AN IMPROVED UPPER BOUND FOR SORTING ON NON-PARTITIONABLE SUPERPOSED PARALLEL BUSES

Bruce W. Arden

Toshio Nakatani

CS-TR-071-87

January 1987

An Improved Upper Bound for Sorting on Non-partitionable 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 presents $O(n\log n)$ and $O(n\log\log n)$ sorting methods on $n\times n$ *non-partitionable* superposed parallel buses. For merging and sorting on n processors connected by a linear bus, an optimal method based on enumeration is developed. It takes $3n/2$ and $2n$ steps, which are $O(\log n)$ and $O(\log^2 n)$ improvements from bitonic merge and sort respectively. For two dimensional sorting on superposed parallel buses, three different methods are considered: bitonic, shear, and reverse sort. Improvements in time complexity of $O(\log n)$ using the first two methods and of $O(\log^2 n/\log\log n)$ using the third method are obtained. Also time complexity measures based on the bus bandwidth are presented. A final discussion on VLSI optimality for the three sorting schemes is included.

January, 1987

An Improved Upper Bound for Sorting on Non-partitionable 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

We have shown that bitonic sort on a $n \times n$ grid of *partitionable* superposed parallel buses (SPB) is optimal in VLSI complexity measures assuming the unit-delay model (Arden and Nakatani[1986d]). That is, its time complexity is $O(n)$. Optimality can be also argued from the fact that any non-degenerate permutation, where every data item must move to a different row and column from the original position, takes at least n cycles (Arden and Nakatani[1986a and 1986c]). We further developed time complexity measures for bitonic sort on a grid of *non-partitionable* SPB, which is $O(n\log^2 n)$ and therefore not optimal. It is interesting to determine whether further improvements of the time performance of sorting on non-partitionable SPB can be achieved.

In this paper, we improve this bound to $O(n\log n)$ and $O(n\log\log n)$, using optimal merging and sorting on a linear bus, and develop a new method based on enumeration. That is, by ranking and permutation, we can merge and sort n data items on a linear bus in $\frac{3n}{2}$ and $2n$ cycles respectively. The lower bound for merging and sorting on a linear bus with n

processors is $O(n)$ since any non-degenerate permutation takes n cycles. Therefore, merging and sorting by enumeration is optimal for a linear bus.

For two dimensional sorting on a $n \times n$ grid of SPB, we consider three different sorting methods: bitonic sort leading to $O(n\log n)$ sorting complexity on SPB; shear sort (Scherson, Sen, and Shamir[1986]) also leading to $O(n\log n)$ sorting complexity; reverse sort (Schnorr and Shamir[1986]) leading to $O(n\log\log n)$ sorting complexity. We develop these time complexity measures for the three different sorting methods with bus bandwidth, B as a parameter. Finally, we discuss VLSI optimality for the three sorting schemes.

This material is organized as follows: In section 2, we summarize the model of computation we use for this analysis. In section 3 and 4, sorting and merging by enumeration on a linear bus is presented. In section 5, 6, and 7, we present three different sorting methods on SPB (bitonic, shear, and reverse sort), and develop time complexity measures based on the bus bandwidth for each case. In section 8, we discuss VLSI optimality for the three sorting schemes.

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 and a counter. The instructions can be either broadcast from the central control unit or distributed to each processor site beforehand. For the former case, there is complete synchrony. For the latter case, synchronization need occur only at the network interface. With this interpretation, the algorithms described

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. We shall discuss the more general case, such as $N = mP$, where each processor has m data items in a separate paper, and focus on the case of $N = P$ here.

To represent time complexity, we use t_r as the time for a processor to send one data item to one of the other processors on a bus. We use t_c as the time for a processor to compare the contents of two registers and to update the counter value.

2.1. Bus Interconnections

The interconnections between the processors are buses. Bus speed is represented by an integer, 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 bus can make one transmission per processor cycle. Clearly, $B \leq P$ and B, P, and N are all integral powers of two.

