BITONIC SORTING ON SUPERPOSED PARALLEL BUSES

Bruce W. Arden
Toshio Nakatani

CS-TR-063-86

November 1986

Bitonic Sorting 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

The paper develops time complexity measures, including constants, for bitonic sorting on non-partitionable and partitionable buses, in both linear and two dimensional cases. The P processing elements, which share the time multiplexed buses, have a single input port and single output port which are connected to the buses. The bus bandwidth, B, represented by the number of bus cycles per processor cycle, is a parameter of the model. This parameter can alternately be viewed as the number of parallel buses having one bus cycle per processor cycle. The routing over the buses is based on an optimal data movement for bitonic merge on a linear bus. For the two dimensional case, the execution time is close to optimal in terms of VLSI complexity.

November, 1986

Bitonic Sorting 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. Introductions

Area-efficient sorting is an important issue in parallel supercomputing (Leiserson[1983], Thompson[1983], Ullman[1984ab]). Sorting schemes based on the hypercube, the cube-connected-cycle, or the shuffle-exchange networks suffer from the quadratically growing wire area in VLSI implementations. Recent results based on the AKS algorithm (Ajtai, Komlos and Szemeredi[1983] and Bilardi and Preparata[1985a]) and related studies (Bilardi and Preparata[1985b] and Leighton[1985]) have shown logarithmic time performance for new, parallel sorting schemes. However, huge constant factors as well as the large wire area make these algorithms unrealistic for VLSI implementation. Thus, a simple parallel sorting scheme on a small area using a simple interconnection is still desirable for future systems. In other words, the constant factors in complexity measures are important in the evaluation of parallel sorting algorithms.

Bitonic sort (Batcher[1968] and Knuth[1973]) has been studied extensively. Its adaptations include the shuffle-exchange network (Stone[1971]), ILLIAC-IV (Orcutt[1976]), a mesh-connected computer (Thompson and Kung[1977] and Nassimi and Sahni[1979]), STAR (Stone[1978]), Ultracomputer (Meertens[1979]), a data-base machine (Jayanata and Hsiao[1979]), magnetic bubble memories (Chung, Luccio, and Wong [1980]), the cube-connected-cycles (Preparata and Vuillemin[1981]), the orthogonal tree (the mesh-of-trees) (Nath, Maheshwari, and Bhatt[1983] and Bonuccelli and Pagli[1984])), the pleated cube-connected

cycles (Bilardi and Preparata[1984]), the distributed computers (Loui[1984]), and more recently parallel computers with reduced hardware (Ja' Ja' and Owen[1984], Owen and Ja' Ja'[1985], Hsiao and Shen[1985], and Tseng, Hwang, and Kumar[1985]). Thompson[1983] also included several adaptations of bitonic sort in VLSI networks. Recently, the authors have extended this approach to a k-way bitonic sort (Arden and Nakatani[1986d]).

Interest in area-efficient adaptations of bitonic sort has returned to mesh-connected computers because of VLSI optimality (Thompson[1983]). One study showed that the broadcasting feature cannot improve the asymptotic performance of sorting on mesh-connected computers, although it can improve the time complexity of other algorithms such as semi-group computation (Stout[1983], Bokhari[1984] and Kumar and Raghavendra[1985]).

In this paper, we show that bus-partitionability as well as greater bus speed can improve the time performance of bitonic sorting on bus interconnections. The buses are shared by many processors and are not limited to adjacent neighbors as in mesh interconnections. First, we introduce an optimal data movement for bitonic merge on a linear bus. By using this data movement, we present an efficient adaptation of bitonic merge and sort on a two dimensional grid of superposed parallel buses. We also describe how the bus ratio, B, affects the time performance of bitonic merge and sort by means of the related VLSI complexity measures.

The paper is organized as follows. In section 2, we summarize the model of computation to understand the basic assumptions. In section 3, we review bitonic merge and sort and introduce an optimal data movement for bitonic merge on a linear bus. We develop time complexity measures for bitonic merge and sort on both a non-partitionable and a partitionable linear bus. In section 4, using the data movement developed in section 3, we further show how we can efficiently adapt this approach to a two dimensional grid of superposed parallel buses. We also develop time complexity measures for bitonic merge and sort in the two dimensional cases for both non-partitionable and partitionable buses. In section 5, we develop the VLSI complexity of bitonic merge and sort in both the linear and two dimensional cases, and the resultant optimality for the two

dimensional case.

2. Model of Computation

The model of computation we assume is a SIMD (Single Instruction stream Multiple Data stream) machine. That is, we assume a parallel computer with P identical processors that execute the same instructions. Each processor is essentially a comparator with a pair of registers. The instructions can be either broadcast from the central control unit or distributed to each processor site beforehand. For the former case, the system must be synchronous. That is, there is a single clock. For the latter case, each processor may have a clock but there must be synchronization at the bus interface. With this qualification, the algorithms we describe can be used in both cases.

Concerning the number of data items per processor, we assume that each processor has exactly one data item to sort. Therefore, the total number of data items, N, is equal to the number of processors, P. However, the problem in this allocation scheme is that half of the processors are always idle throughout the merge and sort processes. We shall discuss the more general case, where each processor has m data items and $N=mP$, in the separate paper. Here, we focus on the case of $N=P$.

To represent time complexity, we use t_r as the transmission time for a processor to send one data item to one of the other processors on a shared bus. We use t_c as the time for a processor to compare the contents of two registers. Obviously when a bus is partitioned, data movement can occur simultaneously on the partitions. A processor can make only one comparison at a time but collectively they operate concurrently to the extent that the algorithm permits.

2.1. Bus Interconnections

The interconnections between the processors are buses. We consider two cases: one is a non-partitionable bus, and the other is a partitionable bus. For the second case, a bus can be partitioned into arbitrary, contiguous small segments, so that the system is divided into many subsystems that can work independently. This feature is particularly important for a bus-connected

computer to efficiently execute recursive or divide-and-conquer algorithms (Arden and Ginosar[1982]).

Bus speed is represented by an integral ratio, B, which can be interpreted as the number of bus transmissions per processor cycle. Equivalently, B can be viewed as the number of parallel buses, where each is capable of one bus transmission per processor cycle. Clearly, $B \leq P$ and when $B=P$ for partitionable buses, bus partitionability is no longer productive. B is also an integral power of two. An interesting special case occurs when the P processors are partitioned into K equal-sized, independent segments. Since dense, binary addressing is used throughout, all of the size parameters must be integral powers of two. Accordingly, the number of processors, P, and the size of equal partitions, K, are also integral power of two.

We consider two interconnection topologies: the linear bus and the two dimensional grid of superposed parallel buses. For the linear bus (Figures 2.1a and 2.1b), we assume that P processors are connected by either a non-partitionable or a partitionable bus, with a bandwidth ratio B ($1 \leq B \leq P$). When $B=1$, a processor can access the bus either via an input or output port, but only one of these within a processor cycle. For $B \geq 2$, a processor can access one of the buses (or alternately, one bus cycle) via input port and another via output port within a processor cycle. For the two dimensional grid interconnection (Figures 2.2a and 2.2b), $P=p \times p$ processors, and every row and column of p processors are connected horizontally and vertically by a non-partitionable or a partitionable bus. For the two dimensional case, we assume $1 \leq B \leq p$.

In an actual implementation, the bandwidth B of the bus would have to be managed. Either the B time slots in processor cycle would have to be assigned to processors or, in the case of B processor-speed buses the buses would have to be assigned to specific processor pairs. In both cases, the assignment could be done *a priori* and carried out by a hardware bus manager, or arbitration unit. For the purposes of determining time, it is not necessary to know the order in which bus transmissions are made, but only the length of time the bus (or buses) are occupied in making the necessary transfers. Therefore, the algorithmic details of bus arbitration are not included in this paper.

2.2. Processor Design

The important feature of the processor design (Figure 2.3) in this paper is the ingate and outgate pointer, *pointer*, which selects one of a pair of registers, i.e., *input-register*[*pointer*], where *pointer*=0 or 1. In the conventional designs, bitonic sorting algorithms require register exchange (swap) operations requiring two processor cycles. We reduce this time commitment by introducing the ingate and outgate pointer, *pointer* to the pair of registers. After the contents of the pair of registers are compared, the condition code is set. The combination of the condition code, the order bit, and the flag bit determines which register's contents is transmitted on the bus.

The condition code, *cc*, will be set as follows:

cc=0 if *input-register*[0] $>$ *input-register*[1]

cc=1 if *input-register*[0] $\leq$ *input-register*[1]

The order bit, *order*, indicates whether the final order is to be increasing or decreasing:

order=0 means the sequence will be sorted by non-decreasing order.

order=1 means the sequence will be sorted by non-increasing order.

To achieve the specified final order, a particular pattern of low-high and high-low exchanges are needed in the stages preceding the final result. This comparison specification can be obtained from the order value and a simple bit function of the processor identification, *id*. More specifically, in the linear case, we can define the order for the segment of size $K/2$ as $order \bullet id[\log P - \log K]$, where $\bullet$ indicates *bit-exclusive-or*. In the two dimensional case, we can define the order for the segment of size $K/2 \times K/2$ as $order \bullet id[\log P - \log K]$.

The processors are assumed to be indexed by the preloaded processor identification registers, *id*. For the linear case, we assume that the processors are indexed in the natural order. For the two dimensional case, we assume that the processors are indexed in row-major order. All the processors execute the same (comparison) instructions but generate different transmission outcomes depending upon their indices and the keys of their data to be sorted. Within a processor, *id* may be treated as a bit vector. Thus, $id[\log P - \log K]$ selects a

particular bit in *id*.

The flag bit, *flag*, is used to represent whether the datum with the smaller key is to be broadcast or not:

flag=0 if the datum with the larger key is to be transmitted.

flag=1 if the datum with the smaller key is to be transmitted.

The ingate and outgate pointer, *pointer*, is determined by the following rules:

pointer=*flag*•*order*•*cc*.

If *pointer*=0, then the content of *input-register*[0] will be transmitted.

If *pointer*=1, then the content of *input-register*[1] will be transmitted.

3. Linear Bitonic Merge and Sort

Batcher's linear bitonic merge algorithm (Figures 3.1a and 3.1b) for a bitonic sequence of length K, X_K, on P/K contiguous segments of P processors on a linear bus can be described recursively as follows:

procedure *Bitonic-Merge*(X_K, *order*);
begin
  if $K \geq 2$ **begin**
    for X_K **do**
      Compare-and-Exchange(X_K, *order*);
    for each half of X_K in parallel **do**
      Bitonic-Merge($X_{K/2}$, *order*)
  end
end;

3.1. Optimal Data Movement for Linear Bitonic Merge

Performance in parallel sorting based on Batcher's bitonic merge depends on how efficiently we can move the data to the right position. Comparison may be carried out simultaneously as soon as all the data are moved to the right position. A simple way (Figure 3.1a) to move the data for *Compare-and-Exchange* of *Bitonic-Merge* is:

Data Movement 1:

procedure *Compare-and-Exchange*(X_K);

begin

Move the value x_j of processor j *to* processor i, where $i=j\ (mod\ K/2)$;

Compare-Exchange(x_i, x_j);

Move the result r_i *to processor* j, where $j>i$ and $i=j\ (mod\ K/2)$;

end;

The bitonic sorting algorithms (Thompson and Kung[1977] and Nassimi and Sahni[1979]) published for a mesh-connected computer are based on this scheme. However, there is inefficiency in this data movement because comparison results must be moved back to the original positions after they are compared.

