Bitonic Sort on Ultracomputers

by
Lambert Meertens†

Ultracomputer Research Laboratory
Courant Institute of Mathematical Sciences
715 Broadway, 10th Floor
New York, NY 10003

Ultracomputer Note #1
March, 1979

† Matematisch Centrum, Amsterdam

ABSTRACT

Batcher's *bitonic sort* (cf. Knuth, v. III, pp. 232 ff) is a sorting network, capable of sorting n inputs in $\Theta((\log n)^2)$ stages. When adapted to conventional computers, it gives rise to an algorithm that runs in time $\Theta(n(\log n)^2)$. The method can also be adapted to *ultracomputers* (Schwartz [1979]) to exploit their high degree of parallelism. The resulting algorithm will take time $\Theta((\log N)^2)$ for ultracomputers of ''size'' N. The implicit constant factor is low, so that even for moderate values of N the ultracomputer architecture performs faster than the $\Theta(N \log N)$ time conventional architecture can achieve. The purpose of this note is to describe the adapted algorithm. After some preliminaries a first version of the algorithm is given whose correctness is easily shown. Next, this algorithm is transformed to make it suitable for an ultracomputer.

1. Introduction

Batcher's *bitonic sort* (cf. Knuth, v. III, pp. 232 ff) is a sorting network, capable of sorting n inputs in $\Theta((\log n)^2)$ stages. When adapted to conventional computers, it gives rise to an algorithm that runs in time $\Theta(n(\log n)^2)$. The method can also be adapted to *ultracomputers* (Schwartz [1979]) to exploit their high degree of parallelism. The resulting algorithm will take time $\Theta((\log N)^2)$ for ultracomputers of ''size'' N. The implicit constant factor is low, so that even for moderate values of N the ultracomputer architecture performs faster than the $\Theta(N \log N)$ time conventional architecture can achieve. The purpose of this note is to describe the adapted algorithm. After some preliminaries a first version of the algorithm is given whose correctness is easily shown. Next, this algorithm is transformed to make it suitable for an ultracomputer.

Definition A sequence $s_0,\dots,s_{n-1}$ of elements from a totally ordered set is *bitonic* if there exist i and j, $0 \le i \le j \le n-1$, such that either

$$s_i \le s_{i+1} \le \dots \le s_j \text{ and } s_j \ge s_{j+1} \ge \dots \ge s_{n-1} \ge s_0 \ge s_1 \ge \dots \ge s_i,$$

or

$$s_i \ge s_{i+1} \ge \dots \ge s_j \text{ and } s_j \le s_{j+1} \le \dots \le s_{n-1} \le s_0 \le s_1 \le \dots \le s_i.$$

(If the sequence is made into a cycle by connecting the rear back to the front, this means that both ways of going from s_i to s_j give an ordered "run.") Note that a sequence of length ≤ 3 is always bitonic.

Bitonic sort hinges on the following.

Lemma 1. *Let* $s_0,\dots,s_{2n-1}$ *be bitonic. For* $i = 0,\dots,n-1$, *interchange* s_i *and* s_{n+1} *if* $s_{n+1} < s_i$. *Then for the resulting sequence, both* $s_0,\dots,s_{n-1}$ *and* $s_n,\dots,s_{2n-1}$ *are bitonic. Moreover, each of the elements* $s_0,\dots,s_{n-1}$ *is less than or equal to each of the elements* $s_n,\dots,s_{2n-1}$.

Proof: See Batcher (1968) or Stone (1971). (The proofs given are rather informal. A more formal proof would be elementary but not very enlightening; it would proceed by distinguishing a number of cases.)

The elements to be sorted are stored in an array a[0:N-1], where $N=2^D$ for some integer D. The indices of the array will often be written as bitstrings (binary numbers) $b_{D-1}b_{D-2}\dots b_0$, corresponding to the integer $b_{D-1}2^{D-1}+...+b_0 2^0$. The notation $b_{H:L}$ denotes the substring $b_H b_{H-1}...b_L$. (Note that the subscript runs from high to low; in order to minimize confusion, capital letters will be used for such subscripts.)

