been studied in the context of space-time equivalence in communication networks for real time telephone traffic control. For example, the queueing crossbar (Marcus and McDonald[1969]) contains a time-division queue (a set of shift registers) at each cross point of a crossbar to switch digitally encoded speech information (Figure 3.4). *A multi-pipelined* schedule is also considered to be equivalent to a time-

division-multiplexing of a n^2 Ω-network (Lawrie[1975]) (Figure 3.5).

4. VLSI Complexities of the SPB

VLSI area, time, and AT^2 measures for the two schedules are discussed in this section. The area is measured in both communication area and total area. The communication area is the greater concern when the implementation involves only buses and the broadcast control is not implemented as separate chips but is instead programmed within the existing processors. On the other hand, the total area is of more concern when the implementation involves both buses and highspeed, but area-limited, broadcast chips.

In slightly more detail, the area measures considered are only those dependent on N and they are presented as two components. The area due to the horizontal and vertical buses is called the communication area. When registers whose size is dependent on N are used in the processors in the multi-pipelined case, the associated area is added to the communication area. The complexity order is of course determined by the dominant term in this sum. The horizontal and vertical buses in the case of the self-pipelined case and the horizontal buses in the case of the multi-pipelined case are $O(logN)$ width. A connection between one of these buses and a processor occupies $O(log^2N)$ area (Figure 4.1). Therefore, the basic communication area is $O(Nlog^2N)$ in both cases. In the self-pipeline case, a processor area is $O(logN)$ due to the counter and register of size $O(logN)$. Therefore, the total processor area is $O(N\log N)$, which is dominated by $O(Nlog^2N)$ communication area. Note that it is possible to reduce the communication area to $O(N)$ but at the cost of the increased total area of $O(N^{3/2})$. In the multi-pipelined case, the addition of $\sqrt{N}$ buffers of size $O(logN)$ at each processor adds area proportional to $N\times\sqrt{N}\log N$ to the total and this term dominates $O(N\log^2N)$ communication area.

The time is measured by the number of broadcasting steps on the buses. The basic unit of time, or cycle, is the time to broadcast one packet on a bus. That is, we assume the unit delay model based on Thompson[1979]. In the self-pipelined case, a single permutation takes $\sqrt{N}+1$ cycles. In the multi-pipelined case, a single permutation takes $2\sqrt{N}$ cycles. The time and area measures are combined in the obvious way to produce the AT^2 measures.

5. Other Almost Linear Area Interconnections

The other interconnection networks that can be embedded in almost linear area are a linear bus, ring, tree, mesh, and torus. For a linear bus, ring, and tree, a permutation takes at least $O(N)$ time. On the other hand, a permutation takes $O(\sqrt{N})$ time for a mesh and torus. For a permutation of N items, $AT^2=\Omega(N^2)$ due to the lower bound technique using the bisection argument based on Thompson[1979]. When a permutation is performed with linear area, that is $A=O(N)$, then at least $T=\Omega(\sqrt{N})$ is required. That is, the interconnection networks based on a grid, such as a SPB, mesh, torus, are optimal for permutation in linear area.

On a mesh-connected computer (Figure 1.3), Thompson[1977] has presented a $(7\sqrt{N}-8)$ time (SIMD) routine method by simulating the binary Benes permutation network (Waksman[1968]) for static permutation. Here, the basic unit of time is the time for a unit route, that is, the time to pass a packet from a processor to one of the four nearest-neighbour processors. The communication area in this case is simply $O(N)$ since a packet of constant length (no address portion is required) is transferred. On the other hand, a processor area is $O(logN)$ due to both $O(1)$ instructions of size $O(logN)$ and $O(logN)$ data corresponding the switch settings. That is, the total processor area is $O(NlogN)$, which dominates $O(N)$ communication area. For multiple permutations, one permutation

cannot overlap with another in this method. Therefore, the pipeline performance is the same as the one for a single permutaion.

For dynamic permutation, Thompson and Kung [1979] have presented $4(\sqrt{N}-1)$ routing method by simple sweeps, which uses $O(\sqrt{N})$ memories for each processor. The communication area in this case is $O(Nlog^2N)$ due to the horizontal and vertical buses of width $O(logN)$. On the other hand, a processor area is $O(\sqrt{N}\log N)$ due to $O(\sqrt{N})$ buffers of size $O(logN)$. That is, the total processor area is $O(N^{3/2}\log N)$, which dominates $O(Nlog^2N)$ communication area. For multiple permutations, the row sweep of one permutation can overlap with the column sweep of another. Therefore, the pipeline performance is $2(\sqrt{N}-1)$ time.

On a torus interconnection (Figure 1.4), which is a mesh with wrap-around connections, we can simply apply the same method as a mesh-connected computer for static permutation. In this case, area and time complexities are the same as a mesh-connected computer. For multiple permutations, the pipeline performance is the same as the one for a single permutation. For dynamic permutation, we can use a $2(\sqrt{N}-1)$ time routing method using a single $(\sqrt{N}-1)$ horizontal rotation and a single $(\sqrt{N}-1)$ vertical rotation on a torus, which also uses $O(\sqrt{N})$ memories for each processor. In this case, area complexity is the same as a mesh-connected computer for dynamic permutation, but time complexity is improved. For multiple permutaions, the row rotation of one permutation can overlap with the column rotation of another. Therefore, the pipeline performance is $\sqrt{N}-1$ time.

6. Minimum I/O Bandwidth

Minimum I/O bandwidth is determined by the minimum amount of data required to start and keep operating a permutation every cycle with no idle cycles. For example, dynamic permutation for a mesh-connected computer and both static and dynamic permutation for SPB require $O(\sqrt{N})$ new data every cycle. On the other hand, static permutation for a mesh-connected computer and both static and dynamic permutation for a torus requires $O(N)$ new data for the first cycle. On a torus, if only $O(\sqrt{N})$ new data is available every cycle, a dynamic permutation takes $4(\sqrt{N}-1)$ cycles as in the case of a mesh-connected computer.

7. Comparison Tables

VLSI area, time, AT^2, and minimum I/O bandwidth of the SPB, the mesh-connected computer, and the torus are summarized in tabular forms for comparisons of both static and dynamic permutation.

