OPTIMAL PERMUTATIONS ON SUPERPOSED PARALLEL BUSES

Bruce W. Arden

Toshio Nakatani

CS-TR-060-86

November 1986

Optimal Permutations on Superposed Parallel Buses

(Extended Abstract)

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

The paper is concerned with the optimal schedule for the permutation of n^2 data items on an $n \times n$ square grid formed by the orthogonal superposition of $2n$ time multiplexed buses. The upper bound is proved by showing the existence of the $n+1$ cycle, *uniform schedule* (that is, all two-step transfers consistently row first or alternately column first) for an arbitrary permutation. The lower bound is proved by showing the non-existence of n-cycle, non-uniform schedules for some non-degenerate permutations. We also show a simple way to perform an arbitrary permutation dynamically with n buffers at every bus intersection. We further show that, with specific example of permutations, the potential increase in parallelism by dynamically partitioning the $2n$ buses does not lead to a further reduction in time.

November 14, 1986

Optimal Permutations on Superposed Parallel Buses

(Extended Abstract)

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

Consider the permutation of $N=n^2$ data items using two sets of n parallel, time multiplexed buses. If one set of buses is considered to be horizontal and the other vertical, they can be considered to be *superposed* with communication between the horizontal and vertical buses at the bus intersections (Figure 1.1). The time for the transmission of one data item on a bus is considered to be a single *cycle*. With n buses in parallel, there can be at most n simultaneous transmissions on one set of parallel buses. The overlapped, or *pipelined,* use of the two bus planes leads to an overall maximum transmissions of $2n$ data items in a single cycle.

Using the conventional, Cartesian notation for the n^2 bus intersections, the destination address $x_k y_l$ of a data item is initially present at each of the square array of address $x_i y_j$ $(0 \leq i,j,k,l \leq n-1)$. For permutation, all n^2 distinct addresses occur. A permutation is accomplished by transferring a data item from $x_i y_j$ to $x_k y_l$ for all i,j, where $x_k y_l$ was the destination address of the data item initially in the $x_i y_j$ position. In the general case, there are two bus broadcast steps to accomplish this transfer, i.e., horizontal then vertical or vertical then horizontal (Figure 1.2). Bus pairs intersecting at $x_i y_l$ or $x_k y_j$ must be used. Clearly, there are *degenerate* cases where source and destination are in the same row or column, $x_i = x_k$ or $y_j = y_l$, and only one bus broadcast is required. Also none is required when $x_i = x_k$ and $y_j = y_l$.

For permutations without degenerate cases, that is, every transfer requires two steps, a total of $2n^2$ bus broadcast steps are necessary. Exploiting the bus parallelism and overlap, it is clear that the best schedule or shortest time to complete a single permutation would therefore be n cycles. However, to complete a nondegenerate, single permutation within n cycles, n horizontal bus broadcasts and n vertical bus broadcasts must be accomplished every cycle including the first one. For the overlapped operations on the first cycle, *mixed* schedules, or mixtures of both the horizontal-then-vertical and the vertical-then-horizontal transfers, are necessary. It is interesting to know if there exists a minimum, mixed schedule for arbitrary permutations.

On the other hand, as we prove in section 3, Hall's theorem on "Distinct Representatives" (Hall[1935]) guarantees the existence of *a uniform* schedule†, that is, an unmixed schedule when the transfers are uniformly horizontal-then-vertical or vertical-then-horizontal. Because of the inherent serial aspect of the two bus broadcasts, the uniform schedule takes $n+1$ cycles. That is, there can be no bus overlap on the first cycle. The uniform schedule can be regarded *optimal* as well, because sequential permutations could achieve an overlap of the additional cycle and thus complete one of a sequence of permutations every n cycles. We show in section 4 that, with specific example of permutations, no mixed schedule of n cycles exists for single permutations. Although a mixed, n cycle schedule exists for some permutations, one does not exist for all. Hence, the $n+1$ cycle, uniform schedule is optimal for single permutations on an $n \times n$ square grid of *superposed parallel buses.*

However, it is a time-consuming operation to find the broadcast order for the uniform schedules. The uniform schedule is considered to be equivalent to a time-division-multiplexing (Andresen and Harrison[1972] and Marcus[1972]) of a three-stage n^2 Benes-Clos rearrangeable network (Clos[1952] and Benes[1962]) and also to be equivalent to a time-multiplexed crossbar (Marcus[1972]). The best known sequential algorithm

† This is called *a self-pipelined* schedule in the previous paper (Arden and Nakatani[1986a]).

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. On the other hand, if we allow n buffers at every bus intersection, there is a simple way to perform an arbitrary permutation dynamically in $2n$ cycles on a $n \times n$ grid of superposed parallel buses. This simple schedule for dynamic permutations is considered to be equivalent to a time-division-multiplexing of a n^2 Ω-network (Lawrie[1975]) and also to a $n \times n$ queuing crossbar (Marcus and McDonald[1969]).