If we can divide X_K into two halves X_e and X_o so that comparisons are always carried out between X_e and X_o throughout the $\log K$ steps, then we have no need to move the data back to the original place. Instead, we always move X_o to X_e. One method to divide X_N into halves is to use *parity* of indexes. For example, define X_e and X_o as follows:

$X_e=\{x_i \mid even\text{-}parity(i)=0\}$

$X_o=\{x_i \mid even\text{-}parity(i)=1\}$

That is, X_e is the data group whose index numbers have an even number of one's in the binary representation. X_o is the data group whose index numbers have an odd number of one's in the binary representation. Then, an improved way (Figure 3.1b) to move the data for *Compare-and-Exchange* of *Bitonic-Merge* is:

Data Movement 2:

procedure *Compare-and-Exchange*(X_K);

begin

if *initial* **then**

Move the value x_j of processor j *to* processor i, where $x_j \in X_o$ and $i=j \ (mod \ K/2)$;

Compare-Exchange(x_i, x_j);

if $K \geq 4$ **then**

Move the result r_j *to* processor l, where $x_j \in X_o$ and $j=l \ (mod \ K/4)$;

else

Move the result r_j *to the processor* i, where $x_j \in X_o$ and $i=j \ (mod \ K/2)$;

end;

The improvement in data movement is apparent, if we view it on the hypercube of dimension log$K$. For data movement 1 (Figure 3.3a), data moves along the edges of the hypercube and the data must move back to the original position before it moves to the destination of the next step. On the other hand, for data movement 2 (Figure 3.3b), data takes diagonal paths to move to the next destination. In Figures 3.2a and 3.2b, we also show two bitonic merge networks of different topologies corresponding to two data movements 1 and 2.

Furthermore, we can argue that data movement 2 is optimal. In Batcher's bitonic merge algorithm for K data items, at least $K/2$ data items must be moved to the new destinations from one stage to another. In data movement 2, we move exactly $K/2$ data items for the next comparison. Therefore, data movement 2 is optimal.

The advantage of data movement 1 is the unified direction and constant distance of the moves. The disadvantage of data movement 2 is the variable direction and distance of the moves. Therefore, data movement 1 is more suitable for the SIMD mode on a mesh interconnection. Data movement 2 is more

suitable for the SIMD mode on a bus interconnection. Throughout this paper, we use data movement 2. Due to the reduced number of routing steps using data movement 2, bus interconnections outperform mesh interconnections for bitonic sorting.

For completeness, we now describe *Compare-Exchange* in a little more detail:

```
procedure Compare-Exchange(X_K, order);
begin
    flag:=id[logP−logK];
    if input-register[0]≤input-register[1] then
        cc:=1
    else
        cc:=0;
    pointer:=flag•order•cc;
end;
```

3.2. Time Complexity of Bitonic Merge

The time complexity, $T_{BM}(K)$, of *Bitonic-Merge* for P/K bitonic sequences of length K can be described recursively. Each of the P processors has one data item and is connected to a linear bus of bandwidth B.

If a bus is non-partitionable ($B \leq P/2$),

$$T_{BM}(K) = \begin{cases} [2 \cdot \frac{P}{2B} + T_{BM}(\frac{K}{2})] \cdot t_r + 1 \cdot t_c & \text{if } initial^* \\ [\frac{P}{2B} + T_{BM}(\frac{K}{2})] \cdot t_r + 1 \cdot t_c & \text{if } 1 < K \\ 0 \cdot t_r + 0 \cdot t_c & \text{if } K = 1 \end{cases}$$

If a bus is partitionable ($B \leq P/2$),

$$T_{BM}(K) = \begin{cases} [2 \cdot \frac{K}{2B} + T_{BM}(\frac{K}{2})] \cdot t_r + 1 \cdot t_c & \text{if } initial^* \\ [\frac{K}{2B} + T_{BM}(\frac{K}{2})] \cdot t_r + 1 \cdot t_c & \text{if } B < K \\ [1 + T_{BM}(\frac{K}{2})] \cdot t_r + 1 \cdot t_c & \text{if } 1 < K \leq B \\ 0 \cdot t_r + 0 \cdot t_c & \text{if } K = 1 \end{cases}$$

* Note: extra cycles are required for initial positioning of X_o and X_e.

Then, by solving the above recurrence equation, we obtain the following equations:

If a bus is non-partitionable ($B \leq P/2$),

$$T_{BM}(K) = [\frac{P}{2B} \cdot \log 2K] \cdot t_r + [logK] \cdot t_c \qquad \text{if } K > 1$$

If a bus is partitionable ($B \leq P/2$),

$$T_{BM}(K) = [\frac{3}{2} \cdot \frac{K}{B} + \log B - 1] \cdot t_r + [logK] \cdot t_c \qquad \text{if } K > 1$$

3.3. Linear Bitonic Sort

Batcher's linear bitonic sort algorithm (Figures 3.4a and 3.4b) for an arbitrary sequence of length K, X_K, on P/K contiguous segments of P processors on a linear bus can be described recursively as follows:

procedure *Linear-Bitonic-Sort*(X_K, *order*);
begin
  if $K \geq 2$ **then begin**
    for each half of X_K in parallel **do**
      Linear-Bitonic-Sort($X_{K/2}$, *order*•*id*[log$P$—log$K$]);
    for X_K **do**
      Bitonic-Merge(X_K, *order*)
  end
end;

3.4. Time Complexity of Linear Bitonic Sort

The time complexity, $T_{LS}(K)$, of *Linear-Bitonic-Sort* for P/K arbitrary sequences of length K can be described recursively. Each of the P processors has one data item and is connected to a linear bus of bandwidth B.

If a bus is non-partitionable ($B \leq P/2$),

$$T_{LS}(K) = \begin{cases} T_{BM}(K)+T_{LS}(\frac{K}{2}) & \text{if } \textit{initial} \\ [T_{BM}(K)-\frac{P}{2B}]+T_{LS}(\frac{K}{2}) & \text{if } K>1 \\ 0\cdot t_r+0\cdot t_c & \text{if } K=1 \end{cases}$$

If a bus is partitionable ($B \leq P/2$),

$$T_{LS}(K) = \begin{cases} T_{BM}(K)+T_{LS}(\frac{K}{2}) & \text{if } \textit{initial} \\ [T_{BM}(K)-\frac{K}{2B}]+T_{LS}(\frac{K}{2}) & \text{if } K>1 \\ 0\cdot t_r+0\cdot t_c & \text{if } K=1 \end{cases}$$

Then, by solving the above recurrence equation, we obtain the following equations:

If a bus is non-partitionable ($B \leq P/2$),

$$T_{LS}(K) = [\frac{P}{2B}\cdot(\frac{1}{2}\log^2 K+\frac{1}{2}\log K+1)]\cdot t_r+[\frac{1}{2}\log^2 K+\frac{1}{2}\log K]\cdot t_c \quad \text{if } K>1$$

If a bus is partitionable ($B \leq P/2$),

$$T_{LS}(K) = [3\cdot\frac{K}{B}+(\log B-2)\cdot\log K-\frac{1}{2}\log^2 B+\frac{5}{2}\log B-2]\cdot t_r+[\frac{1}{2}\log^2 K+\frac{1}{2}\log K]\cdot t_c \quad \text{if } K>1$$

4. Two-Dimensional Bitonic Merge and Sort

Two dimensional bitonic merge algorithm requires two versions depending on the merge direction of two subfiles; vertical or horizontal. When row-major order is used, vertically merging two subfiles preserves the order of indices. On the other hand, horizontally merging two subfiles requires both merging and reordering. For this reason, horizontal merge requires a modified version of *Bitonic-Merge*, called *Bitonic-Merge-Relocate*, which involves special data movement (Figure 4.1). In *Bitonic-Merge-Relocate*, a bitonic sequence of length $2K$ is *globally-folded* (Hsiao and Shen[1985]) at contiguous K-length segments of the P processors on a linear bus. Also, each processor has exactly two data items. After the completion of the algorithm, a monotonic sequence of length $2K$ is *locally-folded* (Hsiao and Shen[1985]) at each K-length segment on a linear bus (Figure 4.1). Therefore, *Bitonic-Merge-Relocate* performs both merging and reordering. The algorithmic outline for *Compare-and-Exchange* process of *Bitonic-Merge* is:

Data Movement 3 (*Bitonic-Merge-Relocate*):

procedure *Compare-and-Exchange*(X_K);

begin

Compare-Exchange(x_i, y_i);

if $K \geq 2$ **then**

Move the result r_i *to* processor j, where $i=j$ ($mod\ K/2$);

end;

We can view it on the hypercube of dimension $\log K+1$ (Figure 4.2). For data movement 3, data moves to compare-exchange and it stops when it reaches the new destination.

4.1. Time Complexity of Bitonic Merge Relocate

The time complexity, $T_{MR}(K)$, of *Bitonic-Merge-Relocate* for P/K globally-folded, bitonic sequences of length $2K$ can be described recursively. Each processor has two data items and is connected to a linear bus of bandwidth B.

If buses are non-partitionable ($B \leq P$),

$$T_{MR}(K) = \begin{cases} [\frac{P}{B}+T_{MR}(\frac{K}{2})]\cdot t_r+1\cdot t_c & \text{if } K>1 \\ 0\cdot t_r+1\cdot t_c & \text{if } K=1 \end{cases}$$

If buses are partitionable ($B \leq P$),

$$T_{MR}(K) = \begin{cases} [\frac{K}{B}+T_{MR}(\frac{K}{2})]\cdot t_r+1\cdot t_c & \text{if } K>1 \\ 0\cdot t_r+1\cdot t_c & \text{if } K=1 \end{cases}$$

Then, by solving the above recurrence equation, we obtain the following equations:

If buses are non-partitionable ($B \leq P$),

$$T_{MR}(K) = [\frac{P}{B}\cdot \log K]\cdot t_r+[log K]\cdot t_c \qquad \text{if } K>1$$

If buses are partitionable ($B \leq P$),

$$T_{MR}(K) = [2 \cdot \frac{K}{B} + \log B - 2] \cdot t_r + [logK] \cdot t_c \qquad \text{if } K > 1$$

4.2. Vertical Merge

Vertical merge algorithm (Figures 4.3a, 4.3b, and 4.3c) for a bitonic sequence of length JK, $X_{J,K}$, on P/JK $J \times K$ rectangular segments of $P = p \times p$ processors on superposed parallel buses can be described as:

procedure *Vertical-Merge*($X_{J,K}$, *order*);
begin
  for each column k ($0 \leq k \leq K-1$) in parallel **do**
    Bitonic-Merge(X_J, *order*);
  for each row j ($0 \leq j \leq J-1$) in parallel **do**
    Bitonic-Merge(X_K, *order*)
end;

4.3. Time Complexity of Vertical Merge

The time complexity, $T_{VM}(J,K)$, of *Vertical-Merge* for P/JK bitonic sequences of length JK can be described recursively. Each of the $P = p \times p$ processors has one data item and is connected to a grid of superposed parallel buses of bus bandwidth B.

$$T_{VM}(J,K) = T_{BM}(J) + T_{BM}(K)$$

Then, by solving the above recurrence equation, we obtain the following equations:

If buses are non-partitionable ($B \leq p/2$),

$$T_{VM}(J,K) = \begin{cases} [\frac{p}{2B} \cdot \log 4JK] \cdot t_r + [\log JK] \cdot t_c & \text{if } J \geq 2 \text{ and } K > 1 \\ 2 \cdot \frac{p}{2B} \cdot t_r + 1 \cdot t_c & \text{if } J = 2 \text{ and } K = 1 \end{cases}$$

If buses are partitionable ($B \leq p/2$),

$$T_{VM}(J,K) = \begin{cases} [\frac{3}{2} \cdot \frac{(J+K)}{B} + 2\log B - 2] \cdot t_r + [\log JK] \cdot t_c & \text{if } J \geq 2 \text{ and } K > 1 \\ 2 \cdot t_r + 1 \cdot t_c & \text{if } J = 2 \text{ and } K = 1 \end{cases}$$

4.4. Horizontal Merge

Horizontal merge algorithm (Figures 4.4a, 4.4b, and 4.4c) for a bitonic sequence of length JK, $X_{J,K}$, on P/JK $J \times K$ rectangular segments of $P = p \times p$ processors on superposed parallel buses can be described as:

procedure *Horizontal-Merge*($X_{J,K}$, *order*);

begin

for each row j ($0 \leq j \leq J-1$) in parallel **do**

Move the value x_l of processor l *to* processor k, where $l > k$ and $k = l$ (*mod* $K/2$);

for each column k ($0 \leq k \leq K-1$) in parallel **do**

Bitonic-Merge-Relocate(X_J, *order*);

for each row j ($0 \leq j \leq J-1$) in parallel **do**

Move the result r_k *to* the processor l, where $l > k$ and $k = l$ (*mod* $K/2$);

for each half of row j ($0 \leq j \leq J-1$) in parallel **do**

Bitonic-Merge($X_{K/2}$, *order*)

end;

4.5. Time Complexity of Horizontal Merge

The time complexity, $T_{HM}(J,K)$, of *Horizontal-Merge* for P/JK bitonic sequences of length JK can be described recursively. Each of the $P = p \times p$ processors has one data item and is connected to a grid of superposed parallel buses of bus bandwidth B.