Communication Area (Bus area dependent on N)		
The Interconnections	Static Permutation (without buffers)	Dynamic Permutation (with buffers)
SPB	$O(Nlog^2N)$	$O(Nlog^2N)$
Mesh/Torus	$O(N)$	$O(Nlog^2N)$

Total Area (Bus and buffer area dependent on N)		
The Interconnections	Static Permutation (without buffers)	Dynamic Permutation (with buffers)
SPB	$O(Nlog^2N)$	$O(N^{3/2}\log N)$
Mesh/Torus	$O(NlogN)$	$O(N^{3/2}\log N)$

Time for A Single Permutation		
The Interconnections	Static Permutation (without buffers)	Dynamic Permutation (with buffers)
SPB	$\sqrt{N}+1$	$2\sqrt{N}$
Mesh	$7\sqrt{N}-8$	$4(\sqrt{N}-1)$
Torus	$7\sqrt{N}-8$	$2(\sqrt{N}-1)$

Time for Multiple Permutations		
The Interconnections	Static Permutation (without buffers)	Dynamic Permutation (with buffers)
SPB	$\sqrt{N}$ [1]†	$\sqrt{N}$ [$\sqrt{N}$]
Mesh	$7\sqrt{N}-8$ [0]	$2(\sqrt{N}-1)$ [$2(\sqrt{N}-1)$]
Torus	$7\sqrt{N}-8$ [0]	$\sqrt{N}-1$ [$\sqrt{N}-1$]

† Note: The number in [ ] indicates the pipe-filling time.

AT^2 (A: Communication Area)		
The Interconnections	Static Permutation (without buffers)	Dynamic Permutation (with buffers)
Lower bound	$O(N^2)$	$O(N^2)$
SPB	$O(N^2\log^2 N)$	$O(N^2\log^2 N)$
Mesh/Torus	$O(N^2)$	$O(N^2\log^2 N)$

AT^2 (A: Total Area)		
The Interconnections	Static Permutation (without buffers)	Dynamic Permutation (with buffers)
Lower bound	$O(N^2)$	$O(N^2)$
SPB	$O(N^2\log^2 N)$	$O(N^{5/2}\log N)$
Mesh/Torus	$O(N^2\log N)$	$O(N^{5/2}\log N)$

Minimum I/O Bandwidth		
The Interconnections	Static Permutation (without buffers)	Dynamic Permutation (with buffers)
SPB	$\sqrt{N}$	$\sqrt{N}$
Mesh	N	$\sqrt{N}$
Torus	N	N

8. Conclusions

We presented two schedules on a square grid of superposed parallel buses; one for static and the other for dynamic permutations. They are area-efficient and yet almost optimal in AT^2 measure of VLSI complexity. Moreover, they take a shorter cycle time than previously reported permutation schemes using almost linear communication area.

The area, time, and AT^2 comlexity measures for the SPB are compared to those for two other linear area interconnections; the mesh-connected computer and the torus. The SPB compares favorably in almost all respects, including the necessary I/O bandwidth. The most striking improvement afforded by the SPB is in the time for a static permutation. In fact, on the basis of linear area arguments, the SPB is almost optimal.

If the broadcast control and interconnection hardware are entirely implemented using highspeed technologies, then, for example, a very fast permutation network could be realized to interconnect off-the-shelf RISC microprocessors (Patterson[1981], Hennessy[1981], Radin[1983]). The two schedules described in this paper are particularly suitable for this purpose because of the simple hardware control algorithms and the regular, area-efficient interconnections.

References

Andresen, S. and S. R. Harrison [1972]. "Toward a general class of time-division multiplexed connection networks," *IEEE Trans. on Communications* COM-20:5, pp. 836-846.

Andresen, S. [1977]. "The looping algorithm extended to base 2^t rearrangeable switching networks," *IEEE Trans. on Communications* COM-25:10, pp. 1057-1063.

Arden, B. W. and T. Nakatani [1985]. "Superposed parallel buses: a systolic area-time optimal VLSI interconnection," Technical Report CS-TR-19, Department of Computer Science, Princeton University.

Arden, B. W. and T. Nakatani [1986]. "The optimal uniform schedule of arbitrary static permutations on superposed parallel buses," Technical Report CS-TR-33, Department of Computer Science, Princeton University.

Batcher, K. E. [1968]. "Sorting networks and their applications," *1968 Spring Joint Comput. Conf., AFIPS Conf. Proc.,* 32, Washington, DC, pp. 307-314.

Benes, V. E. [1962]. "On rearrangeable three-stage connection networks," *Bell Sys. Tech. J.,* 41, pp. 1481-1492.

Benes, V. E. [1965]. *Mathematical Theory of Connecting Networks and Telephone Traphic,* Academic Press, New York.

Franklin, M.A. [1981]. "VLSI performance comparison of Banyan and crossbar communication networks," *IEEE Trans. on Computer,* C-30:4, pp. 283-290.

Gottlieb, A. and C. P. Kruskal [1984]. "Complexity results for permuting data and other computations on parallel processors," *J. ACM,* 31:2, pp. 193-209.

Hall, P. [1935]. "On representatives of subsets," *J. London Math. Soc.,* 10, pp.26-30.

Hennessy, J., J. Norman, F. Baskett, and J. Gill [1981]. "MIPS: a VLSI processor architecture," In Kung, H. T., R. Sproull, and G. Steele(eds.), *VLSI Systems and Computations,* Computer Science Press, Rockville, Md.

Johnsson, S. L. [1985]. "Data permutation and basic linera algebra computations on ensemble architectures," Research Report YALEU/DCS/RR-367, Yale University.

Lawrie, D. H. [1975]. "Access and alignment of data in array processor," *IEEE Trans. on Computers,* C-24:12, pp. 1145-1155.

Leiserson, C. E. [1983]. *Area efficient VLSI computations,* MIT Press, Cambridge, Mass.

Lev, G. F., N. Pippenger, and L. Valiant [1981]. "A fast algorithm for routing in permutation networks," *IEEE Trans. on Computers* C-30:2, pp. 93-100.

Marcus, M. J. and H. S. McDonald [1969]. "The queuing crossbar: a hybrid division and space-division network," *Proc. 1969 National Electronics Conf.,* pp. 605-610.

Marcus, M. [1972]. "Space time equivalents in connecting networks," *Conf. Record, 1972 IEEE Int. Conf. Communications* 31, pp. 35/25-31.

Nassimi, D. and S. Sahni [1979]. "Bitonic sort on a mesh-connected parallel computer," *IEEE Trans. on Computers,* C-27:1, pp. 2-7.

Nassimi, D. and S. Sahni [1980]. "An optimal routing algorithm for mesh-connected parallel computers," *J. ACM,* 27:1, pp. 6-29.