After these observations, the question naturally arises whether the performance of permutations can be improved by partitionable buses. By partitionable buses, we mean that the bus can be separated into arbitrary, contiguous segments. if the system can be partitioned into many subsystems that can work independently in parallel, then the time to perform permutations may be improved. This is the case for the divide-and-conquer algorithms (Arden and Ginosar [1982] and Arden and Nakatani[1986b]). However, we prove that partitionable buses cannot improve the performance of permutations. That is, it is shown that some permutations, such as the bit-reversal permutation, require the same number of steps whether the buses involved are partitioned or non-partitioned.

The paper is organized as follows: In section 2, some notation and definitions are given for later use. In section 3, the upper bound for schedules (that is, the existence of the uniform schedule) of an arbitrary permutation on a $n \times n$ square grid of superposed parallel buses is proved. In section 4, the lower bound for the schedules (that is, the non-existence of mixed, n cycle schedules) of some permutations is proved. In section 5, it is proved that if we assume n buffers at every bus intersection then there is a simple way to perform an arbitrary permutation dynamically in $2n$ cycles. In section 6, it is proved by example that partitionable buses do not improve the performance of some permutations on a square grid of superposed parallel buses.

2. Mathematical Notations and Definitions

In this section, notation and definitions are described for later use.

Definition 1: A $n \times n$ square grid of *superposed parallel buses,* denoted as $SPB(n,n)$, is a system with n^2 processors and two sets of n buses; one, called *the row buses* R_0 through R_{n-1}, can be viewed as horizontal and the other, called *the column buses* C_0 through C_{n-1}, as vertical. A processor is located at each cross point and each processor has two ports; one for a horizontal bus and the other for a vertical bus. *The address* of each processor is denoted by (x,y) or xy, where x represents x-coordinate, called *the column address,* and y represents y-coordinate, called *the row address.* There are n processors on each bus. *The row bus* R_i supports n processors $(i, 0)$ through $(i,n-1)$ for $0 \leq i \leq n-1$. *The column bus* C_j supports n processors $(0,j)$ through $(n-1,j)$ for $0 \leq j \leq n-1$. It is assumed in this paper that each processor at the address (x,y) has a data item with its mailing address, called *destination address* $D(x,y)$, and that a mapping from (x,y) to $D(x,y)$ forms a permutation from $SPB(n,n)$ to $SPB(n,n)$.

Definition 2: *A column selection* is a set of n processors at the addresses $(x_0,0)$ through $(x_{n-1},n-1)$, where x_i is an integer which is not necessarily unique $(0 \leq x_i \leq n-1)$. Similarly, *a row selection* is a set of n processors at the addresses $(0,y_0)$ through $(n-1,y_{n-1})$, where y_j need not be unique $(0 \leq y_j \leq n-1)$. *A column-row selection* is a set of n processors that forms both a column and row selections at the same time.

Definition 3: A column selection is called *compatible* when each processor i in a column selection has a data item with the destination address (x_i,y_i) and a set of n processors at the addresses $\{(x_i,y_i)\}$ for $0 \leq i \leq n-1$ forms a row selection. Similarly, a row selection is called *compatible* when each processor i in a row selection has a data item with the destination address (x_i,y_i) and a set of n processors at the addresses $\{(x_i,y_i)\}$ for $0 \leq i \leq n-1$ forms a column selection. A column-row selection is called *perfectly compatible* when it is compatible both as a column selection and as a row selection. Moreover, k sets of column-row selections are called *perfectly k-compatible,* (even if each column-row selection is not necessarily perfectly compatible but) if the processors in k sets of column-row selections have data items with exactly k distinct destination row addresses for each of n destination

column addresses or with exactly k distinct destination column addresses for each of n destination row addresses.

Definition 4: *The uniform* schedule is n sequences of the horizontal-then-vertical transfers (or equivalently n sequences of the vertical-then-horizontal transfers) of data items chosen by n series of compatible column selections (or equivalently n series of compatible row selections). This is always possible according to Hall's theorem on "Distinct Representatives" (Hall[1935]) as we shall show in the next section. Since each selection is compatible, all the selected data items can be broadcast on the row (the column) buses right after they are broadcast on the column (the row) buses without further delays. Therefore, by the uniform schedule, a single permutation takes $n+1$ cycles and each of sequential or multiple permutations takes n cycles in pipeline fashion. On the other hand, *a mixed* schedule is n sequences including both the horizontal-then-vertical and the vertical-then-horizontal transfers of data items chosen by a series of n compatible selections. The minimum schedule of n cycles is possible for some specific permutations (for example, the bit-reversal permutation) using the mixed strategy, but in general $n+1$ is the best that can be achieved. Hence, such schedules are optimal.