If buses are non-partitionable ($B \leq p/2$),

$$T_{HM}(J,K) = [2 \cdot \frac{p}{2B}] \cdot t_r + T_{MR}(J) + T_{BM}(\frac{K}{2})$$

If buses are partitionable ($B \leq p/2$),

$$T_{HM}(J,K) = \begin{cases} [2 \cdot \frac{K}{2B}] \cdot t_r + T_{MR}(J) + T_{BM}(\frac{K}{2}) & \text{if } K > B \\ 2 \cdot t_r + T_{MR}(J) + T_{BM}(\frac{K}{2}) & \text{if } K \leq B \end{cases}$$

Then, by solving the above recurrence equation, we obtain the following equations:

If buses are non-partitionable ($B \leq p/2$),

$$T_{HM}(J,K) = \begin{cases} [\frac{p}{2B} \cdot (2 log J + \log K + 2)] \cdot t_r + [\log JK] \cdot t_c & \text{if } J \geq 2 \text{ and } K > 2 \\ 4 \cdot \frac{p}{2B} \cdot t_r + 2 \cdot t_c & \text{if } J = 2 \text{ and } K = 2 \\ 2 \cdot \frac{p}{2B} \cdot t_r + 1 \cdot t_c & \text{if } J = 1 \text{ and } K = 2 \end{cases}$$

If buses are partitionable ($B \leq p/2$),

$$T_{HM}(J,K) = \begin{cases} [2 \cdot \frac{J}{B} + \frac{7}{4} \cdot \frac{K}{B} + 2 \log B - 3] \cdot t_r + [\log JK] \cdot t_c & \text{if } J \geq 2 \text{ and } K > 2 \\ [2 + \frac{2}{B}] \cdot t_r + 2 \cdot t_c & \text{if } J = 2 \text{ and } K = 2 \\ 2 \cdot t_r + 1 \cdot t_c & \text{if } J = 1 \text{ and } K = 2 \end{cases}$$

4.6. 2D Bitonic Sort

The two dimensional bitonic sort algorithm (Figures 4.5a, 4.5b, 4.5c, and 4.5d) is simply a pair of vertical and horizontal merges. However, the order depends on the time complexity of vertical and horizontal merges. That is, if vertical merge takes fewer cycles than horizontal merge, then horizontal merge should precede vertical merge to save cycles. This is the case when each processor has one data item.

Two dimensional bitonic sort algorithm for an arbitrary sequence of length K^2, $X_{K,K}$, on P/K^2 $K\times K$ square segments of $P=p\times p$ processors on superposed parallel buses can be described as:

procedure *2D-Bitonic-Sort*($X_{K,K}$, *order*):

begin

if $K\geq 2$ **then begin**

for each quarter of $X_{K,K}$ in parallel **do**

2D-Bitonic-Sort($X_{K/2,K/2}$, *order*•*id*[2*logp*−log*K*]);

for each half of $X_{K,K}$ in parallel **do**

Horizontal-Merge($X_{K/2,K}$, *order*•*id*[2*logp*−2log*K*]);

for $X_{K,K}$ **do**

Vertical-Merge($X_{K,K}$, *order*)

end

end;

4.7. Time Complexity of 2D Bitonic Sort

The time complexity, $T_{2S}(K,K)$, of *2D-Bitonic-Sort* for P/K^2 arbitrary sequences of length K^2 can be described recursively. Each of the $P=p\times p$ processors has one data item and is connected to a grid of superposed parallel buses of bus bandwidth B.

$$T_{2S}(K,K) = \begin{cases} T_{2S}(\frac{K}{2},\frac{K}{2})+T_{HM}(\frac{K}{2},K)+T_{VM}(K,K) & \text{if } K>1 \\ 0\cdot t_r+0\cdot t_c & \text{if } K=1 \end{cases}$$

Then, by solving the above recurrence equation, we obtain the following equations:

If buses are non-partitionable ($B\leq p/2$),

$$T_{2S}(K,K) = [\frac{p}{2B}\cdot(\frac{5}{2}\cdot\log^2 K+\frac{9}{2}\cdot\log K)]\cdot t_r+[2\log^2 K+\log K]\cdot t_c \qquad \text{if } K>1$$

If buses are partitionable ($B \leq p/2$),

$$T_{2S}(K,K) = [\frac{23}{2} \cdot \frac{K}{B} + (4\log B - 5) \cdot \log K - 2\log^2 B + 10\log B - 12] \cdot t_r + [2\log^2 K + \log K] \cdot t_c \quad \text{if } K > 1$$

5. VLSI Complexity of Sorting

We discuss two complexity measures for VLSI circuits. First, we consider conventional AT^2 measures (Thompson[1979]) for the point-to-point networks. Second, we consider AT^2M^2 measures which are more applicable for multi-point networks (Ullman[1984b]).

5.1. AT^2 Measures

We are concerned about the chip area, A, and the computation time, T of the sorting networks. The basic assumptions we make on a VLSI model of sorting are based on Thompson[1979] and also Bilardi and Preparata[1984]. Their main features are as follows:

1) The *synchronous* model: we assume *unit-delay* for a wire.
2) The *semellective* model: we assume *one-time-and-place* availability of inputs.
3) The *when-and-where-determinate* model: we assume *the predetermined* time and places for inputs and outputs.
4) The *word-local* model: we assume the *same* input port for all the bits of a word.

Under these assumptions, $AT^2 = O(P^2 \log^2 P)$ lower bound can be obtained (Thompson[1979]) for sorting P words of $(1+\epsilon) \log P$ bits each ($\epsilon > 0$).

For both linear and two dimensional bus interconnections with P processors, the area is $A = O(P \log^2 P)$. We assume that bus bandwidth, B, is constant and independent of P. We also assume $B \leq P$. For a partitionable linear bus, the time for bitonic sorting is $T = O(P)$ and $AT^2 = O(P^3 \log^2 P)$. On the other hand, for a two dimensional grid of partitionable superposed parallel buses, $T = O(\sqrt{P})$ and $AT^2 = O(P^2 \log^2 P)$, and it is thus optimal within a constant factor.

A two dimensional mesh interconnection has the same area, time, and AT^2 asymptotic complexities as a two dimensional grid of superposed parallel buses. However, if we take the constant factors into consideration, bus interconnections outperform mesh interconnections for time and AT^2 complexities.

5.2. AT^2M^2 Measures

As a next step, we consider VLSI complexity for the *multi-point* and high "*flux*" networks proposed by Ullman[1984b]. That is, we assume a high-bandwidth bus where each processor on the bus can transmit one data item to its destination processor on the same bus within a processor cycle. Under this assumption, $AT^2M^2=O(P^2)$ lower bound can be obtained (Ullman[1984b]), where M is the maximum number of processors supportable on a bus within a unit of time. For our model, we can assume $M=B=P$ for a linear bus and $M=B=\sqrt{P}$ for a two dimensional grid of superposed parallel buses.

For a linear bus, the time for bitonic sorting is $T=O(log^2P)$ and $AT^2M^2=O(P^3\log^6P)$. On the other hand, for a two dimensional grid of superposed parallel buses, $T=O(log^2\sqrt{P})$ and $AT^2M^2=O(P^2\log^6P)$, and it is thus optimal within a logarithmic factor.

6. Conclusions

We developed time complexity measures, including constants, for bitonic sorting on non-partitionable and partitionable buses, for both linear and two dimensional cases. First, an optimal data movement for bitonic merge and sort on a linear bus was given. Using this improved data routing over the buses, an efficient adaptation of bitonic merge and sort on one and two dimensional bus interconnections was devised. It was also shown that bus-partitionability as well as greater bus speed improves the time performance of sorting on bus interconnections, with the result that partitionable bus interconnections outperform one and two dimensional mesh interconnections executing bitonic merge and sort. Particularly for the two dimensional case, a grid of superposed parallel buses is optimal with regard to VLSI complexity, within a constant factor or logarithmic factor using AT^2 and AT^2M^2 measures, respectively.

Partitionable bus interconnections are not always advantageous, however. We have shown that bus-partitionability cannot improve the performance of permutations on bus interconnections (Arden Nakatani[1986a,b,c,e]). Ongoing studies include adaptations of other parallel sorting algorithms to bus interconnections.

References

Ajtai, M., J. Komlos and E. Szemeredi [1983]. "An O(NlogN) sorting network," *Proc. 15th ACM Symp. on Theory of Computing,* pp. 1-9.

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

Arden, B. and T. Nakatani [1986a]. "Superposed parallel buses: a systolic area-time optimal VLSI interconnection," Technical Report CS-TR-019-86, Department of Computer Science, Princeton University.

Arden, B. and T. Nakatani [1986b]. "Permutations on superposed parallel buses," Technical Report CS-TR-024-86, Department of Computer Science, Princeton University. (*submitted for publication*)

Arden, B. and T. Nakatani [1986c]. "The optimal uniform schedules of arbitrary static permutations on superposed parallel buses," Technical Report CS-TR-033-86, Department of Computer Science, Princeton University.

Arden, B. and T. Nakatani [1986d]. "K-way bitonic sort," Technical Report CS-TR-040-86, Department of Computer Science, Princeton University. (*submitted for publication*)

Arden, B. and T. Nakatani [1986e]. "Bus partitionability and parallel permutations," Technical Report CS-TR-041-86, Department of Computer Science, Princeton University.

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

Bilardi, G. and F. P. Preparata [1984]. "An architecture for bitonic Sorting with optimal VLSI performance," *IEEE Trans. on Computers* C-33:7, pp. 646-651.

Bilardi, G. and F. P. Preparata [1985a]. "The VLSI optimality of the AKS sorting network," *Information Processing Letters* 20:2, pp. 55-59.

Bilardi, G. and F. P. Preparata [1985b]. "A minimum VLSI network for O(log n) time sorting," *IEEE Trans. on Computers* C-34:4, pp. 336-343.

Bokhari S. H. [1984]. "Finding maximum on an array processor with a global bus," *IEEE Trans. on Computers* C-33:2, pp.133-139.

Bonuccelli M. A. and L. Pagli [1984]. "External sorting in VLSI," *IEEE Trans. on Computers* C-33:10, pp. 931-934.

Chung K. M., F. Luccio, C. K. Wong [1980]. "On the complexity of sorting in magnetic bubble memory systems," *IEEE Trans. on Computers* C-29:7, pp. 553-563.

Hsiao C.C. and N. Shen [1985]. "k-Fold bitonic sort on a mesh-connected parallel computer," *Information Processing News Letters* 21:10, pp. 207-212.

Ja' Ja' J. and R. M. Owens [1984]. "VLSI sorting with reduced hardware," *IEEE Trans. on Computers* C-33:7, pp. 668-671.

Jayanata B. and D. K. Hsiao [1979]. "Parallel bitonic record sort - an effective algorithm for the realization of a post processor," Technical Report OSU-CISRC-TR-79-1, Ohio State University Columbus Computer and Information Science Research Center.

Knuth, D. E. [1973]. *The Art of Computer Programming, Vol 3: Sorting and Searching.* Reading, M.A., Addison-Wesley.

Kumar, V. K. and C. S. Raghavendra [1985]. "Array processor with multiple broadcasting," *Proc. of 12th Annual Symposium on Computer Architecture,* pp. 2-10.

Leighton, T. [1985]. "Tight bound on the complexity of parallel sorting," *IEEE Trans. on Computers* C-34:4: pp. 344-354.