Nassimi, D. and S. Sahni [1982]. "Parallel algorithms to set up the Benes permutation network," *IEEE Trans. on Computers* C-31:2, pp. 148-154. Opferman, D. C. and Tsao-Wu [1971]. "On a class of rearrangeable switching networks," *Bell Sys. Tech. J.* 50, pp.1579-1618.

Orcutt, S. E. [1976]. "Implementation of permutation functions in Illiac IV-type computers," *IEEE Teans. on Computers,* C-25:9, pp. 929-936.

Papadimitriou, C. H. and K. Steiglitz [1982]. *Combinatorial Oprimization: Algorithms and Complexity,* Prentice Hall, Englewood Cliffs, New Jersey.

Patterson, D. A. and C. H. Sequin [1981]. "A reduced instruction set VLSI computer," *Proc. of The Eighth Annual Symposium on Computer Architecture,* Minneapolis, Minn.

Pippenger, N. [1975]. "On crossbar switching networks," *IEEE Trans. on Communications,* COM-23:6, pp. 646-659.

Radin, G. [1983]. "The 801 minicomputer," *IBM Journal of Research and Development,* 27:3, pp. 237-246.

Rosenberg, A. L. [1983]. "Three-dimensional VLSI: a case study," *J. ACM,* 30:3, pp. 397-416.

Schwartz, J. T. [1980]. "Ultracomputers," *ACM Trans. on Programming languagees and Systems,* 2:4, pp.484-521.

Stone, H. S. [1971]. "Parallel processing with the perfect shuffle," *IEEE Trans. on Computers,* C-20:2, pp.153-161.

Thompson, C.D., H. T. Kung [1977]. "Sorting on a Mesh-connected Parallel Computer," *Comm. ACM,* 20:4, pp. 263-271.

Thompson, C.D. [1978]. "Generalized connection networks for parallel processor interconnection," *IEEE Trans. on Computers,* C-27:12, pp. 1119-1125.

Thompson, C.D. [1979]. "A complexity theory for VLSI," Ph.D. dissertation, Carnegie-Mellon Univ., Pittsburgh, PA.

Ullman, J. D. [1984]. *Computational Aspects of VLSI,* Computer Science Press, Rockville, Maryland.

Waksman, A. [1968]. "A permutation network," *J. ACM,* 15, pp. 159-163.

Wise, D. S. [1981]. "Compact layouts of Banyan/FFT networks," *Proc. of the CMU Conference on VLSI Systems and Computations,* Pittsburgh, Penn., pp. 186-195.

```
procedure self-pipelined.hardware-control.algorithm;

    for all broadcasting-chips i on every row do id: = i;
    for all broadcasting-chips i on every row do location: = initial, second, or third;
    for all broadcasting-chips i on every row do begin

        if location = initial, then
            input&output-register←bus;
        if location = second, then begin
            if bus[destination.partA] = id, then begin
                input&output-register←bus[destination.partB, data];
            end
        end
        if location = third, then begin
            if bus[destination.partB] = id, then begin
                input&output-register←bus[data];
            end
        end

    end
    for all broadcasting-chips i on every row do begin

        if location = initial, then begin
            if modulo-counter = input&output-register[broadcast-order], then begin
                bus←input&output-register[destination, data];
            end
        end
        if location = second, then begin
            bus←input&output-register[destination.partB, data];
        end
        if location = third, then begin
            bus←input&output-register[data];
        end

    end

end;
```

Algorithm 2.1: The self-pipelined hardware-control algorithm

```
procedure multi-pipelined.hardware-control.algorithm;

  for all broadcasting-chips i on every row do id: = i;
  for all broadcasting-chips i  on every rowdo location: = initial, second, or third;
  for all broadcasting-chips i on every row do begin

      if location = initial, then input&output-buffer←bus;
      if location = second, then begin
          if bus[destination.partA] = id, then begin
              input&output-buffer + + ←bus[destination.partB,  data];
          end
      end
      if location = third, then begin
          if modulo-counter = id, then begin
              input&output-buffer←bus[data];
          end
      end

  end
  for all broadcasting-chips i on every row do begin

      if location = initial, then begin
          if modulo-counter = id, then begin
              bus←input&output-buffer[destination,  data];
          end
      end
      if location = second, then begin
          if modulo-counter = ∃input&output-buffer[destination.partB], then begin
          bus←∃input&output-buffer[data] – – ;
      end
      if location = third, then begin
          bus←input&output-buffer[data];
      end

   end

end;
```

Note: + + means a *first-in/first-out* addition of a buffer entry
– – means a deletion of a buffer entry matched by the associative matching
∃ means existence of an entry matched by the associative matching