Definition 5: A permutation is called *degenerate,* if some pairs of source and destination are in the same row or column. Otherwise, a permutation is called *nondegenerate.*

Example 1: The bit-reversal permutation is nondegenerate and has an n-cycle mixed schedule (Figure 2.1).

Example 2: The permutation $D_1(x,y)$ is a bijection from the $SPB(n,n)$ to $SPB(n,n)$ by the following rule:

$$D_1(x,y)=([x-2]_n,[y+1]_n)\dagger \text{ for } y\neq n-1$$

$$D_1(x,y)=([x-1]_n,[y+1]_n) \text{ for } y=n-1$$

This permutation $D_1(x,y)$ is nondegenerate and has no n-cycle schedule, as will be shown in the following section (see Figure 2.2 and Figure 2.3 for example of the uniform schedule and the mixed schedule of $D_1(x,y)$ respectively).

† $[m]_n=m \pmod n$

3. The Upper Bound

In this section, we prove the existence of the $n+1$ cycle, uniform schedule for an arbitrary permutation on the $SPB(n,n)$.

Theorem 3.1: There always exists the uniform schedule for an arbitrary permutation of n^2 data items in $n+1$ cycles on the $SPB(n,n)$.

Proof: Consider $n \times n$ array of column destinations. There are exactly n occurrences 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 compatible column selection. That is, 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 compatible column selection until exhausted. Thus, all the data items on the rows can be broadcast in n cycles, and all the data items reach their destinations in $n+1$ cycles by interleaving operations of row and column broadcasts.□

4. The Lower Bound

In this section, the lower bound is proved by showing an existence of a single permutation that requires at least $n+1$ cycles and n cycles, in pipeline fashion, for each of a sequence of permutations.

Theorem 4.1: The uniform schedule is optimal for permutations on the $SPB(n,n)$. That is, there exists a permutation that takes $n+1$ cycles for a single permutation and n cycles in pipeline fashion for each of a sequence of the permutations.

Proof: From Lemma 4.4, the permutation $D_1(x,y)$ has no n-cycle mixed schedule. The uniform $(n+1)$-cycle schedule exists for all permutations and is therefore optimal.□

Lemma 4.1: The permutation $D_1(x,y)$ is non-degenerate, that is it takes at least n cycles on the $SPB(n,n)$. This can be achieved only by compatible selections.

Proof: Every destination address of $D_1(x,y)$ requires both column and row broadcasts. One cycle of $SPB(n,n)$ can transmit at most n data items on the column buses and at most n data items on the row buses. Therefore, at least $n^2 \times 2/(2n) = n$ cycles are required and this can be achieved only by compatible selections.□

Lemma 4.2: Any compatible column selection of $D_1(x,y)$ is also a row selection but not compatible row selection. Therefore, any compatible column selection of $D_1(x,y)$ is a column-row selection but not perfectly compatible.

Proof: For any compatible column selection of $D_1(x,y)$, let $(0,y_0)$ through $(n-1,y_{n-1})$ be the destination addresses of the selected processors' data items. Then, $\{y_j\}$ for $0 \leq j \leq n-1$ must form a permutation of integers $\{0,1,...,n-1\}$. Since (x,y_j) appears only on the row y_j+1 for any x, a set of n processors, which have the data items with the destination addresses $\{(j,y_j)\}$ for $0 \leq j \leq n-1$, forms a row selection. However, in any compatible column selection of $D_1(x,y)$, exactly one processor must have a data item with the destination address $(x_{n-1},n-1)$ and the other processors must have data items with the destination addresses (x_i,i) for $0 \leq i \leq n-2$. Moreover, one processor must be selected from the column $[x_{n-1}-1]_n$ excluding the location $([x_{n-1}-1]_n,0)$, on which any of the processors has a data item with the destination address (x_{n-1},h) for a h $(0 \leq h \leq n-2)$. That is, two processors, which have the data items with the destination addresses $(x_{n-1},n-1)$ and (x_{n-1},h), must be selected as a part of any compatible column selection. Therefore, this is not a compatible row selection, that is, not a perfectly compatible column-row selection.□

Lemma 4.3: In any compatible column selection of $D_1(x,y)$, there are exactly two processors whose data items have the same destination column address and the rest of the processors have data items with distinct destination column addresses from each other and also from the two processors.