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

Loui M. C. [1984]. "The complexity of sorting on distributed systems," *Information and Control* 60, pp. 70-85.

Meertens L. G. L. T. [1979]. "Bitonic sort on ultracomputers," Technical Report MC-IW-117-79, Mathematisch Centrum, Amsterdam, Netherlands.

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

Nath D., S. N. Maheshwari, and P. C. P. Bhatt [1983]. "Efficient VLSI networks for parallel processing based on orthogonal trees," *IEEE Trans. on Computers* C-32:6, pp. 569-581.

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

Owens R. M. and J. Ja' Ja' [1985]. "Parallel sorting with serial memories," *IEEE Trans. on Computers* C-34:4, pp. 379-383.

Preparata F. P. and J. Vuillemin [1981]. "The cube-connected cycles: a versatile network for parallel computation," *Comm. of ACM* 24:5, pp. 300-309.

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

Stone, H. S. [1978]. "Sorting on STAR," *IEEE Trans. on Software Engineering* SE-4:3, pp. 138-146.

Stout, Q. F. [1983]. "Mesh-connected computers with broadcasting," *IEEE Trans. on Computers* C-32:9, pp. 153-161.

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

Thompson, C.D. [1979]. "A complexity theory for VLSI," PH.D. Dissertation, Carnegie-Mellon University, Pittsburgh, PA.

Thompson, C.D. [1983]. "The VLSI complexity of sorting," *IEEE Trans. on Computers* C-32:12, pp. 1171-1184.

Tseng P. S., K. Hwang, and Prasanna Kumar V. K. [1985]. "A VLSI-based multiprocessor architecture for implementing parallel algorithms," *Proc. of IEEE International Conference on Parallel Processing,* pp. 657-664.

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

Ullman, J. D. [1984b]. "VLSI complexity and supercomputers," *Proc. Fourth Jerusalem Conf. on Information Technology,* May, pp.646-649.

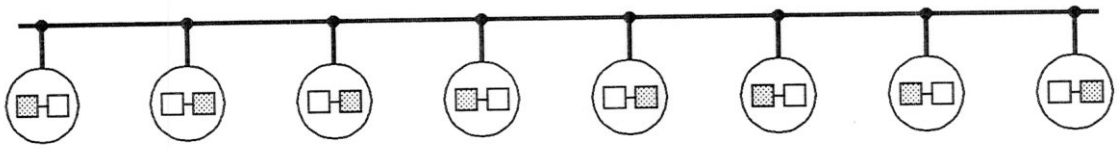


Figure 2.1a: A linear bus ($P = 8$)

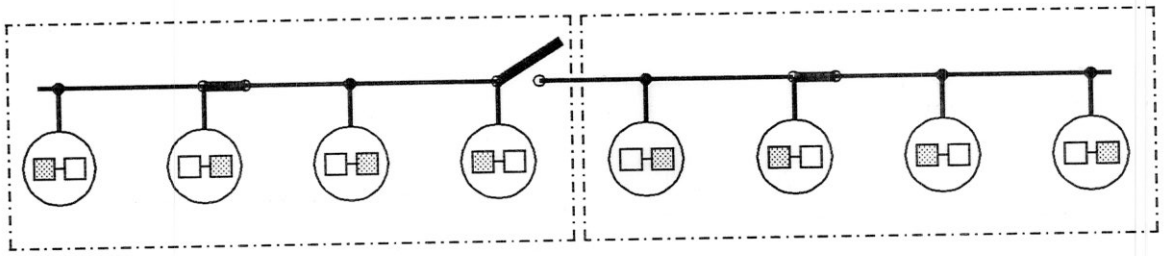


Figure 2.1b: A partitionable linear bus ($P = 8$)

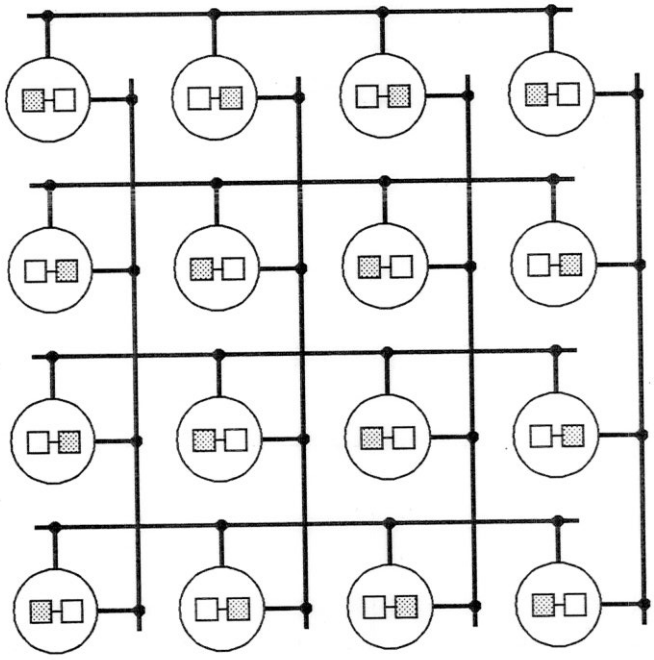


Figure 2.2a: A square grid of superposed parallel buses ($P = 4x4$)

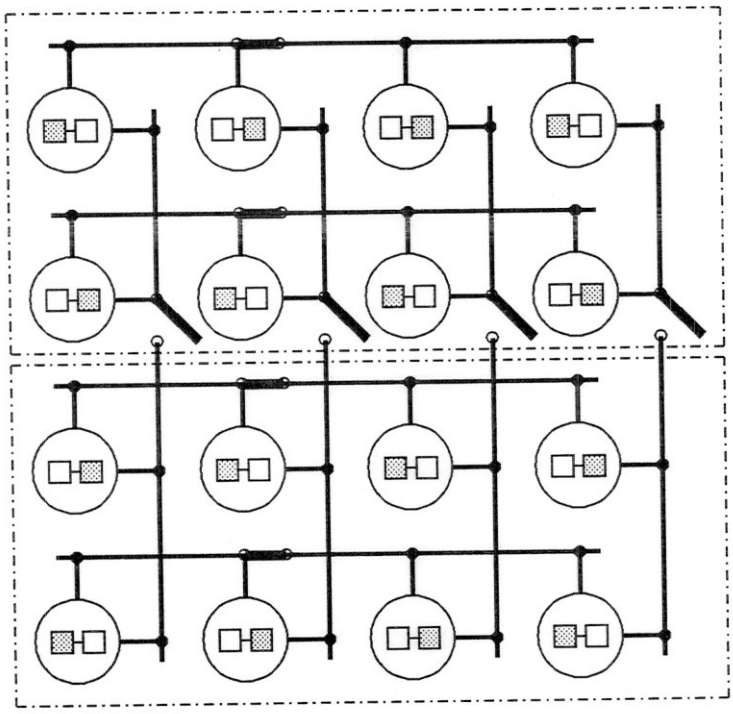


Figure 2.2b: A square grid of partitionable superposed parallel single-buses ($P = 4x4$)