Algorithm 3.1: The multi-pipelined hardware-control algorithm

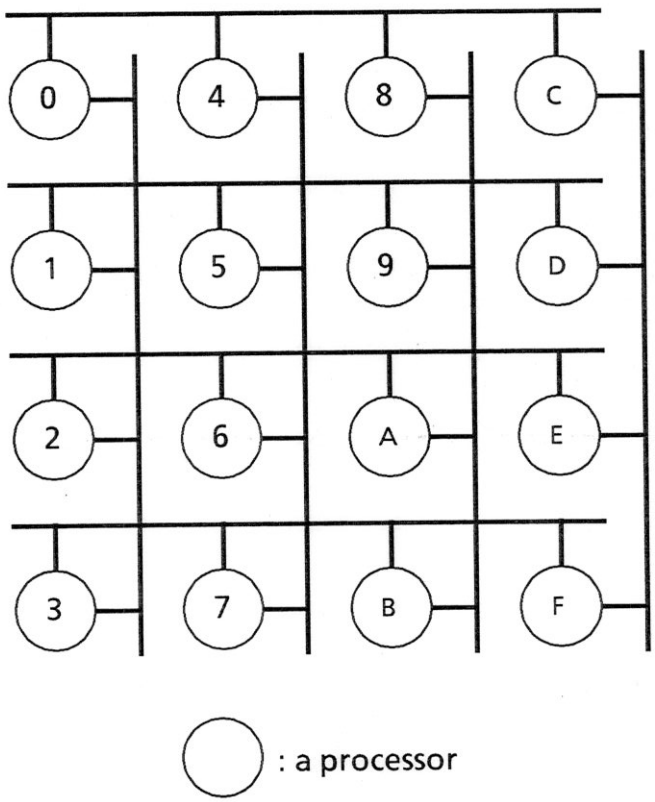


Figure 1.1: A 4x4 square grid of superposed parallel buses

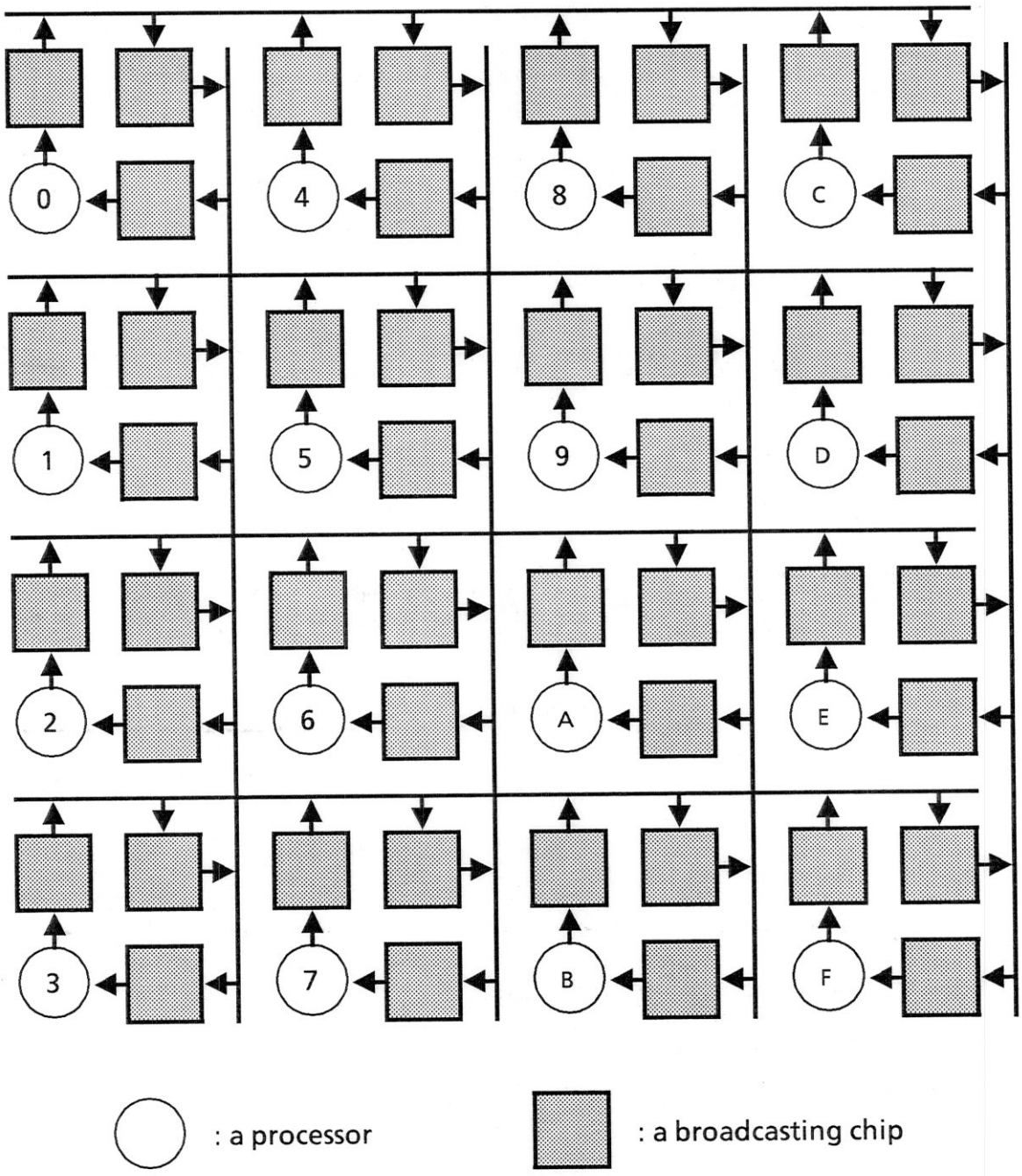


Figure 1.2: A 4x4 pipelined grid interconnection

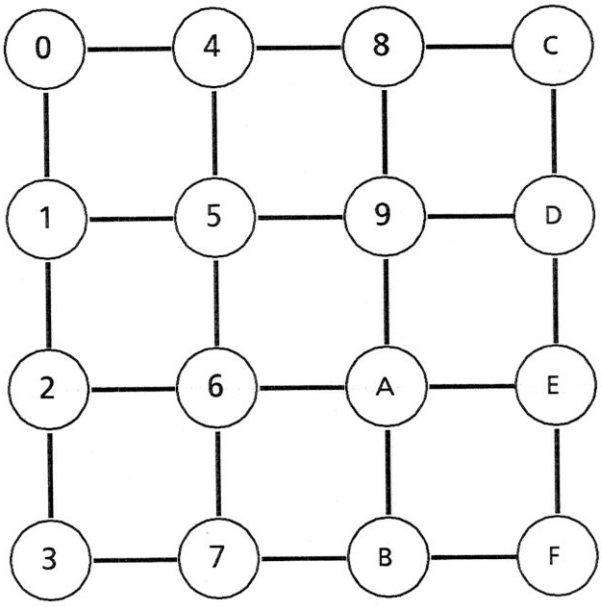


Figure 1.3: A 4x4 mesh interconnection

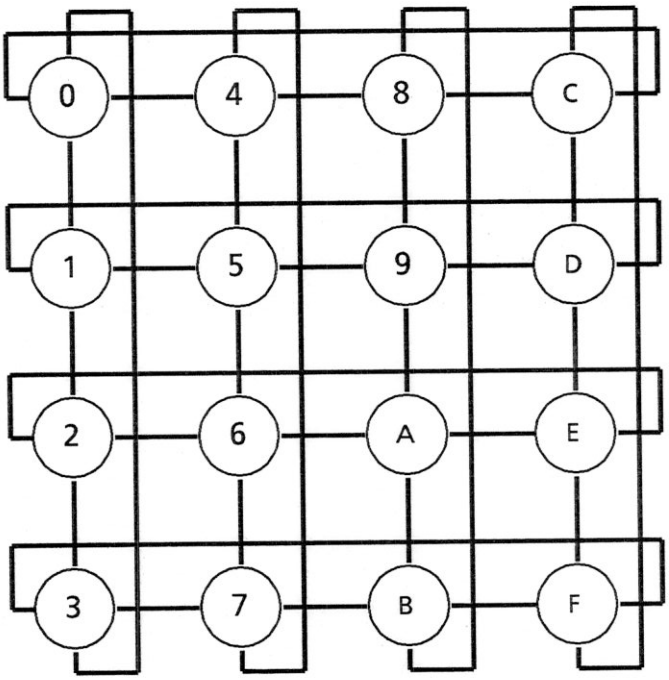
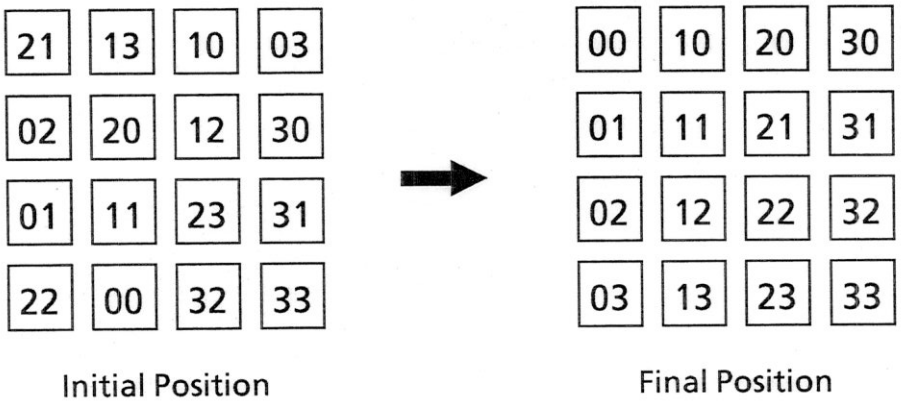