Proof: From the proof of Lemma 4.2, any compatible column selection contains exactly two processors at the addresses $([x_{n-1}-1]_n,0)$ and $([x_{n-1}-2]_n,h+1)$, which have the data items with the destination addresses $(x_{n-1},n-1)$ and (x_{n-1},h) respectively. The rest of the processors are located at the addresses $\{(j,i)\}$, for any i $(i \neq 0,h+1)$ and any j $(j \neq [x_{n-1}-1]_n,[x_{n-1}-2]_n)$, and have the data items with the destination

addresses $\{([j+2]_n,[i-1]_n)\}$. Therefore, the rest of the processors have the data items with distinct destination column addresses $\{[x_{n-1}+t]_n\}$ for $2 \le t \le n-1$ and only two processors have the data items with the same destination column address x_{n-1}.□

Lemma 4.4: The permutation $D_1(x,y)$ has no n-cycle mixed schedule.

Proof: From Lemma 4.3, in any compatible column selection of $D_1(x,y)$, the selected processors' destination column addresses are distinct except for the two. More precisely, the processors in a compatible column selection contains the destination column addresses of two x_{n-1} and no $[x_{n-1}+1]_m$ and each of the other integers. Because of this rule, for any k $(1 \le k \le n)$, the processors in any k sets of compatible column selections cannot have the same number of aliases for each destination column address except for the case of $k=n$. That is, any k sets of compatible column selections cannot be perfectly k-compatible column-row selections except for the case of $k=n$, when all the compatible selections are the column selections. Therefore, either n compatible column selections or n compatible row selections are the only compatible selections for the whole permutation of $D_1(x,y)$. From Lemma 4.1, the permutation $D_1(x,y)$ takes at least n cycles and this is possible only with compatible selections. The only way with compatible selections is either by n compatible column selections or by n compatible row selections. That is, the permutation $D_1(x,y)$ has no n-cycle mixed schedule.□

5. A Simple Method for Dynamic Permutation

If we allow n buffers at every bus intersection, we can perform an arbitrary permutation dynamically in $2n$ cycles on the $SPB(n,n)$. In this schedule†, the broadcast order is fixed and independent of permutations (see Figure 5.1 for example).

Theorem 5.1: There exists a schedule for an arbitrary, dynamic permutation of n^2 data items in $2n$ cycles on the $SPB(n,n)$ and n buffers are necessary and sufficient at every bus intersection.

† This is called a *multi-pipelined* schedule in the previous paper (Arden and Nakatani[1986a]).

Proof: Since there are n data items on each row, all the data items can reach their destination columns by n row broadcasts (that is, n cycles). There are exactly n column broadcasts (that is, n cycles) can convey all the data items 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 the $SPB(n,n)$. In the worst case of permutations, as many as n data items must be received by the processor at a intersection 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 $n-k$ data items remain in the processor for transmissions on columns and at most k data items have been received from row transmissions. Therefore, n buffers are sufficient.□

6. Permutation with Partitionable Buses

In this section, we assume partitionable buses for the $SPB(n,n)$ so that a $n \times n$ square grid of processors can be partitioned into arbitrary, rectangular or square segments (see Figure 6.1 for the partitionable $SPB(4,4)$). Thus, partitionable buses induce more parallelism. However, the performance of permutations cannot be improved for some permutations (see Figure 6.2 for the bit-reversal permutation).

Theorem 6.1: There exist some permutations that take the same number of cycles on the partitionable $SPB(n,n)$ as on the non-partitionable one.

Proof: We assume direct transfers of data items, that is, no intermediate storage of data items, but it is easily shown that no improvement can be made even with intermediate storage. Consider the bit-reversal permutation. First, we partition a square grid into two halves; the upper half and the lower half. Since every data item must cross the boundary, we cannot partition the buses until every processor finishes sending and receiving a data item. When the buses are ready to be partitioned, no processor has a data item to send. Second, we partition a square grid into four quarters; the upper right, the upper left, the lower left, and the lower right. Every processor in the upper left quarter must send and receive a data item to and from the lower right quarter. Similarly, every processor in the upper right quarter must send and receive a data item to and from the lower left

quarter. Therefore, we cannot partition the buses until all the processors finish sending and receiving a data item. In general, wherever partitions are made, we cannot partition the buses until all the processors in one partition finish sending and receiving a data item. When the buses are ready to be partitioned, there is no data items left to send and receive in that partition. Therefore, partitionable buses do not reduce the time required for the permutation.□

7. Concluding Remarks

In this paper, we have proved the upper and lower bounds of permutations on the $SPB(n,n)$. We have also shown the simple schedule of dynamic permutations on the $SPB(n,n)$, although it requires n buffers at every bus intersection. We have further proved that the performance of some permutations, like the bit-reversal permutation, cannot be improved by the partitionable $SPB(n,n)$.

In the previous paper (Arden and Nakatani[1986a]), we studied VLSI optimality of the $SPB(n,n)$ for permutations on the unit delay model and showed that the $SPB(n,n)$ has the best time performance among the linear-area interconnections.