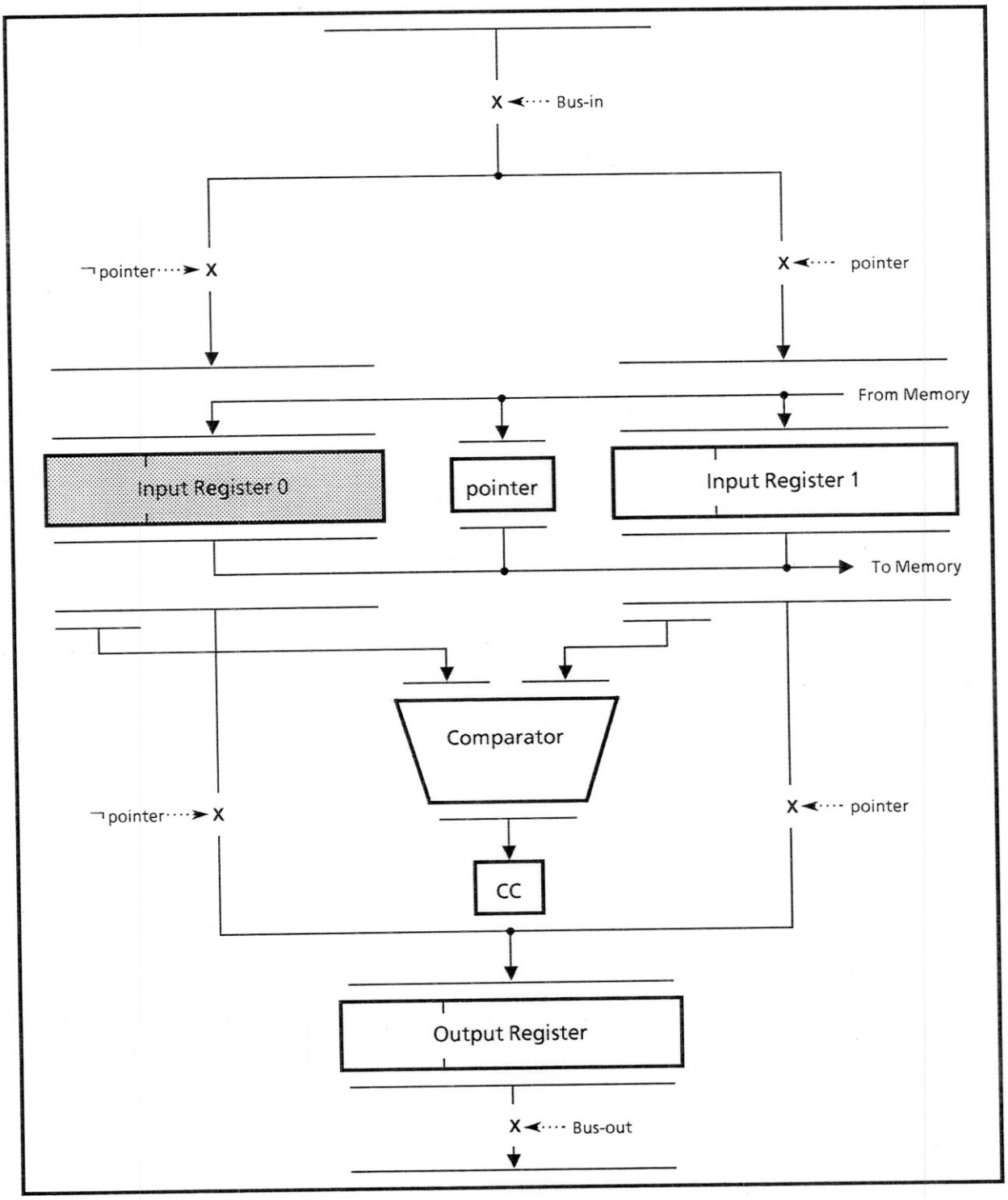


Figure 2.3: A processor chip for bitonic sorting

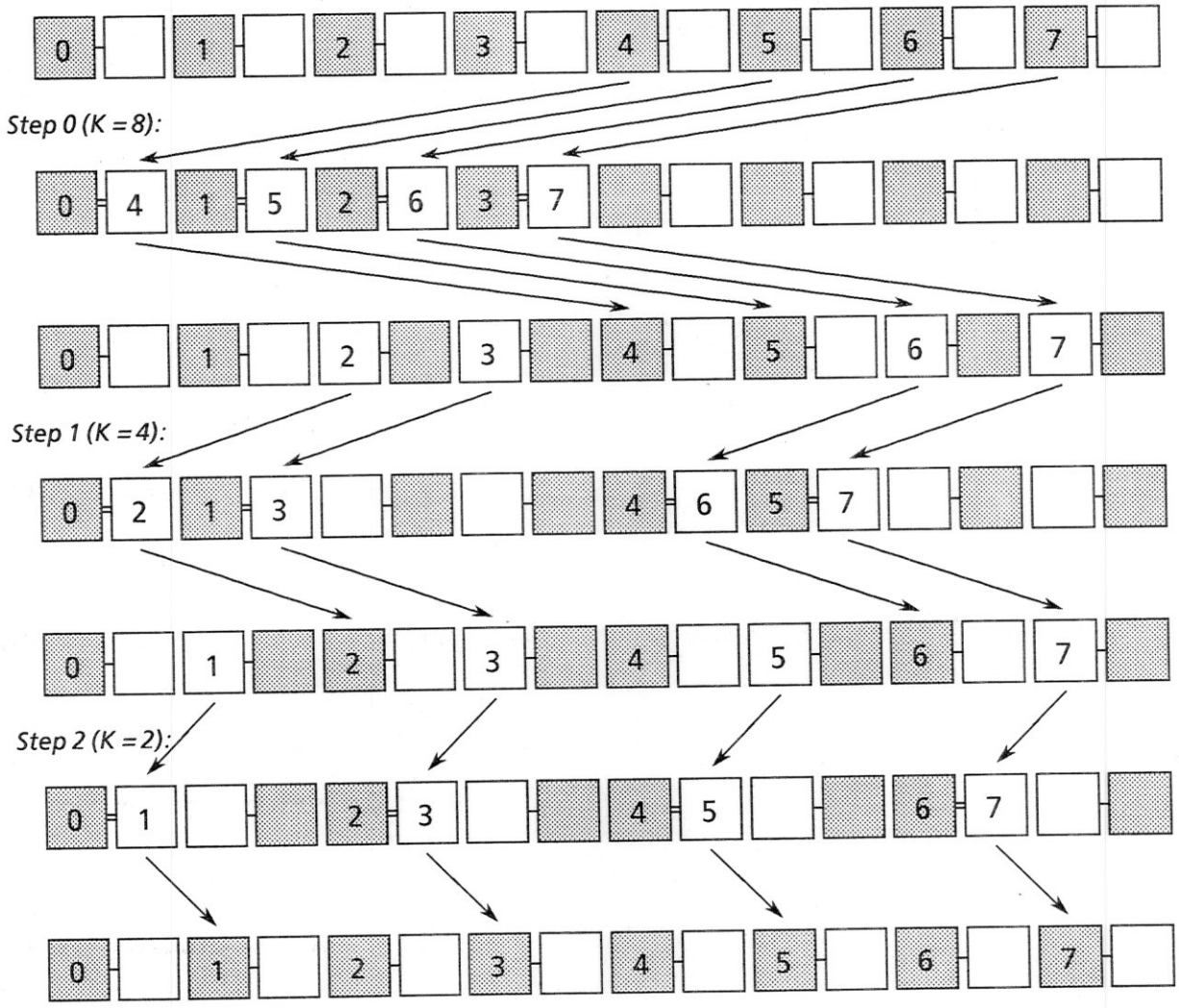


Figure 3.1a: Data movement 1 for the bitonic-merge algorithm ($N = P$, $P = 8$)

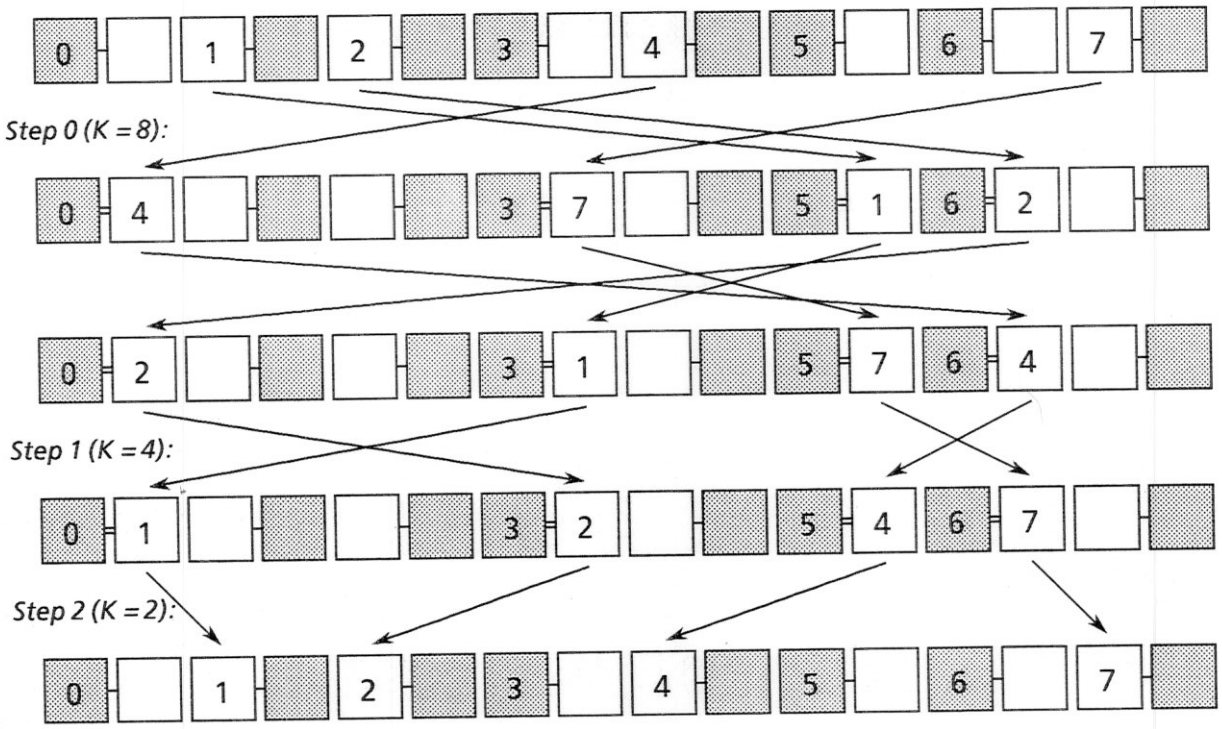


Figure 3.1b: Data movement 2 for the bitonic-merge algorithm ($N = P$, $P = 8$)

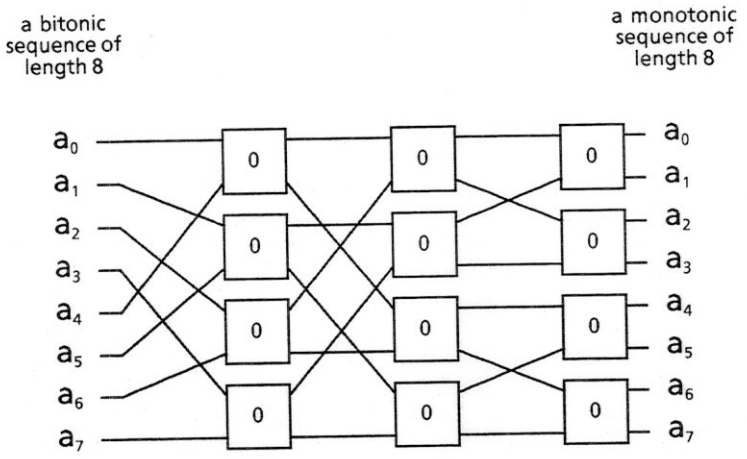
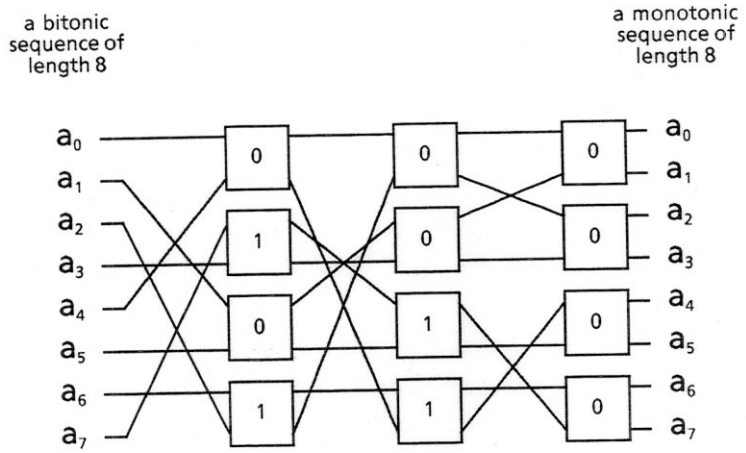


Figure 3.2a: A 8-bitonic merger (Data movement 1)



Note: The comparator with flag = 1 reverses the output.

Figure 3.2b: A 8-bitonic merger (Data movement 2)

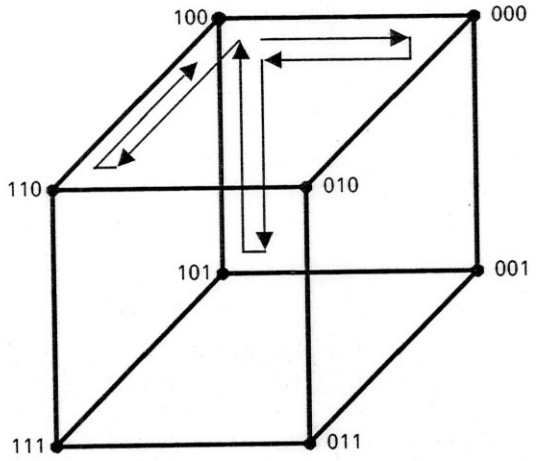


Figure 3.3a: Data movement 1 on the 3-cube

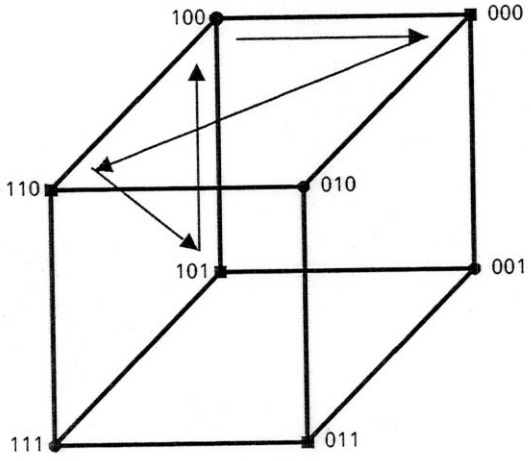
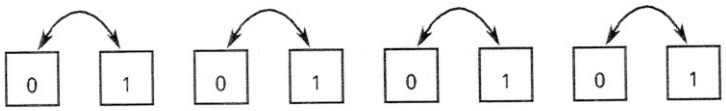
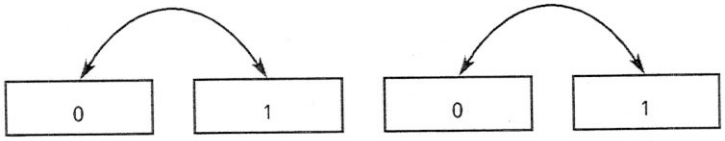


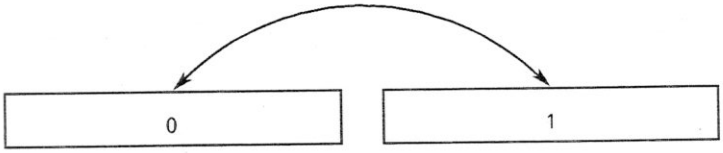
Figure 3.3b: Data movement 2 on the 3-cube



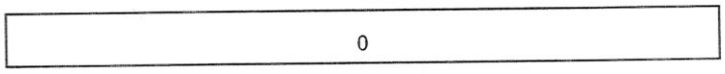
Phase 0 : bitonic-merge (K = 2, order = id[1])



Phase 1 : bitonic-merge (K = 4, order = id[0])



Phase 2 : bitonic-merge (K = 8, order = 0)



Final output (non-decreasing order)

Figure 3.4a: Linear Bitonic-sort ($N = P$, $P = 8$)