Figure 1.4: A 4x4 torus interconnection



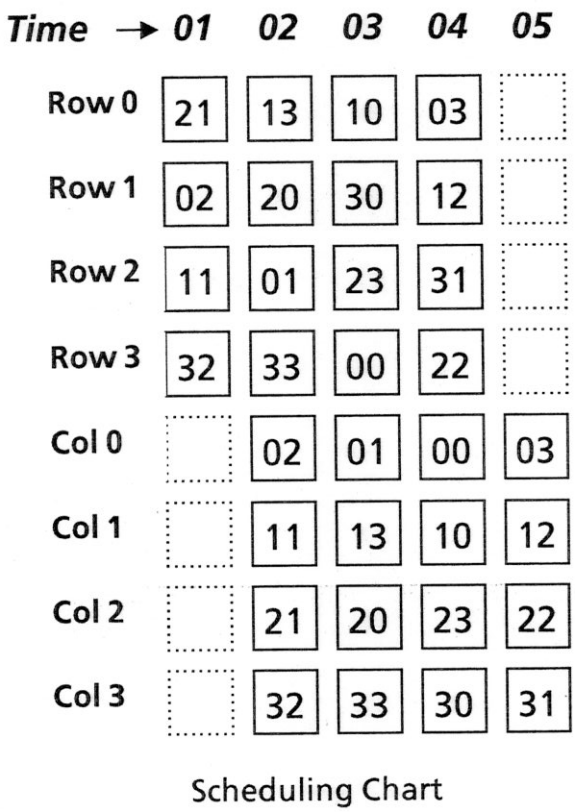


Figure 2.1: A 4x4 self-pipelined schedule

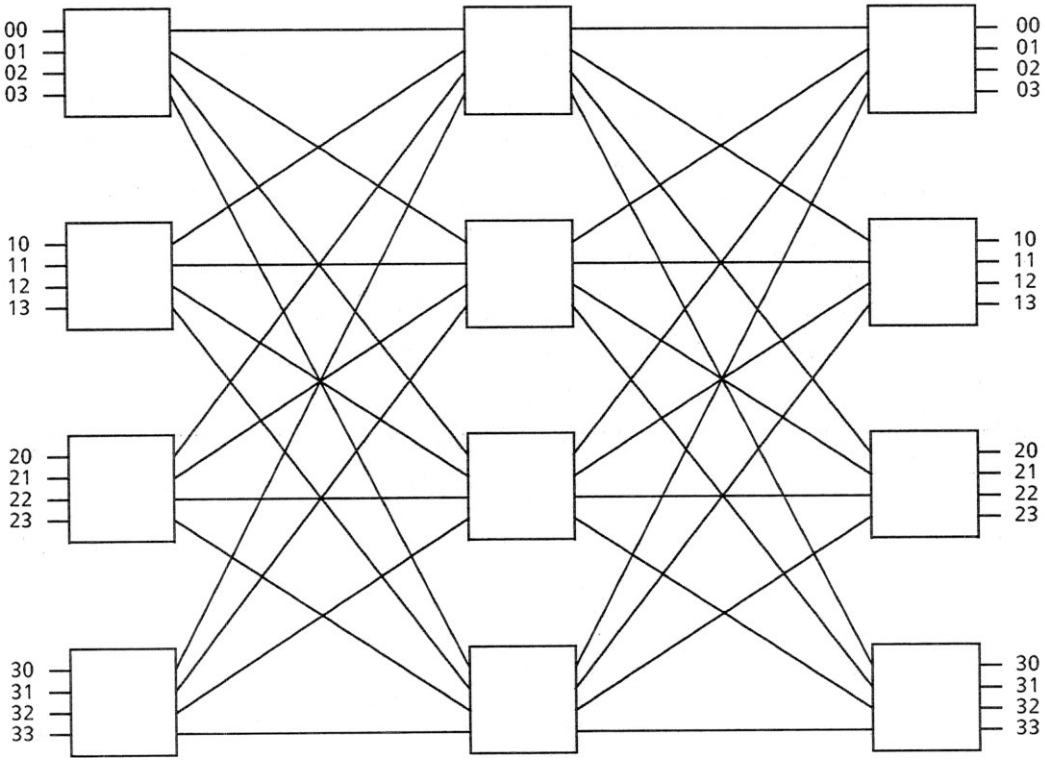


Figure 2.2: A 4^2 Benes-Clos network

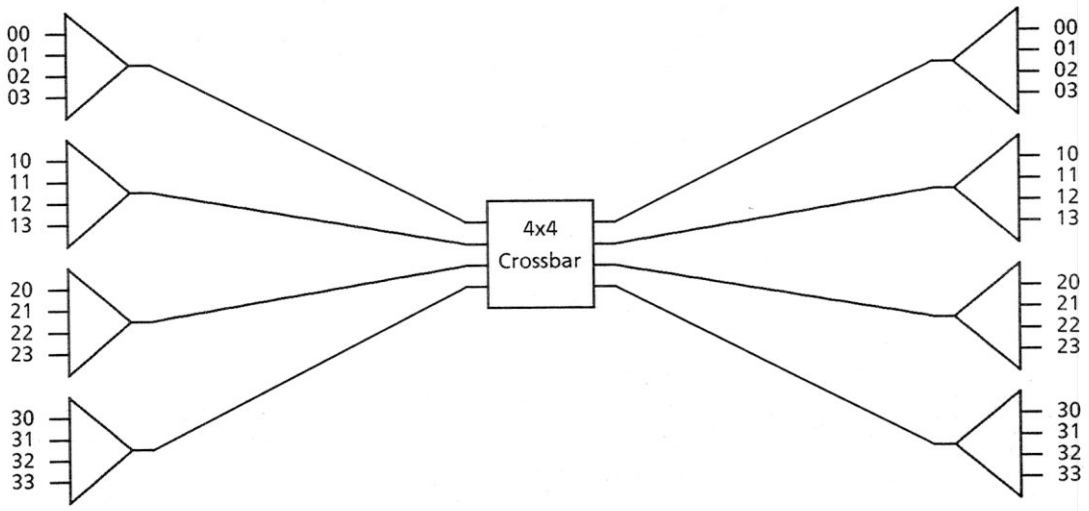