As a closely related subject, we also studied the sorting problem on the $SPB(n,n)$ (Arden and Nakatani[1986b]). We showed that the performance of bitonic sorting can be dramatically improved on the partitionable $SPB(n,n)$ with even better performance than the known adaptation of bitonic sorting on the mesh interconnection (Thompson and Kung[1977] and Nassimi and Sahni[1979]).

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 R. Ginosar [1982]. "MP/C: a multiprocessor/computer architecture," *IEEE Trans. on Computers* C-31:5, pp. 455-473.

Arden, B. W. and T. Nakatani [1986a]. "Permutations on superposed parallel buses," Technical Report CS-24, Department of Computer Science, Princeton University.

Arden, B. W. and T. Nakatani [1986b]. "Bitonic Sorting on superposed parallel buses," Technical Report, Department of Computer Science, Princeton University.

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

Clos, C [1965]. "A study of non-blocking switching networks," *Bell Sys. Tech. J.*, 32, pp. 406-424.

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

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

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 [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.

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

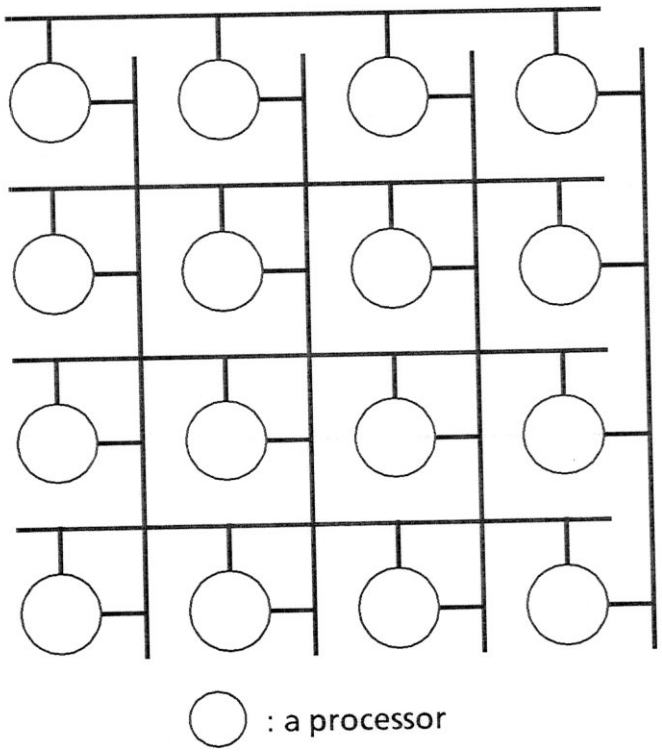


Figure 1.1: A *SPB(4,4)*

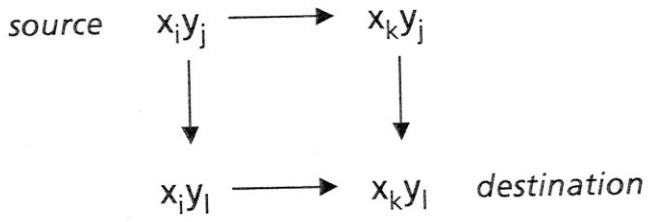


Figure 1.2: A transfer from source to destination

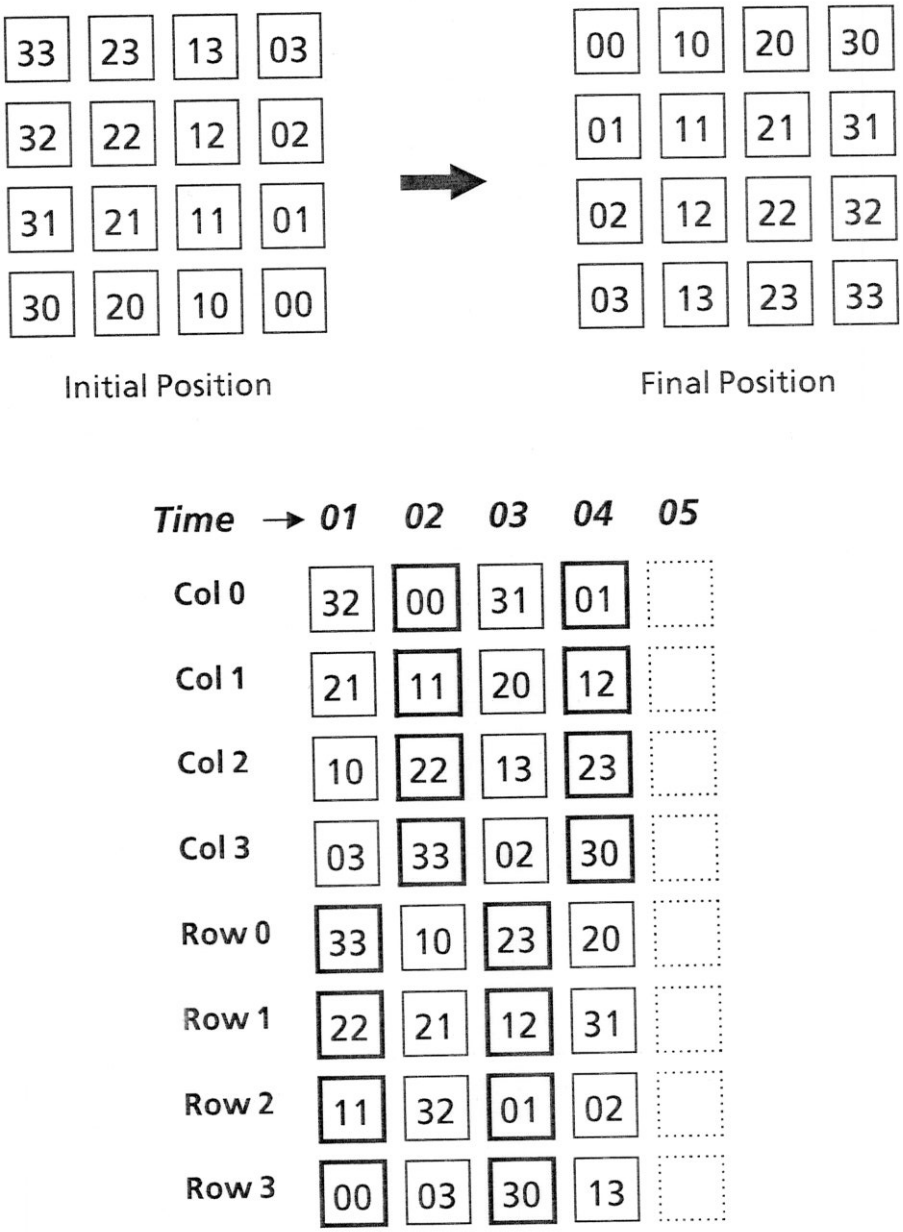
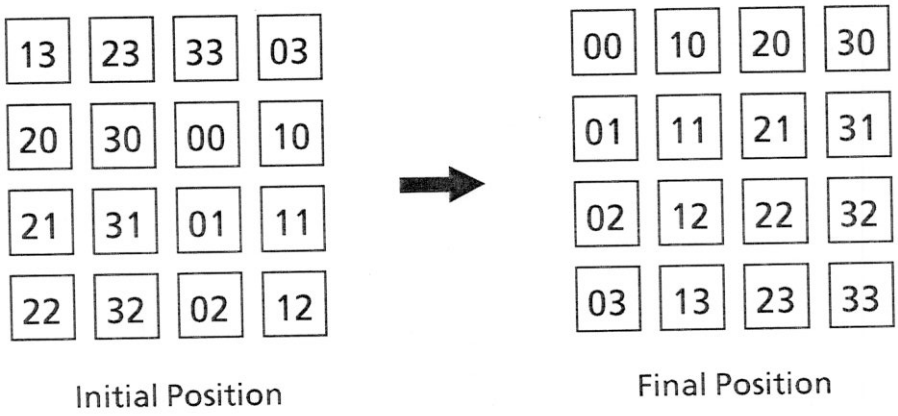


Figure 2.1: The *n*-cycle mixed schedule of the bit-reversal permutation



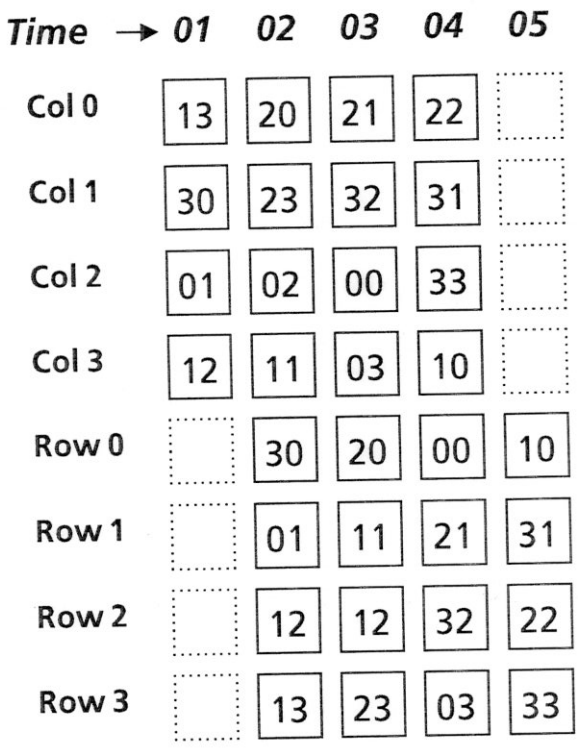
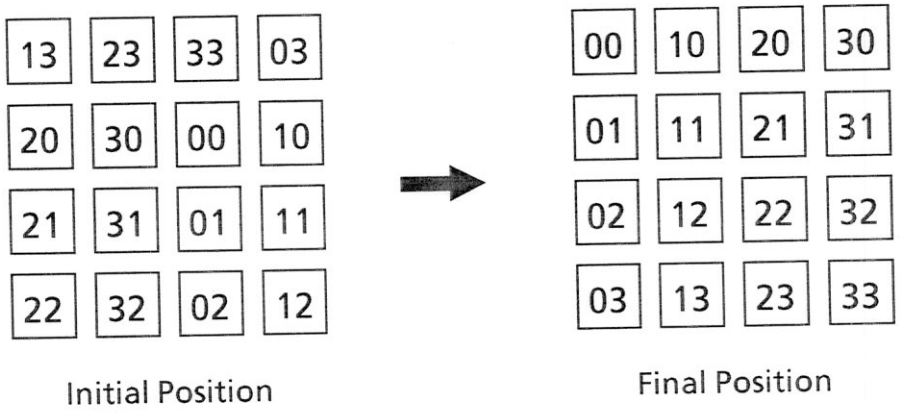