We consider two interconnection topologies: the linear bus and the two dimensional grid of superposed parallel buses. For both cases, we assume *no bus-partitionability* in this paper. For the linear bus (Figure 2.1), we assume that P processors are connected by a 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 such access within a processor cycle. For $B \geq 2$, a processor can access one of the buses (or alternately, one bus cycle) via an input port and another bus via its output port within one processor cycle. For the two dimensional grid interconnection (Figure 2.2), $P = p \times p$

processors, and every row and column of p processors are connected horizontally and vertically by a bus. For the two dimensional case, we assume $1 \leq B \leq p$ and B, p, and n ($N = n \times n$) are all integral powers of two. Of course, for the one data item per processor assumption, $p = n$.

In an actual implementation, the integral valued 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 the paper.

3. Sorting by Enumeration on a Linear Bus

The technique we use here is called *enumeration sorting* (Knuth[1972], Muller and Preparata[1975], Preparata[1978], and Yasuura, Takagi, and Yajima[1982]). This technique is well suited to a linear bus interconnection. Enumeration sorting is based on comparisons of one key against all the others to find the rank and on the subsequent permutation of data items by the determined ranks. On a linear bus, one-to-all broadcasting is natural and permutation can be done simply by broadcasting every data item once.

The algorithm, *Enumeration-Sort*, for an arbitrary sequence, $X_N = \{x_i \mid i = 0, 1, ..., N-1\}$, on $P = N$ processors on a linear bus is:

```
procedure Enumeration-Sort(X_N, order);
begin
    load processor-id based on order;
    rank:=0;
    for i=0 to N-1 do begin
        if processor-id=i then
            broadcast key.data to the bus;
        load received-key.data from the bus to the input register;
        if {received-key < key} or {received-key = key and i < processor-id}
        then
            rank:=rank+1
    end;
    for i=0 to N-1 do begin
        if rank=i then
            broadcast key.data to the bus;
        if processor-id=i then
            load received-key.data from the bus to the input register;
    end
end;
```

3.1. Time Complexity of Enumeration Sort

Time complexity, $T_{ES}(N)$, of *Enumeration-Sort* for an arbitrary sequence of length N on P ($P=N$) processors on a linear bus of bandwidth B is:

$$T_{ES}(N) = [N + \frac{N}{B}]\cdot t_r + [N]\cdot t_c$$

The rank calculation requires N broadcasts and N comparisons regardless of the bus bandwidth. Permutation requires $\frac{N}{B}$ broadcasts.

4. Merging by Enumeration on a Linear Bus

Merging can also be done using enumeration. Suppose there are two sorted sequences, a and b, of length $\frac{K}{2}$ on K processors on a linear bus. First, we broadcast every data item of the sequence b. Then, every data item of the sequence a now knows its rank in the whole sequence of length K. Second, we broadcast every data item of the sequence a. Then, every data item of the sequence b now knows its rank in the whole sequence of length K. At the same time, every data item of the sequence a is permuted to its destination by its rank. Third, we broadcast every data item of the sequence b. Then, every data item of the sequence b is permuted to its destination by its rank.

The algorithm, *Enumeration-Merge*, for two sorted sequences of length $\frac{K}{2}$, $X_K = \{x_i \mid i = 0,1,...,N-1$ and x_i is sorted for $i = 0,1,..., \frac{K}{2}-1$ and $i = \frac{K}{2}, \frac{K}{2}+1,..., K-1\}$, on K contiguous processors (which are assumed to be on the integral boundary of K for program simplicity) on a linear bus follows. Here, we note that *processor-id-low* represents the least significant log$K$ bits of *processor-id*.

procedure *Enumeration-Merge*(X_K, *order*);

begin

load *processor-id* based on *order*;

rank:=*rank-in-sub-sequence*;

processor-id-low:=*processor-id*[log$N$ $-$log$K$;log$N$ -1];

for $i=0$ to $\frac{K}{2}-1$ **do begin**

if *processor-id-low*$=\frac{K}{2}+i$ **then**

broadcast *key.data* to the bus;

load *received-key.data* from the bus to the input register;

if *processor-id-low*$=i$ and [{*received-key* $<$ *key*} or {*received-key* $=$ *key* and $i <$ *processor-id*}] **then**

rank =*rank* $+1$

end;

for $i=0$ to $K-1$ **do begin**

if *processor-id-low*$=i$ **then**

broadcast *rank.key.data* to the bus;

load *received-key.data* from the bus to the input register;

if *processor-id-low*=*received-rank* **then**

load the final register from the input register;

if *processor-id-low*$\geq\frac{K}{2}$ and [{*received-key* $<$ *key*} or {*received-key* $=$ *key* and $i <$ *processor-id*}] **then**

rank =*rank* $+1$

end

end;