Figure 2.3: A 4^2 time-multiplexed crossbar

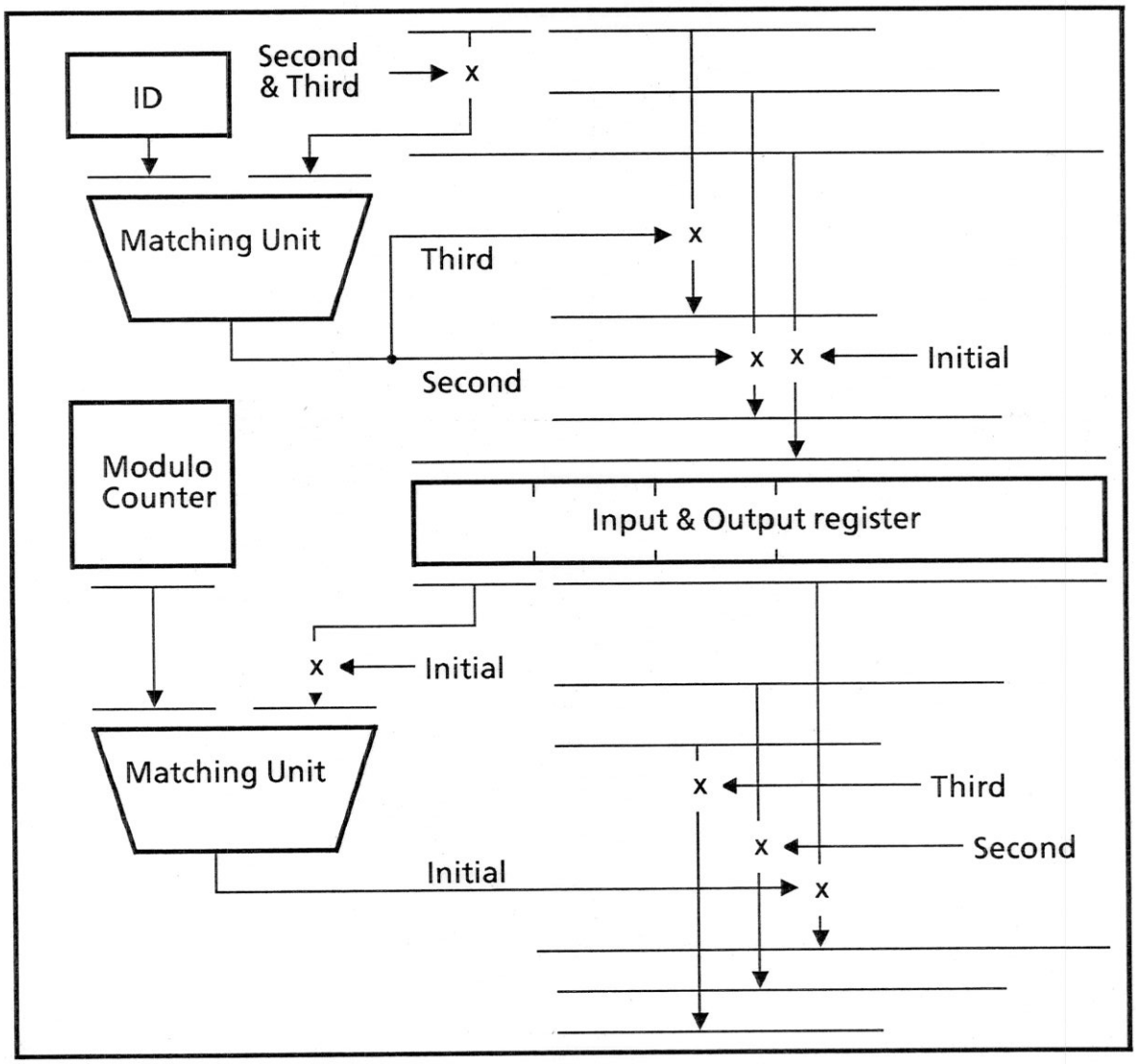


Figure 2.4: A broadcasting chip for the self-pipelined schedule

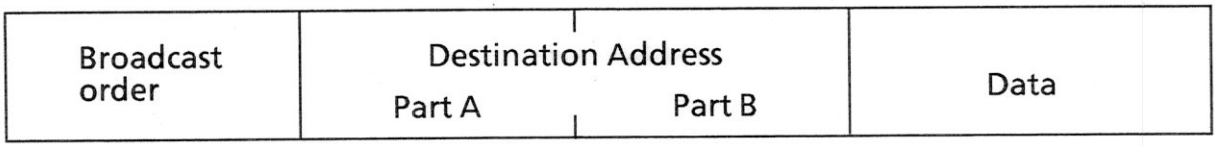
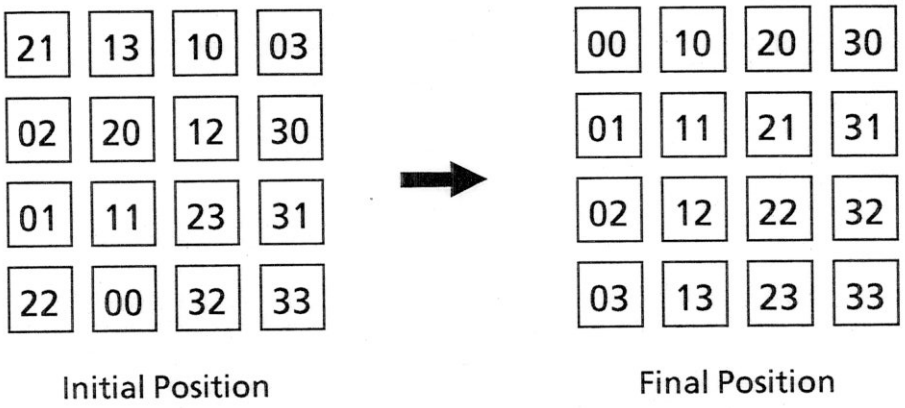