Figure 2.2: The uniform (column-row) schedule of $D_1(4,4)$



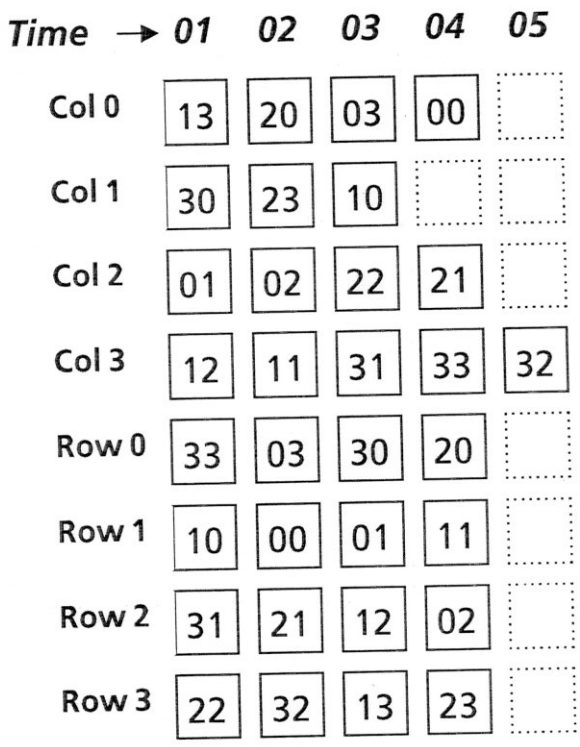


Figure 2.3: The mixed (column-row) schedule of $D_1(4,4)$

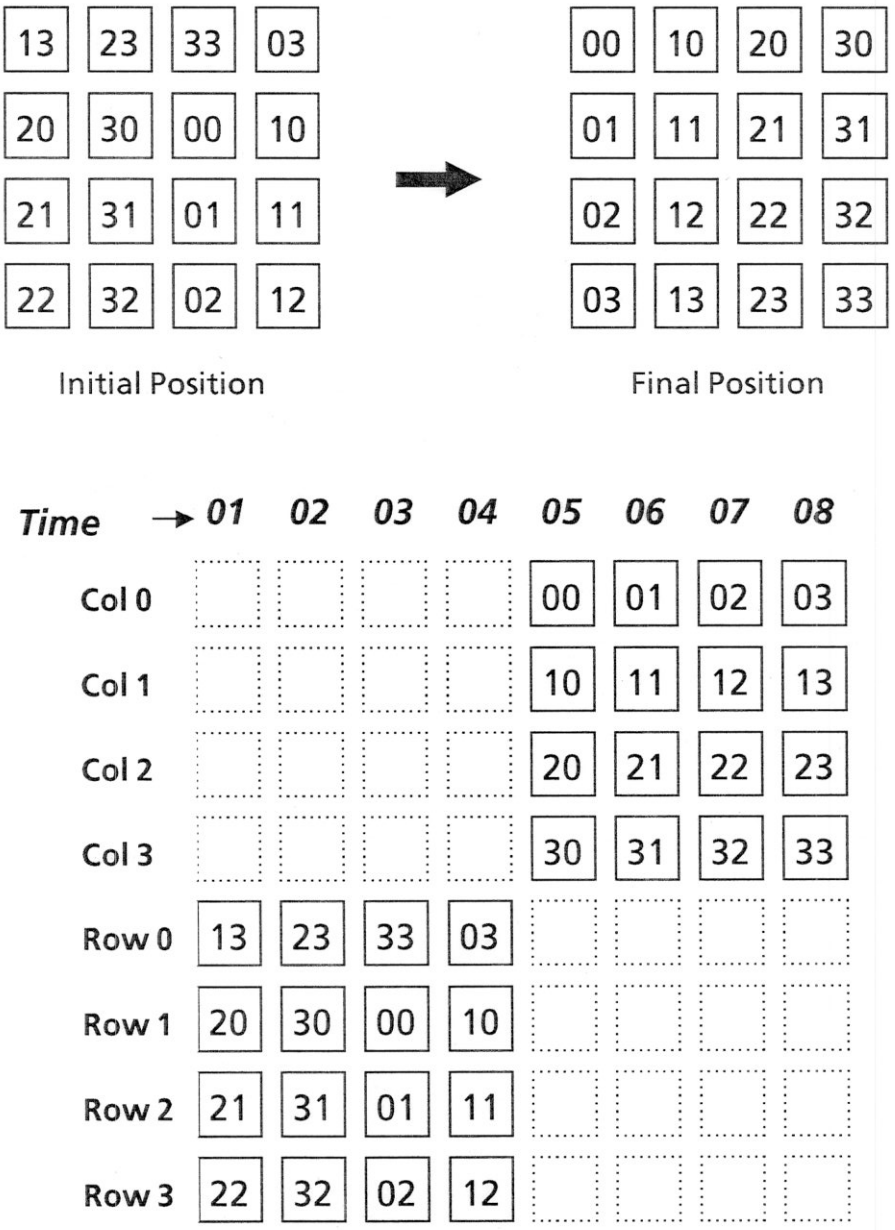