4.1. Time Complexity of Enumeration Merge

Time complexity, $T_{EM}(K)$, of *Enumeration-Merge* for two sorted sequences of length $\frac{K}{2}$ on K contiguous processors on a linear bus of bandwidth B is:

$$T_{EM}(K) = [K + \frac{K}{2B}]\cdot t_r + [K]\cdot t_c$$

The rank calculation requires K broadcasts and K comparisons regardless of the bus bandwidth. Permutation requires $\frac{K}{B}$ broadcasts. However, second half of the broadcast and comparison cycles are overlapped with first half of broadcast cycles for permutation. Therefore, $K + \frac{K}{2B}$ broadcasts and K comparisons are necessary and sufficient.

5. Bitonic Sort on SPB

An optimal adaptation of bitonic sort on SPB requires partitioning buses to progressively sort larger sub-grids independently (Arden and Nakatani[1986d]). With non-partitionable buses, the bitonic approach does not lead to an optimal sort. In this paper, we use *Vertical-Merge* (Arden and Nakatani[1986d]) for two dimensional sorting technique, combined with *Enumeration-Merge* and *Enumeration-Sort* we developed in the previous sections. *Vertical-Merge* algorithm we use here is based on the previously described k-way bitonic sort (Arden and Nakatani[1986b]).

The *Vertical-Merge* algorithm to merge a $k \times n$ rectangular array of data items (k is an integral power of two and $2 \leq k \leq n$) in row-major order

is:

procedure *Vertical-Merge*($X_{k,n}$, *order*);

begin

for each column c ($c = 0, 1, ..., n-1$) in parallel **do**

Enumeration-Merge(X_k, *order*);

for each row r ($r = 0, 1, ..., n-1$) in parallel **do**

Enumeration-Sort(X_n, *order*);

end;

As in the study of the k-way bitonic sort, the column merge is regular-bitonic merge and the row merge is also bitonic merge. *Enumeration-Merge* can simulate regular-bitonic merge (Arden and Nakatani[1986b]) but not bitonic merge. Therefore, we use *Enumeration-Sort* to simulate bitonic-merge.

In the following algorithm, some special notations are used. First, $r[\log n - 1]$ represents the $(\log n - 1)$-th bit of r in binary representation. For example, if $r = 100$ and n=2, then $r[\log n - 1] = 1$. Second, $(0.r)$ represents the concatenation of 0 and r in binary representation. For example, if $r = 100$, then $(0.r) = 0100$.

Accordingly, the *Bitonic-Sort* algorithm to sort a $n \times n$ square array of data items, $X_{n,n}$, in row-major order is:

procedure *Bitonic-Sort*($X_{n,n}$);

begin

for each row r ($r=0,1,...,n-1$) in parallel **do**

order:=r[log$n-1$];

Enumeration-Sort(X_n, *order*);

for i=1 to log$n$ **do begin**

order:=(0.r)[log$n-i$]

for each $2^i \times n$ rectangular subsection in parallel **do**

Vertical-Merge($X_{2^i,n}$, *order*);

end

end;

5.1. Time Complexity of Bitonic Sort

The time complexity, $T_{BS}(n,n)$, of *Bitonic-Sort* for an arbitrary sequence of length $N=n^2$ on $P=p\times p$ ($P=N$) processors on a grid of SPB with the bus bandwidth B is:

$$T_{BS}(n,n) = T_{ES}(n)+\sum_{i=1}^{\log n}\frac{n}{2^i}\cdot T_{EM}(2^i)+\log n\cdot T_{ES}(n)$$

$$= (\log n+1)\cdot T_{ES}(n)+\sum_{i=1}^{\log n}\frac{n}{2^i}\cdot T_{EM}(2^i)$$

$$= (\log n+1)\cdot([n+\frac{n}{B}]\cdot t_r+[n]\cdot t_c)+\log n\cdot([n+\frac{n}{2B}]\cdot t_r+[n]\cdot t_c)$$

$$= [(2+\frac{3}{2B})\cdot n\cdot\log n+(1+\frac{1}{B})\cdot n]\cdot t_r+[2n\cdot\log n+n]\cdot t_c$$