Figure 2.5: A packet format for the self-pipelined schedule



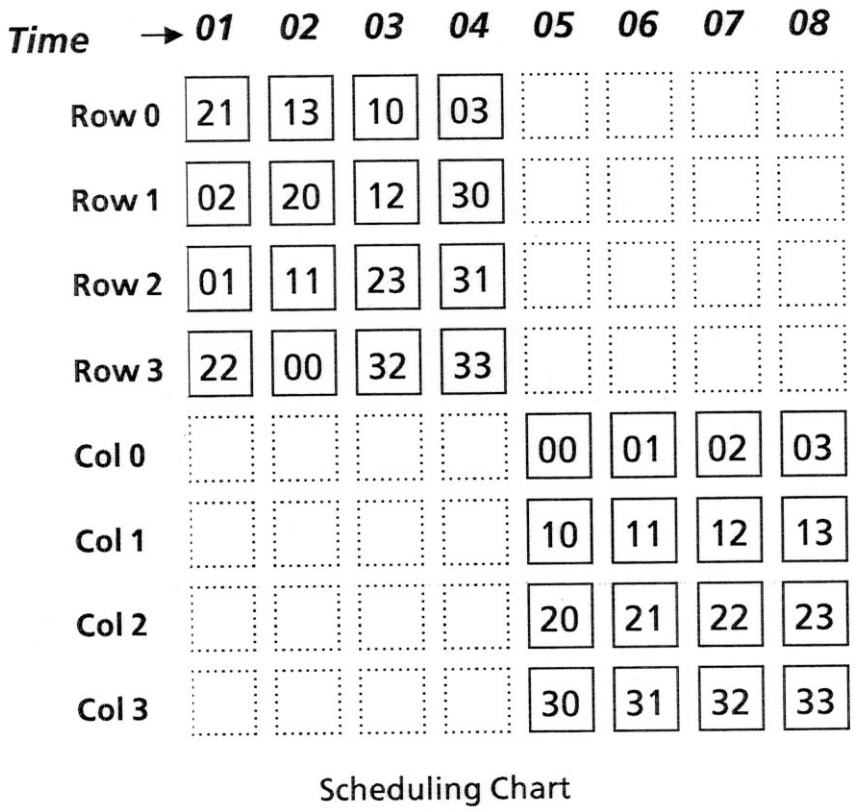


Figure 3.1: A 4x4 multi-pipelined schedule

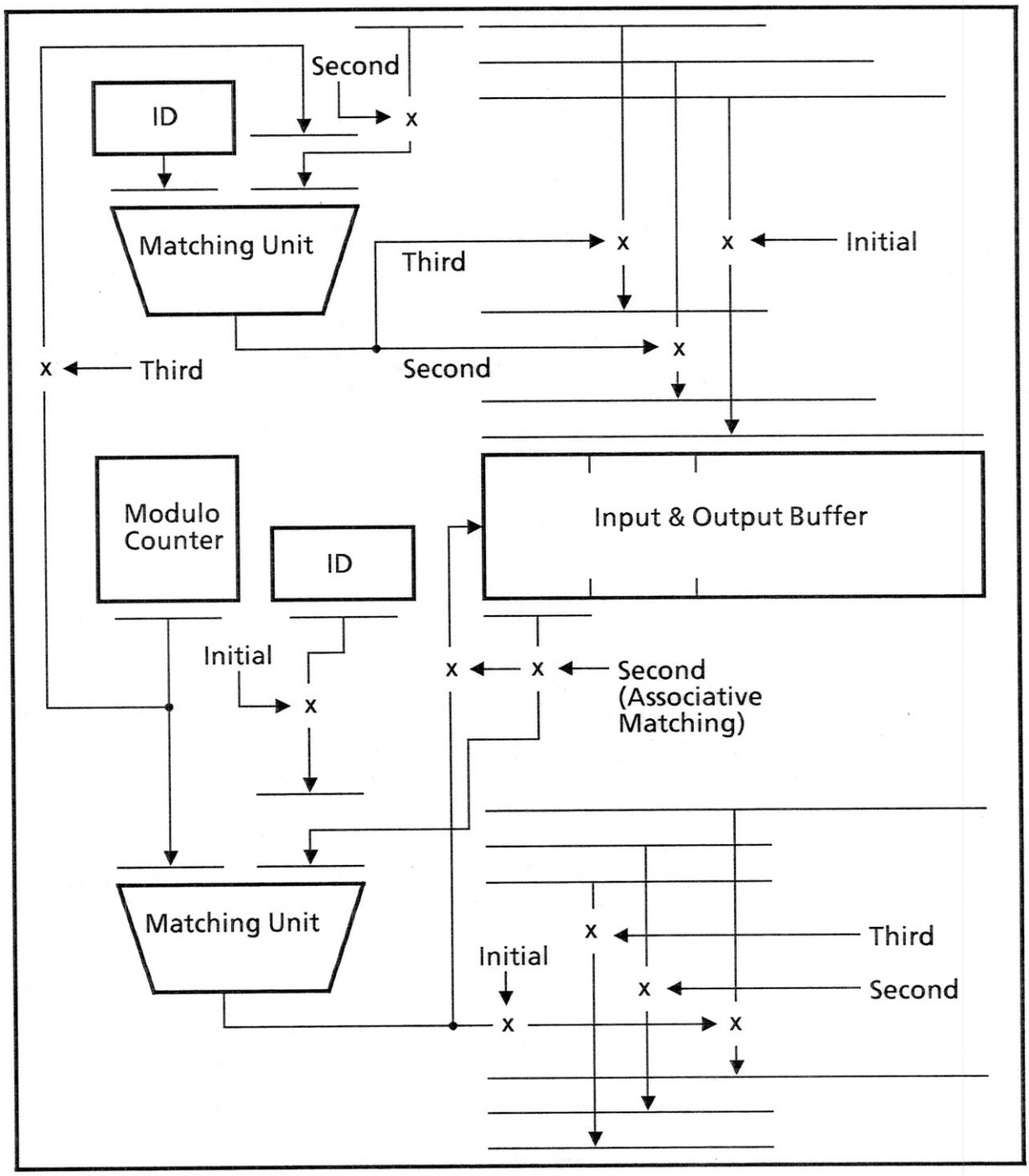


Figure 3.2: A broadcasting chip for the multi-pipelined schedule

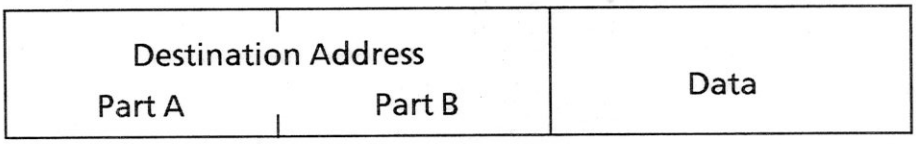