Definition. Ω stands for a mapping from the set of substrings $b_{H:L}$ into the set of order relations $\leq$ and $\geq$, satisfying $\Omega(b_{H:H+1})$ is $\leq$ and $\Omega(b_{H:L+1}0) \neq \Omega(b_{H:L+1}1)$. One possible solution is given by

$$\Omega\,(b_{H:L}) \text{ is } \leq \text{ if } b_H \oplus b_{H-1} \oplus ... \oplus b_L = 0,$$
$$\Omega\,(b_{H:L}) \text{ is } \geq \text{ if } b_H \oplus b_{H-1} \oplus ... \oplus b_L = 1.$$

The symbol $\oplus$ stands for the ''logical sum'' or ''exclusive or'', so the summation determines the parity of $b_{H:L}$. A simpler solution is given by: $\Omega(b_{H:L+1}0)$ is $\leq$, $\Omega(b_{H:L+1}1)$ is $\geq$. (By convention, $\Omega(b_{H:H+1})$ is $\leq$ in either case.)

The assertions of the correctness proof will use three predicates, defined below. Let the array a be (conceptually) divided into 2^{D-P} segments of 2^P elements each. The indices of the elements of a given segment are precisely those which have a common initial bitstring $b_{D-1:P}$.

Definition. Ordered (P) stands for:

within each segment the elements are sorted in $\Omega(b_{D-1:P})$-order.

Definition. Bitonic (P) stands for:

each segment forms a bitonic sequence.

Let now each segment be subdivided into 2^{P-Q} subsegments, or *boxes*, of 2^Q elements each. If the elements of a segment were sorted in some order, each element would end up in its *destination box* according to that order.

Definition. In_Boxes (P,Q) stands for:

within each segment the elements are (already) in their destination boxes according to $\Omega(b_{D-1:P})$ -order.

Lemma 2. *If* $0 \leq P \leq D$, *then*

(1) In_Boxes (P,P);

(2) *if* In_Boxes (P,0), *then* Ordered (P)

(3) *for* $P \geq 1$, *if* Ordered (P-1), *then* Bitonic(P).

Proof: As to (a), In_Boxes (P,P) means that the boxes coincide with the segments. As there is only one destination box per segment, each element of a segment must be in its destination box. As to (b), if In_Boxes (P,O), the boxes have one element. So if within a segment the elements are in their destination box, they must be in place and each segment is sorted. (Actually, In_Boxes (P,O) is equivalent to Ordered (P).) As to (c), if Ordered (P-1), then for each segment of length 2^P the lower half and the upper half are both sorted in $\Omega(b_{D-1:P-1})$- order. For the lower half $b_{P-1} = 1$, so the upper half is sorted in the reverse order of the order of the lower half. The whole segment is then bitonic.

Definition. ich(H:P,Q), $0 \le Q \le P \le H+1 \le D$, stands for the following action:

for all b, interchange a[b with b_Q=0] and a[b with b_Q=1] if they are not in $\Omega(b_{H:P})$- order.

Lemma 3. *If* $0 \le Q \le P \le D$, *then*

{Bitonic (Q+1)&In_Boxes(P,Q+1)} ich(D-1:P,Q) {Bitonic(Q)&In_Boxes(P,Q)}.

Proof: This lemma is a generalization of Lemma 1 for sequences whose length is a power of two. (Lemma 1 is obtained from Lemma 3 by taking P=D and Q=D -1.) The generalization follows by applying Lemma 1 to each (bitonic) box of length 2^{Q+1} in a segment of length 2^P. The boxes are then "refined" by splitting each box into two halves (each of which receives again a bitonic sequence), and its elements are divided over the two new boxes of length 2^Q according to Ω(D-1:P)- order. Since the elements were already in their destination boxes of length 2^{Q+1}, they now reach their destination box of length 2^Q.

First version of the algorithm:

```
{In_Boxes (0,0)
{Ordered (0)}
for P = 1,2,...,D do
        {Ordered (P-1)}
        {Bitonic (P) & In_Boxes (P,P)}
         for Q = P - 1, P - 2,...,0 do
            {Bitonic (Q+1) & In_Boxes (P,Q+1)}
            ich (D-1:P,Q)
            {Bitonic (Q) & In_Boxes (P,Q)}
         end for Q
        {In_Boxes (P,0)}
        {Ordered (P)}
end for P
{Ordered (D)}.
```

Correctness proof: Each of the verification conditions is either trivially satisfied or is an immediate consequence of Lemmas 2 and 3. The final assertion Ordered (D) asserts that the whole array is sorted in $\le$- order.