6. Shear Sort on SPB

Shear Sort (Scherson, Sen, and Shamir[1986]) is based on the simple idea that an iteration of both row sort in alternate order and column sort in increasing order decreases the number of "dirty" rows into half, and $\log n + 1$ iterations completely sorts an $n \times n$ square array of data items in snake-like, row-major order. Here, a row is called "dirty" when the row contains zero's and one's in the process of sorting zero's and one's. Conversely, a row is called "clean" when the row contains only zero's or one's.

THe *Shear-Sort* algorithm to sort a $n \times n$ square array of data items, $X_{n,n}$, in snake-like, row-major order is:

procedure *Shear-Sort*($X_{n,n}$);
begin
  for $i=1$ to $\log n + 1$ **do begin**
    for each row r ($r = 0, 1, ..., n-1$) in parallel **do**
      order:=$r[\log n - 1]$;
      Enumeration-Sort(X_n, *order*);
    for each column c ($c = 0, 1, ..., n-1$) in parallel **do**
      order:=0;
      Enumeration-Sort(X_n, *order*);
  end
end;

6.1. Time Complexity of Shear Sort

The time complexity, $T_{SS}(n,n)$, of *Shear-Sort* for an arbitrary sequence of length $N = n^2$ on $P = p \times p$ ($P = N$) processors on a grid of SPB, with the bus bandwidth B, is:

$$T_{SS}(n,n) = (\log n + 1)\cdot 2\cdot T_{ES}(n)$$

$$= [2\cdot(1+\frac{1}{B})\cdot n\cdot(\log n + 1)]\cdot t_r + [2n\cdot(\log n + 1)]\cdot t_c$$

7. Reverse Sort on SPB

Reverse Sort (Schnorr and Shamir[1986]) is based on the clever idea that iteration of three steps (column sort in increasing order, row sort in increasing order, and row rotation of each row by the amount of the bit-reversal of each row address) decreases the number of dirty rows to the order of square root of the initial size. As a result, $\lceil \log\log n \rceil$ iterations sort $n \times n$ square array of data items with exception of at most eight dirty rows, and this number of exceptional rows is independent of n. To complete sorting in snake-like row-major order, we can use shear sort for three iterations.

The *Reverse-Sort* algorithm to sort a $n \times n$ square array of data items, $X_{n,n}$, in row-major order is:

```
procedure Reverse-Sort(X_{n,n});
begin
    for i=1 to ⌈loglogn⌉ do begin
        for each column c (c=0,1,...,n-1) in parallel do
            order:=0;
            Enumeration-Sort(X_n, order);
        for each row r (r=0,1,...,n-1) in parallel do
            order:=processor-id-low+bit-reversal(r) (mod n);
            Enumeration-Sort(X_n, order);
    end;
    for i=1 to 3 do begin
        for each row r (r=0,1,...,n-1) in parallel do
            order:=r[logn-1];
            Enumeration-Sort(X_n, order);
        for each column c (c=0,1,...,n-1) in parallel do
            order:=0;
            Enumeration-Sort(X_n, order);
    end
end;
```

7.1. Time Complexity of Reverse Sort

The time complexity, $T_{RS}(n,n)$, of *Reverse-Sort* for an arbitrary sequence of length $N=n^2$ on $P=p\times p$ ($P=N$) processors on a grid of SPB, with the bus bandwidth B, is:

$$T_{SS}(n,n) = (\lceil \log\log n \rceil)\cdot 2\cdot T_{ES}(n) + 3\cdot 2\cdot T_{ES}(n)$$

$$= [2\cdot(1+\frac{1}{B})\cdot n\cdot(\lceil \log\log n \rceil + 3)]\cdot t_r + [2n\cdot(\lceil \log\log n \rceil + 3)]\cdot t_c$$

8. VLSI Complexity of Sorting

It is ultimately important to relate sorting networks to VLSI implementation and, accordingly, two appropriate complexity measures are considered. We consider conventional AT^2 measures (Thompson[1979]) for point-to-point networks, and also AT^2M^2 measures, which have a special relevance for multi-point networks (Ullman[1984b]).