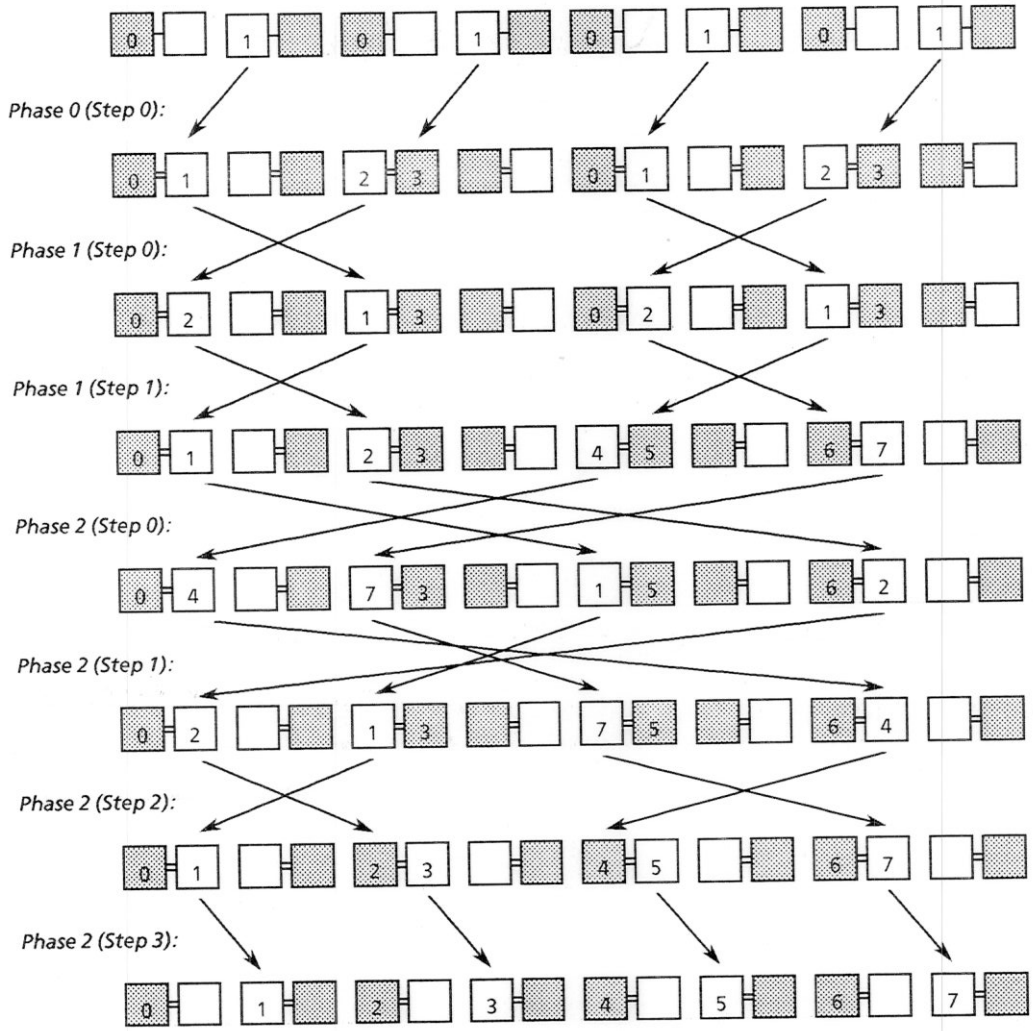


Figure 3.4b: Data movement for the linear-bitonic-sort algorithm ($N = P$, $P = 8$)

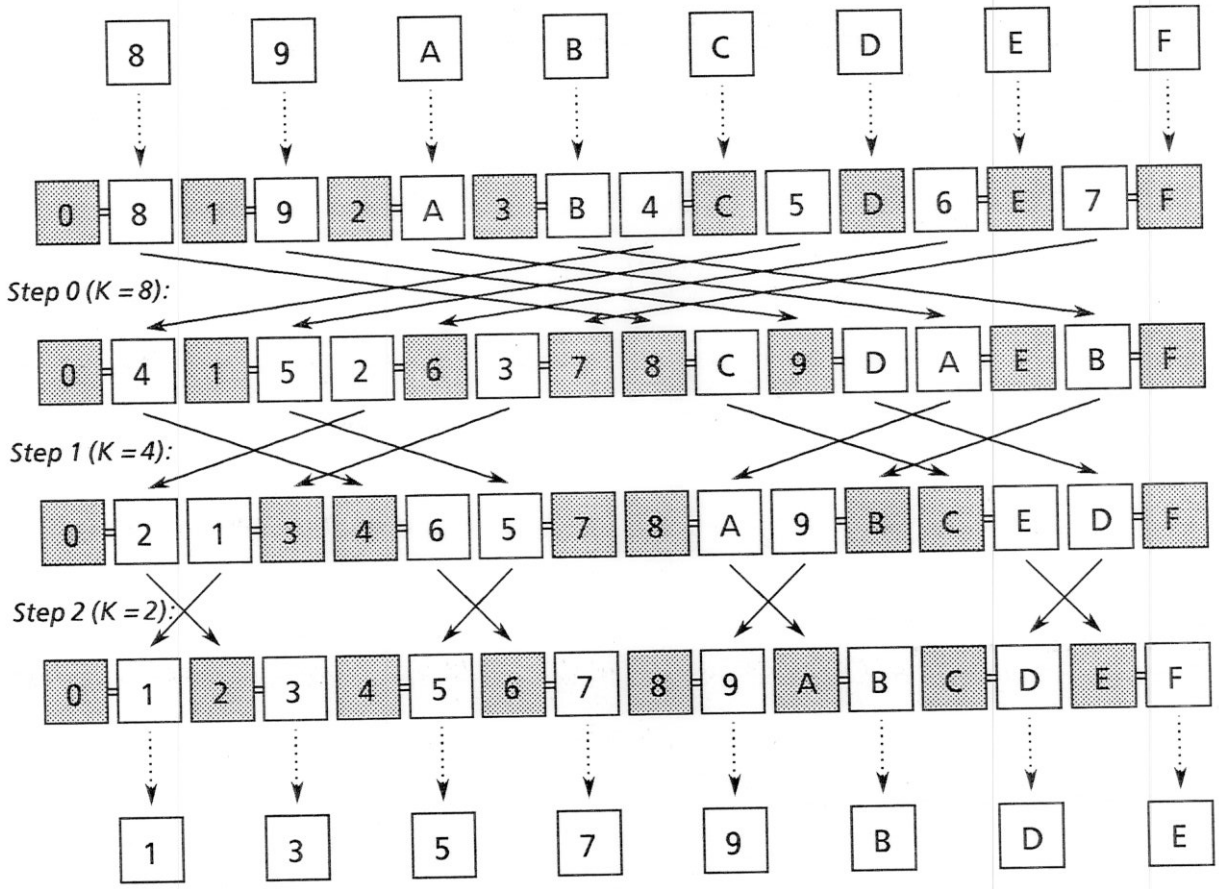


Figure 4.1: Data movement 3 for the bitonic-merge-relocate ($N = 2P$, $P = 8$)

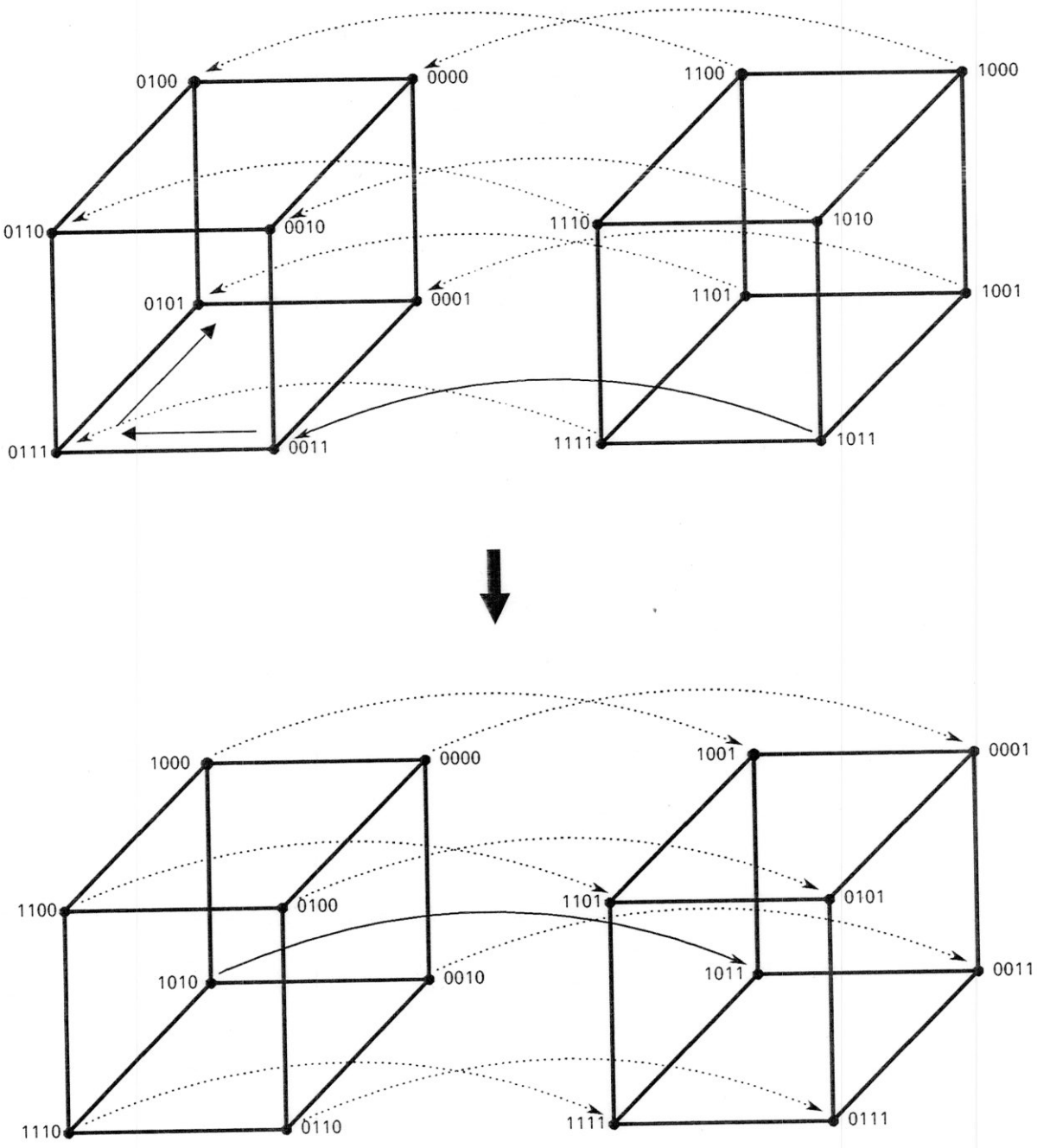
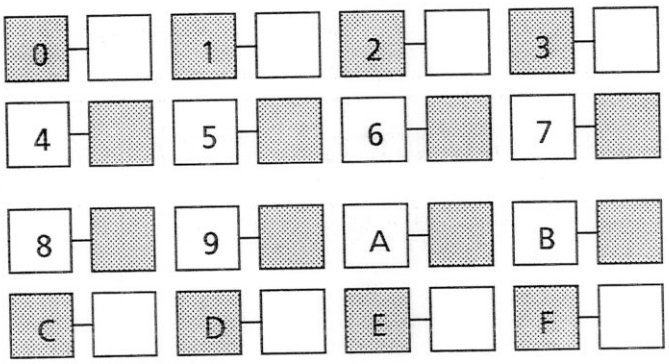
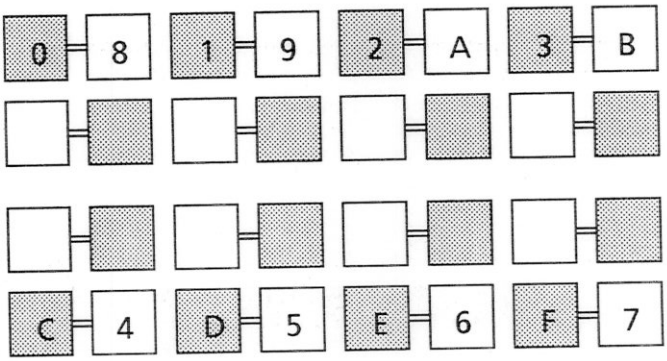