If the operation ich(D-1:P,Q) could be realized in time $\Theta(1)$, the algorithm would take time $\Theta(D^2)$. If the elements of the array a are stored in consecutive processors of an ultracomputer, it is, however, not possible to compare two arbitrary elements immediately, since not all processors are directly connected. Consecutive processors *are* connected, so operations of the form ich(H:P,O) operate in time $\Theta(1)$. Other connections are the *shuffle* lines, connecting each processor $b_{D-1:0}$ to the processor $\sigma(b_{D-1:0}) = b_0 b_{D-1:1}$. Through this connection, the following *parallel* assignments take time $\Theta(1)$:

shuffle: for all b, a[b] :=a[σ(b)];
unshuffle: for all b, a[σ(b)] :=a[b].

The two operations permute a and are each other's inverse.

Let $shuffle^Q$ stand for the null action if Q= 0, and for shuffle $^{Q-1}$; shuffle if Q ≥ 1. So $shuffle^Q$ stands for:

$$\text{for all b, a[b] : =a}[\sigma^Q(b)].$$

Let unshuffle ^{Q} be defined similarly.

Lemma 4. ich (D-1:P,Q), *where* 0≤Q≤P≤D, *is equivalent to*

unshuffle^Q; ich (D-Q-1:P-Q,0); shuffle^Q.

Proof: The operation ich(D-1:P,Q) stands for:

for all b, interchange a[b with b_Q=0] and a[b with b_Q=1] if they are not in $\Omega(b_{D-1:P})$-order.

Using the assignment rule, this is seen to be equivalent to

for all b, a[σ^Q(b)] := a[b] (or unshuffle ^{Q});
for all b, interchange a[σ^Q(b) with b_Q0]
and a[σ^Q(b) with b_Q=1]
if they are not in $\Omega(b_{D-1:P})$ - order;
for all b, a[b] :=a[σ^Q(b)] (or unshuffle ^{Q}).

Substituting in the middle part $\sigma^{-Q}(b')$ for b, using $b_R=\sigma^{-Q}(b')^R=b'_{R-Q}$ for R, we obtain

for all b', interchange a[b' with b'_0 =0]
and a(b' with b'_0 = 1]
if they are not in $\Omega(b_{D-Q-1:P-Q})$-order.

This is exactly the meaning of ich(D-Q-1:P-Q,0).

Using Lemma 4, the algorithm may be transformed to:

```
for P = 1,2,...,D do
  for Q = P-1,P-2,...,0 do
        unshuffle^Q;
        ich (D-Q-1:P-Q,0);
        shuffle ^Q
   end for Q
end for P.
```

This intermediate version would require time $\theta(D^3)$.

Lemma 5. *For* K≥0

LOOP $_K$≡ **for** Q=K,K-1,...,0 **do** unshuffle^Q; S(Q); shuffle^Q **end**.
where S(Q) is any statement depending on Q, *is equivalent to*
unshuffle $^{K+1}$;LOOP'$_K$, *where*
LOOP'$_K$≡ **for** Q = K,K-1,...,0 do shuffle; S(Q) **end.**

Proof: By induction on K. $LOOP_0$ and unshuffle; $LOOP'_0$ reduce to an obvious equivalence. For larger K, we see that $LOOP_K$ is equivalent to

$$\text{unshuffle}^K;\ S(K);\ \text{shuffle}^K;\ \text{LOOP}_{K-1}$$

by moving the first execution of the loop body outside. By the inductive hypothesis, this is equivalent to

$$\text{unshuffle}^K;\ S(K);\ \text{shuffle}^K;\ \text{unshuffle}^K;\ \text{LOOP'}_{K-1}$$

which again is equivalent to

$$\text{unshuffle}^{K+1};\ \text{shuffle};\ S(K);\ \text{LOOP'}_{K-1}.$$

Moving shuffle; S(K) inside the loop, we obtain

$$\text{unshuffle}^{K+1};\ \text{LOOP'}_K.$$

By this lemma, we finally obtain

Algorithm for bitonic sort on ultracomputers

```
for P = 1,2,...,D do
   unshuffle^P;
  for Q = P-1,P-2,...,0 do
      shuffle;
      ich (D-Q-1:P-Q,0)
  end for Q
end for P.
```

This algorithm clearly takes time $\theta(D^2) = \theta((\log N)^2)$.

Remark. The idea of using shuffles to implement bitonic sort is described in Stone [1971].

2. References

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

J.T.Schwartz [1979] *Ultracomputers*. Preprint, Computer Science Department Courant Institute of Mathematical Sciences, New York University, New York.

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