8.1. AT^2 Measures

Specifically, we are concerned about the chip area, A, and the computation time, T for sorting networks. The basic assumptions we make for 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, a lower bound of $AT^2 = O(P^2\log^2 P)$ can be obtained for sorting P words of $(1+\varepsilon)logP$ bits each ($\varepsilon > 0$).

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. Accordingly, B does not affect the complexity measures. We also assume $B \leq P$. For a linear bus, the time for enumeration sort is $T=O(P)$ and thus $AT^2=O(P^3\log^2 P)$. On the other hand, for a two dimensional grid of SPB, $T=O(\sqrt{P}\log P)$ and $AT^2=O(P^2\log^4 P)$ for bitonic and shear sort and $T=O(\sqrt{P}\log\log P)$ and $AT^2=O(P^2\log^2 P\log\log^2 P)$ for reverse sort. Therefore, all three sorting schemes on SPB are close to optimal. If we allow bus-partitionability, bitonic sort can be performed with optimal performance (Arden and Nakatani[1986d]). However, considering the control structures of both optimal bitonic sort and partitionable buses, the sorting algorithms described here are more practical for real implementations.

8.2. 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 SPB. For this analysis, B should be interpreted as bus speed.

For a linear bus, the time for enumeration sort is $T=O(P)$ and $AT^2M^2=O(P^5\log^2 P)$. On the other hand, for a two dimensional grid of SPB, $T=O(P\log P)$ and $AT^2M^2=O(P^4\log^4 P)$ for bitonic and shear sort, and

$T=O(P\text{loglog}P)$ and $AT^2M^2=O(P^4\log^2P\text{loglog}^2P)$ for reverse sort. That is, sorting schemes we presented in this paper are not optimal for AT^2M^2 measures because of linear comparisons required for enumeration sort. In order to get better performance with high-bandwidth buses, we should use bitonic merge and sort for a linear bus and also as subroutines for sorting on SPB. The reasoning is as follows:

Time complexity, $T_{BM}(K)$, of *Bitonic-Merge* for two sorted sequences of length $\frac{K}{2}$ on K contiguous processors on a linear bus of bandwidth B can be described by (Arden and Nakatani[1986d]):

$$T_{BM}(K) = [\frac{K}{2B}\cdot\log 2K\,]\cdot t_r + [\log K\,]\cdot t_c$$

That is, for given B, if $K\leq 4^B$ then bitonic merge is faster than enumeration merge. Conversely, if $K>4^B$, then enumeration merge is faster than bitonic merge. That is, for *Optimal-Merge*, two different merge schemes should be used according to the bus bandwidth B and the size K.

With respect to the given bus bandwidth B, time complexity, $T_{OM}(K)$, of *Optimal-Merge* for two sorted sequences of length $\frac{K}{2}$ on K contiguous processors on a linear bus is given by:

$$T_{OM}(K) = \begin{cases} [\frac{K}{2B}\cdot\log 2K\,]\cdot t_r + [\log K\,]\cdot t_c & \text{if } K\leq 4^B \\ [K+\frac{K}{2B}]\cdot t_r + [K\,]\cdot t_c & \text{if } K>4^B \end{cases}$$

Similarly, the time complexity, $T_{BS}(N)$, of *Bitonic-Sort* for an arbitrary sequence of length N on $P=N$ processors on a linear bus of bandwidth B (Arden and Nakatani[1986d]) is:

$$T_{BS}(N) = [\frac{N}{2B}\cdot(\frac{1}{2}\log^2 N + \frac{1}{2}\log N + 1)]\cdot t_r + [\frac{1}{2}\log^2 N + \frac{1}{2}\log N]\cdot t_c$$

That is, for given B, if $N \leq N^*$ (where N^* is the ceiling of the positive root of the equation $\log^2 N + \log N - 2(2B+1) = 0$. The discriminant is positive so there will be a positive real root but non-integral in general.) then bitonic sort is faster than enumeration sort. On the other hand, if $N > N^*$, then enumeration sort is faster than bitonic sort.