Figure 5.1: A simple schedule of $D_1(4,4)$

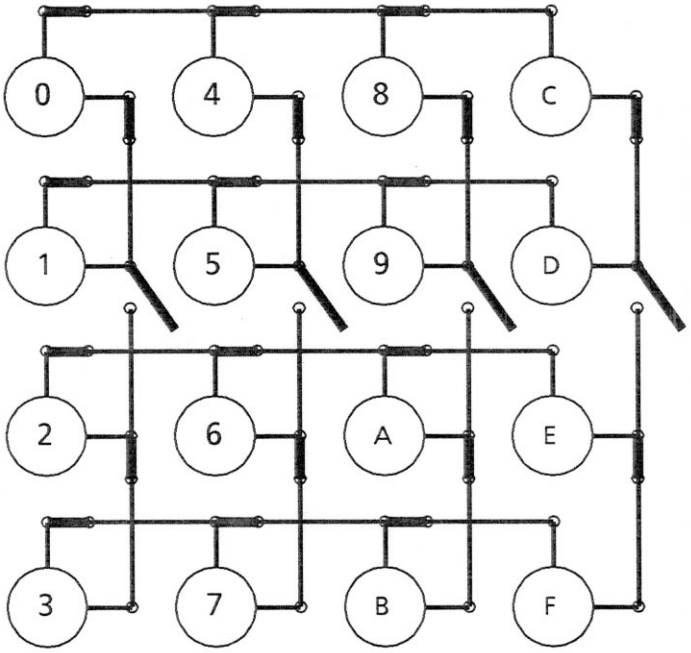


Figure 6.1: A partitionable *SPB(4,4)*

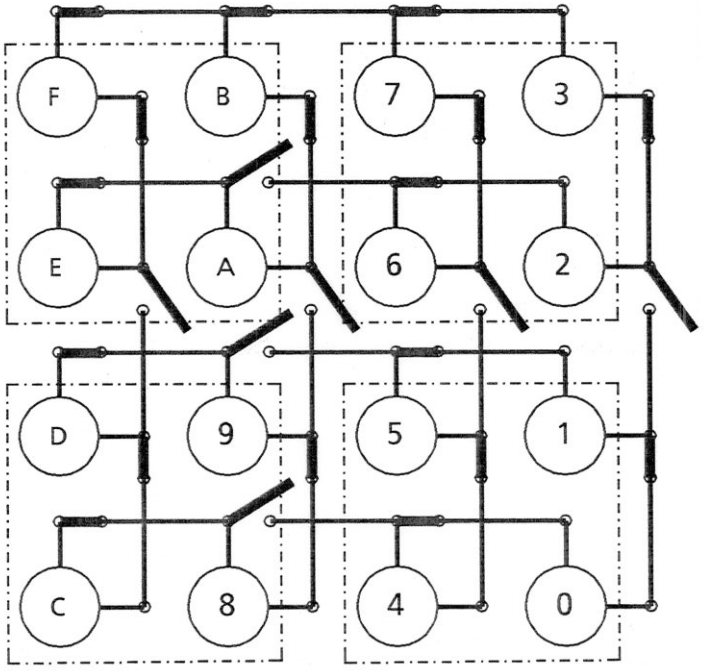


Figure 6.2: Destination addresses (the bit-reversal permutation)