Figure 3.3: A packet format for the multi-pipelined schedule

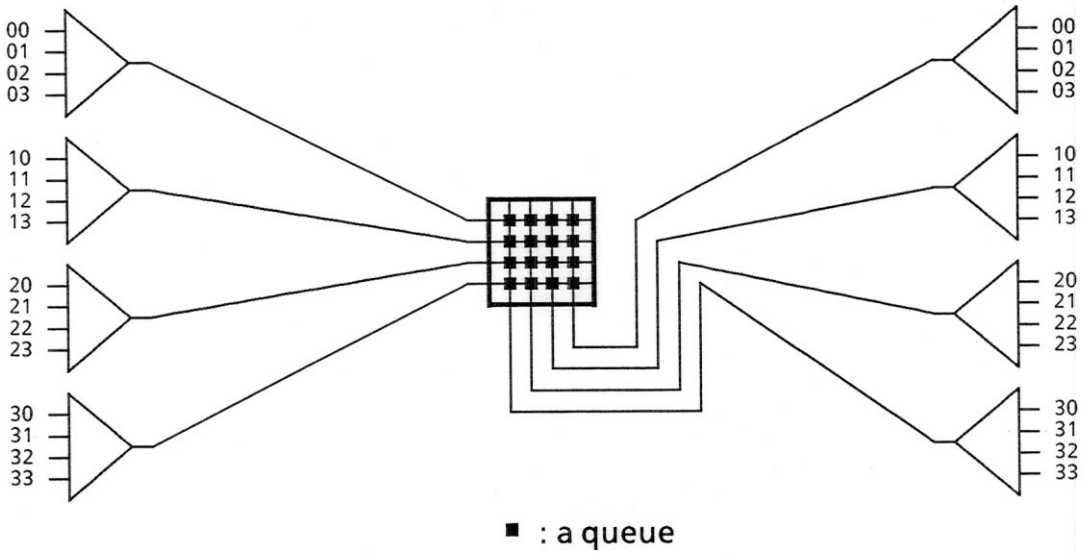


Figure 3.4: A 4^2 queueing crossbar

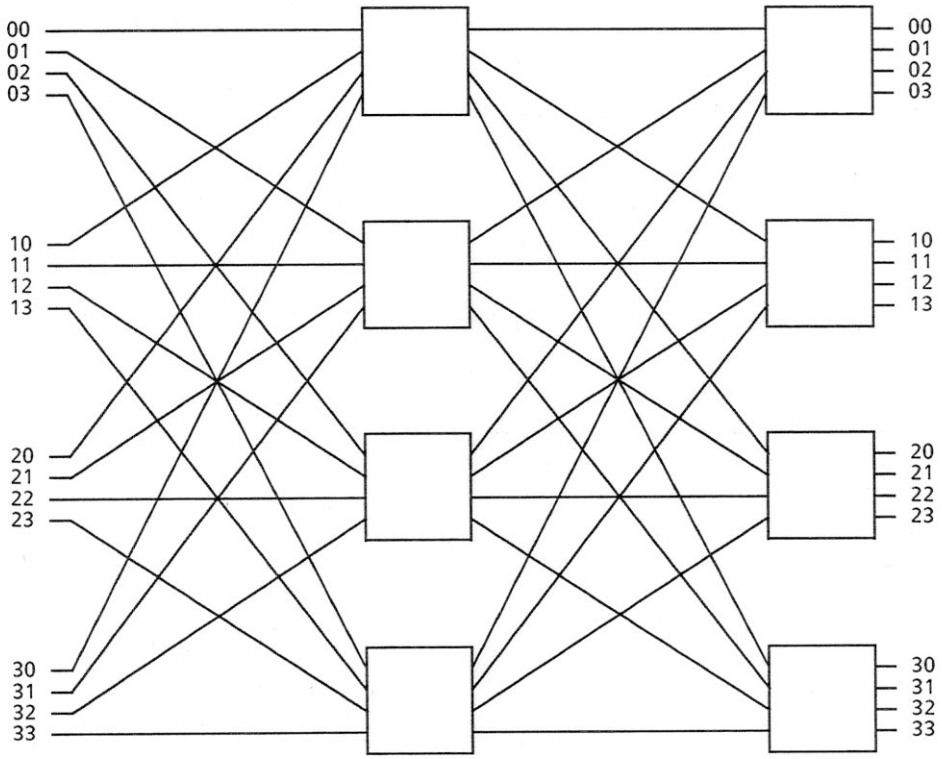


Figure 3.5: A 4^2 Ω-network

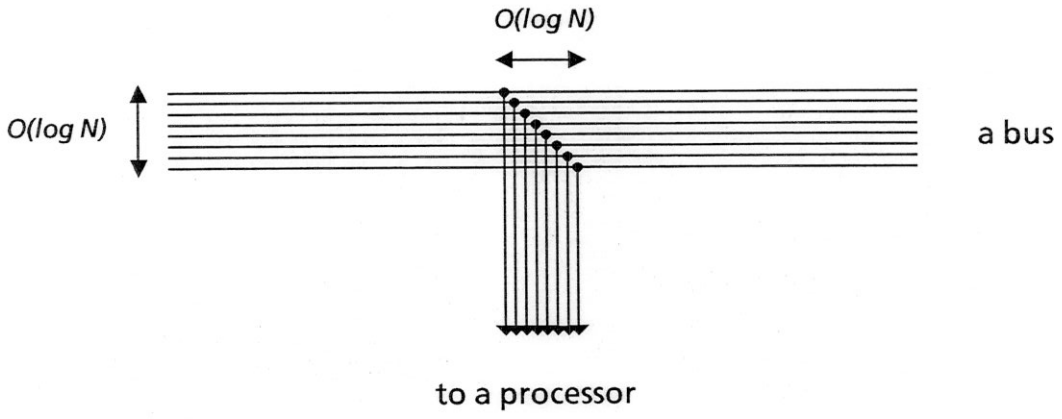


Figure 4.1: *O(log^2 N)* area of a connection between a bus and a processor