With respect to the given bus bandwidth B, time complexity, $T_{OS}(N)$, of *Optimal-Sort* for an arbitrary sequence of length N on $P = N$ processors on a linear bus of bandwidth B is:

$$T_{OS}(N) = \begin{cases} [\frac{N}{2B}\cdot(\frac{1}{2}\log^2 N + \frac{1}{2}\log N + 1)]\cdot t_r + [\frac{1}{2}\log^2 N + \frac{1}{2}\log N]\cdot t_c & \text{if } N \leq N^* \\ [N + \frac{N}{B}]\cdot t_r + [N]\cdot t_c & \text{if } N > N^* \end{cases}$$

Therefore, with a high-bandwidth bus (particularly in the linear case of $M = B = P$ and $M = B = \sqrt{P}$ for the two dimensional SPB), we use bitonic merge and sort instead of the enumeration method. For a linear bus, the time for bitonic sort is $T = O(log^2 P)$ and $AT^2M^2 = O(P^3 \log^6 P)$. On the other hand, for a two dimensional grid of SPB, $T = O(\log^3 P)$ and $AT^2M^2 = O(P^2 \log^8 P)$ using bitonic and shear sort, and $T = O(\log^2 P \log\log P)$ and $AT^2M^2 = O(P^2 \log^6 P \log\log^2 P)$ with the reverse sort. That is, the sorting schemes for SPB presented in this paper, with bitonic merge and sort as subroutines, are close to optimal for AT^2M^2 measures.

9. Concluding Remarks

Three different sorting methods on a grid of non-partitionable SPB have been described. We developed an optimal merging and sorting scheme for a linear bus based on enumeration. And then, using this algorithm as a

subroutine, time complexity measures for three different sorting schemes with bus bandwidth as a parameter were obtained. These schemes are within logarithmic factors of optimality by AT^2 measures. However, they are not optimal in AT^2M^2 measures because of sequential comparisons in the enumeration algorithm. Therefore, for a high-bandwidth bus, we suggested the use of bitonic merge and sort as subroutines for sorting on SPB. With this strategy, the three sorting schemes are nearly optimal, that is, within logarithmic factors in AT^2M^2 measures.

Two dimensional sorting was first studied by Gale and Karp[1972]. Many investigators have studied it in terms of sorting on mesh-connected computers (Orcutt[1976], Thompson and Kung[1977], Nassimi and Sahni[1979], Kumar and Hirschberg[1983], Lang, Schimmler, Schmeck, and Schroder[1985], Scherson, Sen, and Shamir[1986], and Schnorr and Shamir[1986]). They achieved optimal performance within constant and logarithmic factors.

Earlier, we have shown that a two dimensional grid of SPB has better time performance for bitonic sorting with partitionable buses (Arden and Nakatani[1986d]). This paper shows an improvement in the performance of sorting on SPB with non-partitionable buses by factors of $O(\log n)$ and $O(\log^2 n / \log\log n)$. Although it might be possible to implement the $O(\log N)$ sorting scheme (Ajtai, Komlos, and Szemeredi[1983], Bilardi and Preparata[1985], and Leighton[1985]) on SPB, it is not likely to be practical due to huge constant factors involved. The sorting algorithms we presented here are practical because of simple control structures and simple bus interconnections. And the constant factors in time performance are relatively small.

An interesting open problem is to find the lower bound of sorting on a two dimensional grid of non-partitionable SPB. It is easy to show sorting takes at least $O(n)$ on a $n \times n$ grid of SPB, but it is not known whether $O(n \log\log n)$ is the lower bound or not. As a related problem, we have shown that selection problem takes $O(n)$ on a $n \times n$ grid of non-partitionable SPB and further more this is optimal (Arden and Nakatani[1987]).

References

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

Arden, B. and T. Nakatani [1986a]. "Permutations on superposed parallel buses," Technical Report CS-TR-024-86, Department of Computer Science, Princeton University.

Arden, B. and T. Nakatani [1986b]. "K-way bitonic sort," Technical Report CS-TR-040-86, Department of Computer Science, Princeton University.