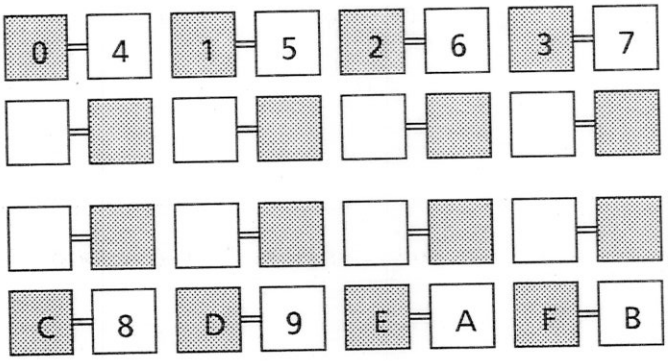
Figure 4.2: Data movement 3 on the 4-cube



Initial (J = K = 4)



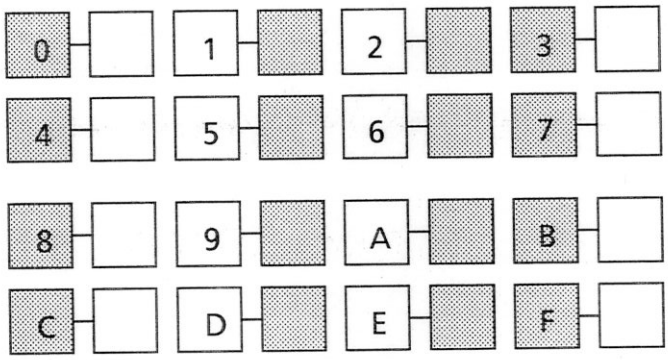
Step 0-0 (J = 4)



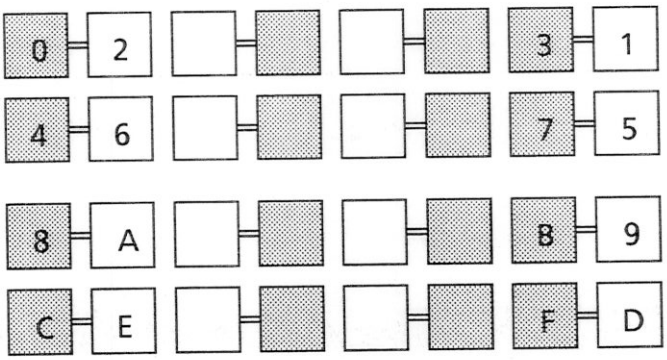
Step 0-1 (J = 4)

Note:  indicates a comparison between registers.

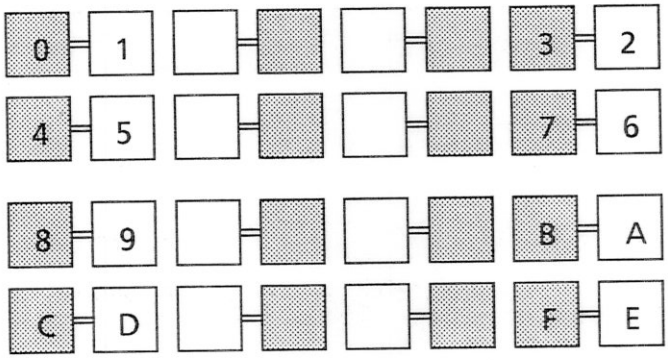
Figure 4.3a: Vertical-merge (*N = P, P = 4x4*)



Step 0-2



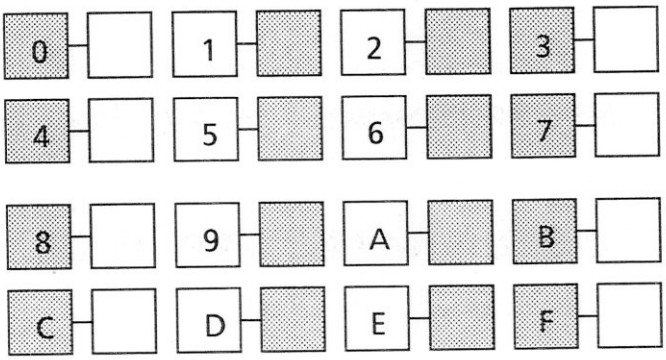
Step 1-0 (K = 4)



Step 1-1 (K = 4)

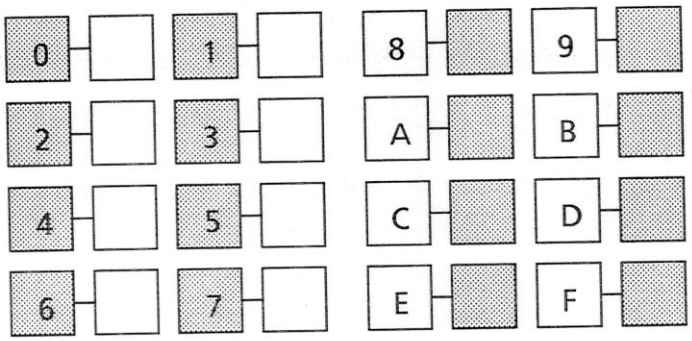
Note:  indicates a comparison between registers.

Figure 4.3b: Vertical-merge (*N = P, P = 4x4*)

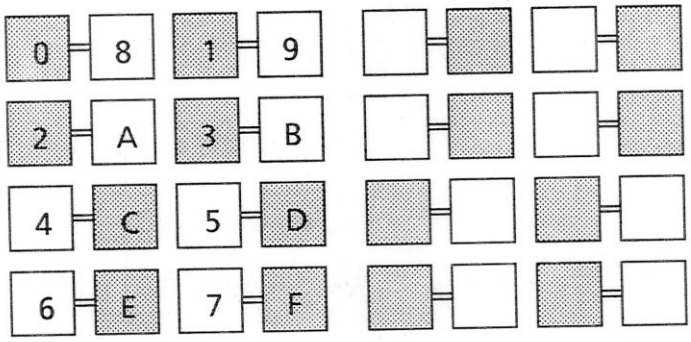


Step 1-2

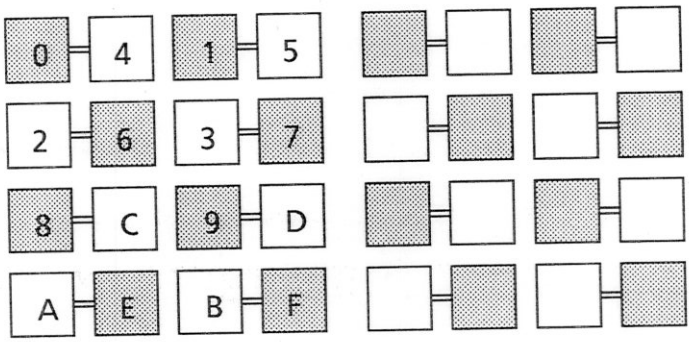
Figure 4.3c: Vertical-merge (*N* = *P*, *P* = *4x4*)



Initial (J = K = 4)



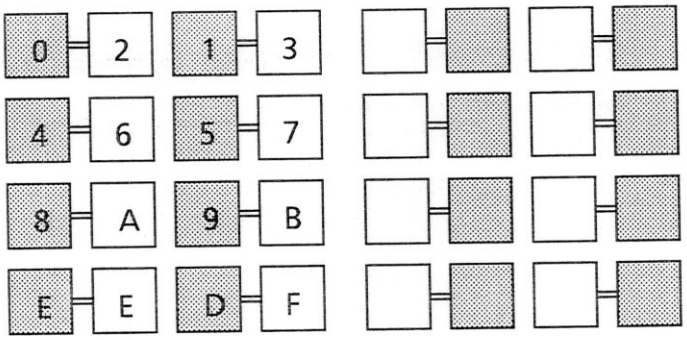
Step 0-0 (J = 4, K = 2)



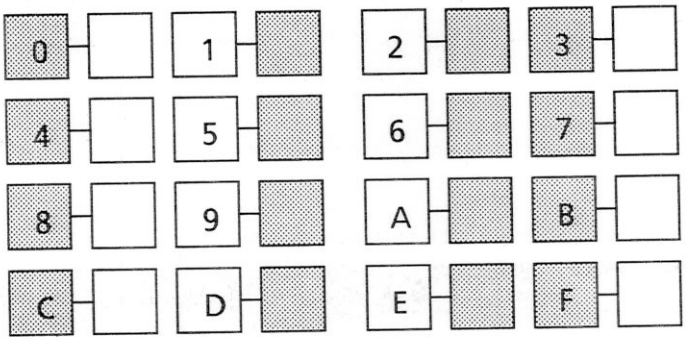
Step 0-1 (J = 4)

Note:  indicates a comparison between registers.

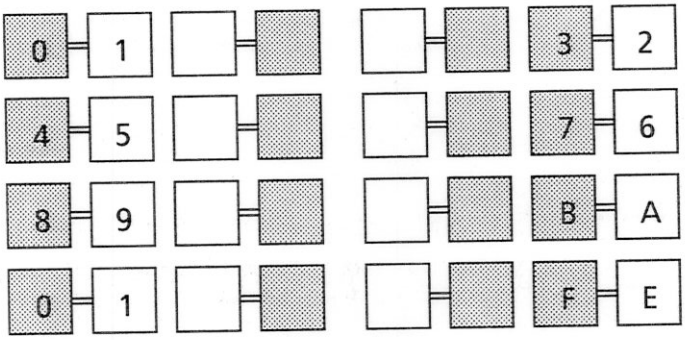
Figure 4.4a: Horizontal-merge (*N = P, P = 4x4*)



Step 0-2 (J = 2)



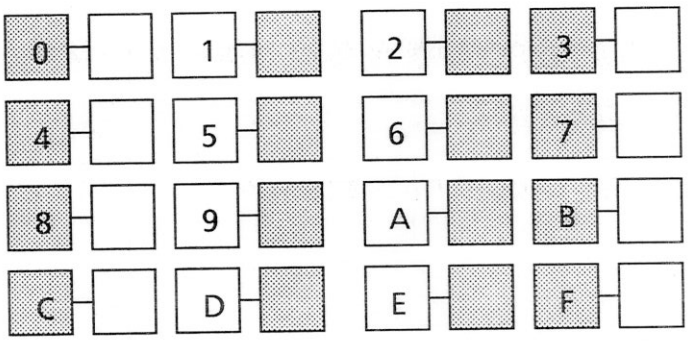
Step 0-3



Step 1-0 (K = 2)

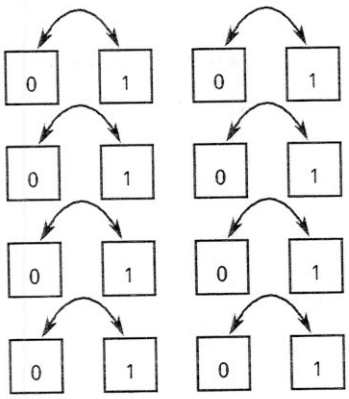
Note:  indicates a comparison between registers.