Arden, B. and T. Nakatani [1986c]. "Optimal permutations on superposed parallel buses," Technical Report CS-TR-060-86, Department of Computer Science, Princeton University.

Arden, B. and T. Nakatani [1986d]. "Bitonic sorting on superposed parallel buses," Technical Report CS-TR-063-86, Department of Computer Science, Princeton University.

Arden, B. and T. Nakatani [1987]. "Optimal selection on superposed parallel buses," Technical Report CS-TR-???-87, 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 [1985]. "A minimum VLSI network for $O(\log n)$ time performance," *IEEE Trans. on Computers* C-34:4, pp. 336-343.

Gale D. and R. M. Karp [1972]. "A phenomenon in the theory of sorting," *J. of Computer and System Sciences,* 6, pp.103-115.

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

Kumar M. and D. S. Hirschberg [1983]. "An efficient implementation of Batcher's odd-even merge algorithm and its application in parallel sorting schemes," *IEEE Trans. on Computers* C-32:3, pp. 254-264.

Lang H., M. Schimmler, H. Schmeck, and H Schroder [1985]. "Systolic sorting on a mesh-connected network," *IEEE Trans. on Computers* C-34:7, pp. 652-658.

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

Muller D. E. and F. P. Preparata [1975]. "Bounds to complexities of networks for sorting and for switching," *J. ACM* 22:4, pp. 195-201.

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

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

Preparata F. P. [1978]. "New parallel-sorting schemes," *IEEE Trans. on Computers* C-27:7, pp. 669-673.

Scherson, I. D., S. Sen, and A. Shamir [1986]. "Shear sort: a true two-dimensional sorting technique for VLSI networks," Technical Report 86-20, Department of ECE, University of California, Santa Barbara.

Schnorr C.P. and A. Shamir [1986]. "An optimal sorting algorithm for mesh connected computers," *Proc. of 18th Annual Symposium on Theory of Computing,* pp. 255-263.

Thompson, C.D. and 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.

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.

Yasuura H., H. Takagi, and S. Yajima [1982]. "The parallel enumeration sorting scheme for VLSI," *IEEE Trans. on Computers* C-31:12, pp. 1192-1201.

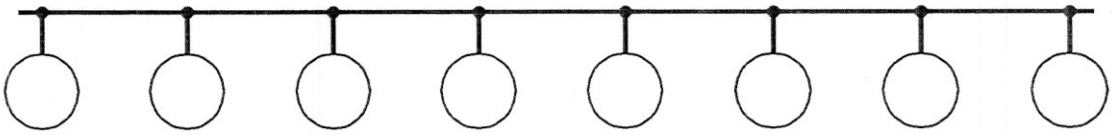

Figure 2.1: A linear bus ($N = 8$)

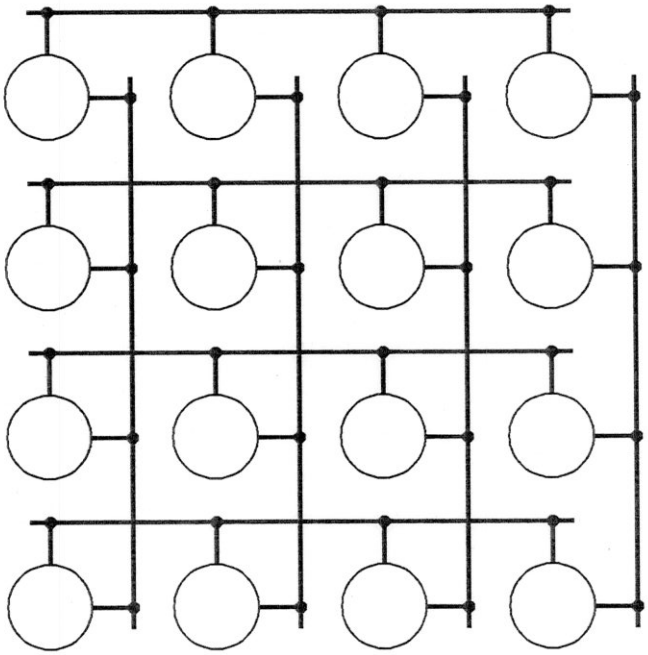

Figure 2.2: A square grid of superposed parallel buses ($N = 4x4$)