Figure 4.4b: Horizontal-merge (*N = P, P = 4x4*)

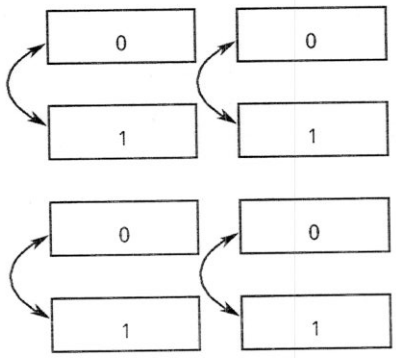


Step 1-1

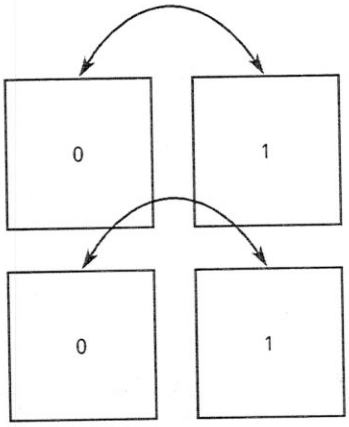
Figure 4.4c: Horizontal-merge (*N* = *P*, *P* = *4x4*)



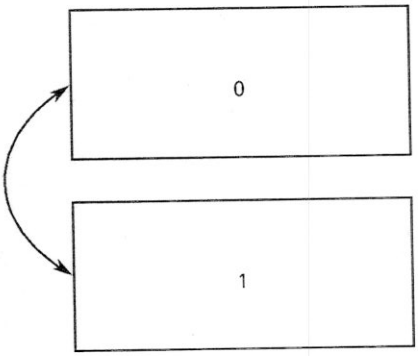
Phase 0-H: Horizontal-merge (J = 1, K = 2, order = id(1))



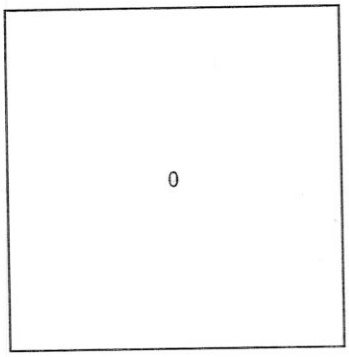
Phase 0-V: Vertical-merge (J = K = 2, order = id(2))



Phase 1-H: Horizontal-merge (J = 2, K = 4, order = id(0))

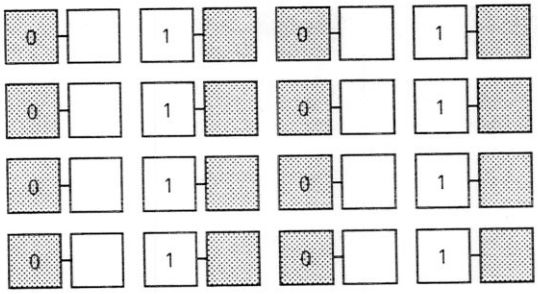


Phase 1-V: Vertical-merge (J = K = 4, order = 0)

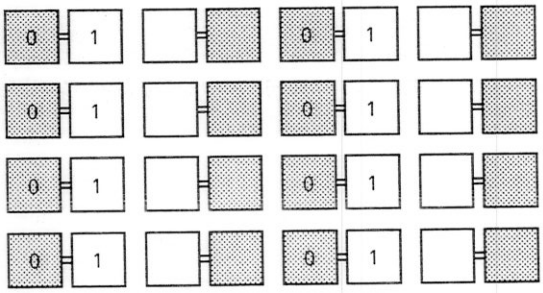


Final output (row-major ordering)

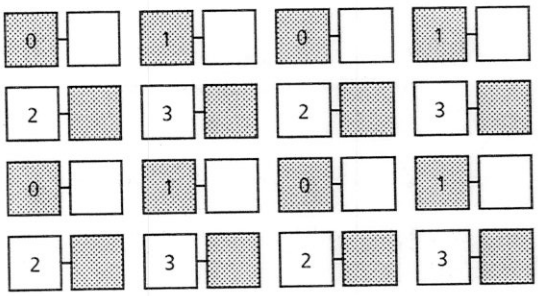
Figure 4.5a: 2D-bitonic-sort (*N = P, P = 4x4*)



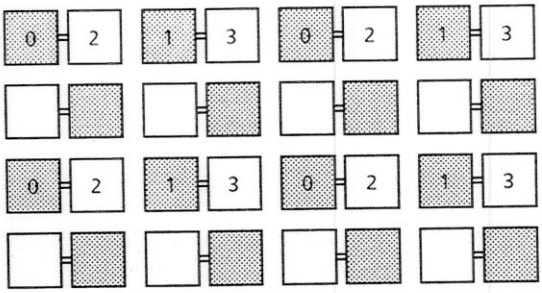
Initial Position:



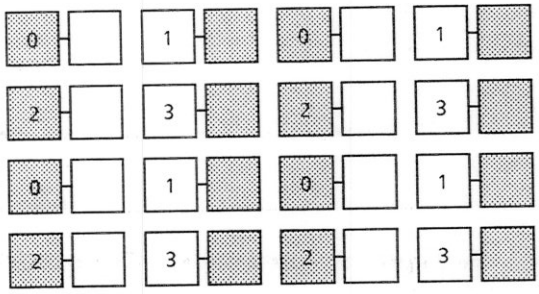
Phase 0-H (Step 0-1):



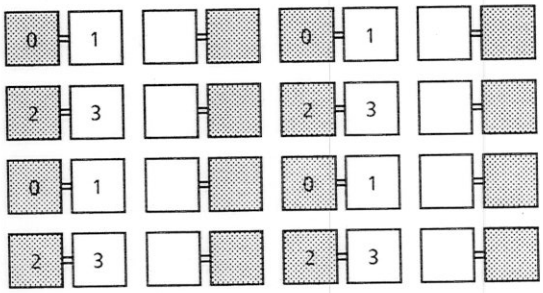
Phase 0-H (Step 0-2):



Phase 0-V (Step 0-0):

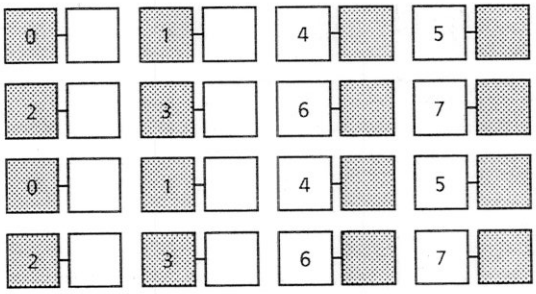


Phase 0-V (Step 0-1):

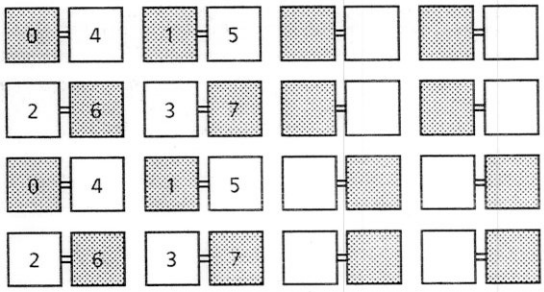


Phase 0-V (Step 1-0):

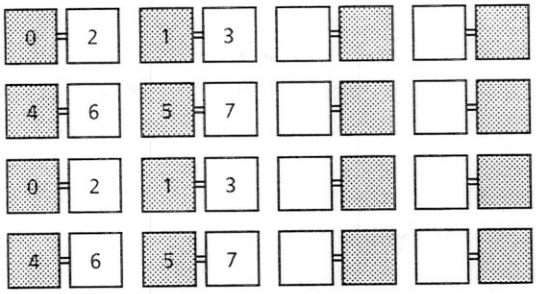
Figure 4.5b: 2D-bitonic-sort ($N = P$, $P = 4x4$)



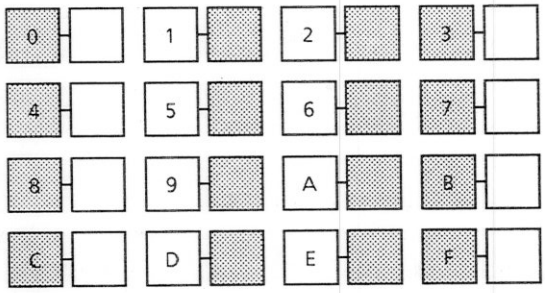
Phase 0-V (Step 1-1):



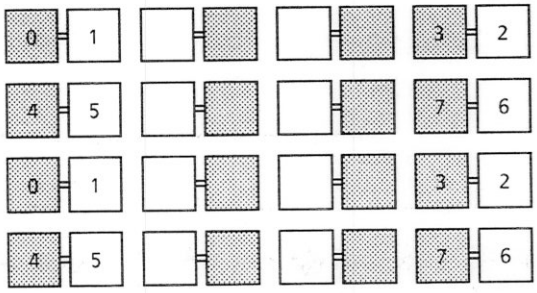
Phase 1-H (Step 0-1):



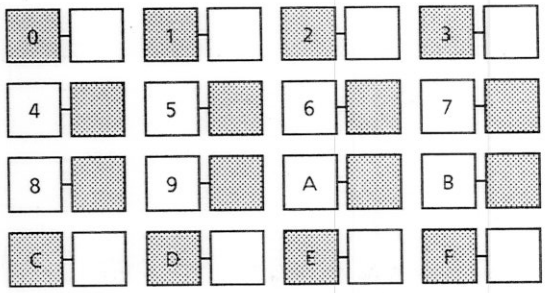
Phase 1-H (Step 0-2):



Phase 1-H (Step 0-3):

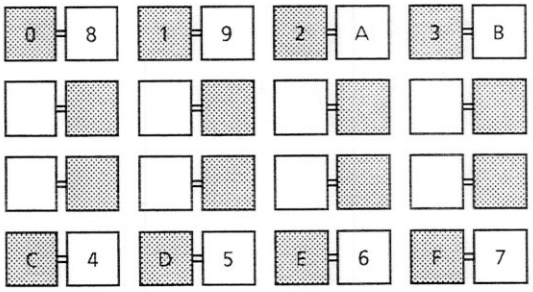


Phase 1-H (Step 1-0):

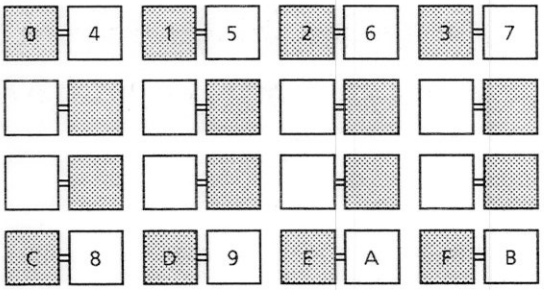


Phase 1-H (Step 1-1):

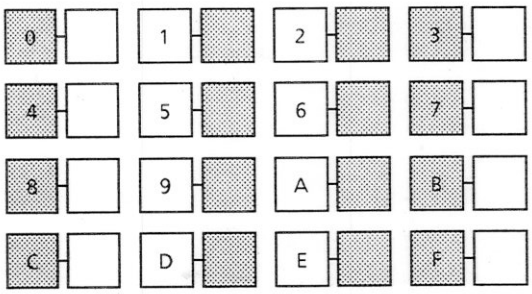
Figure 4.5c: 2D-bitonic-sort ($N = P$, $P = 4x4$)



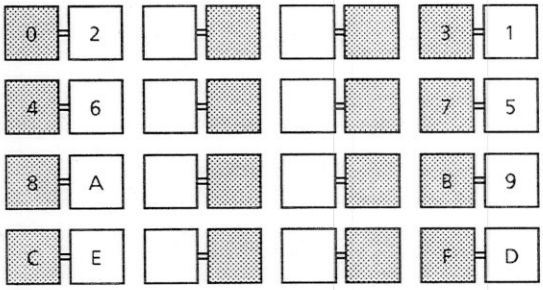
Phase 1-V (Step 0-0):



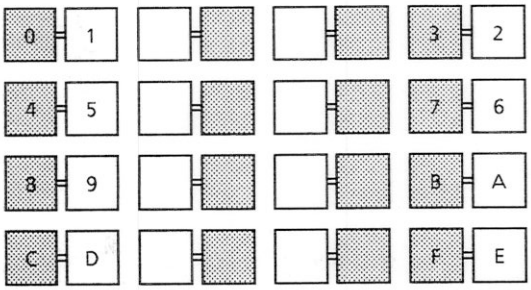
Phase 1-V (Step 0-1):



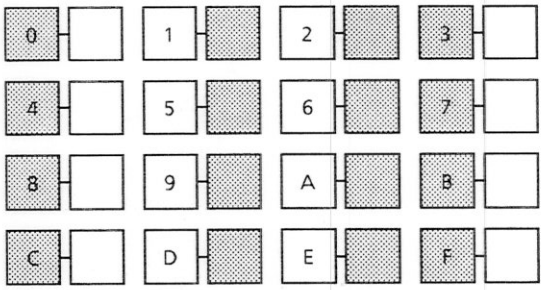
Phase 1-V (Step 0-2):



Phase 1-V (Step 1-0):



Phase 1-V (Step 1-1):



Phase 1-V (Step 1-2):

Figure 4.5d: 2D-bitonic-sort ($N = P$, $P = 4x4$)