Integer Sorting on a Mesh-connected Array of Processors

Danny Krizanc
Department of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

Schnorr and Shamir and independently Kunde, have shown that sorting $N = n^2$ inputs into snake-like ordering on a $n \times n$ mesh requires $3n - o(n)$ steps. Using a less restrictive, more realistic model we show that the sorting $N = n^2$ integers in the range $[1 \dots N]$ can be performed in $2n + o(n)$ steps on a $n \times n$ mesh.

1 Introduction

The problem of sorting is of fundamental importance in computing. In sequential models of computation the difference between sorting general inputs and sorting integers in a fixed range has been well established. In a model where only comparisons between inputs are allowed a lower bound of $\Omega(N \log N)$ for sorting N inputs is well known. However if we know the inputs fall in a polynomial size range, an algorithm with complexity $O(N)$ is possible. (See [AHU].) In this paper we study the effect of limiting the inputs to be integers in the range $[1 \dots N]$ in the case of sorting $N = n^2$ inputs on a $n \times n$ mesh-connected array of processors.

The mesh-connected array has been the object of a great deal of theoretical study as well as being the basis for a number of proposed and implemented parallel computers (see [AS],[Ba], and [Fl]). While its diameter is large in comparison to other well-studied networks (e.g., hypercube, butterfly, shuffle-exchange networks), the simplicity and regularity of its interconnection pattern make it ideal for VLSI implementation. Recent work by Dally

[Da] suggests that high diameter networks such as the mesh may provide a more efficient communication medium for VLSI-based parallel computers. Furthermore a large number of efficient algorithms have been designed to run on this architecture.

The problem of sorting on a mesh has a long history starting with Thompson and Kung [TK] and leading eventually to an optimal algorithm by Schnorr and Shamir [SS] for sorting general inputs on a synchronous MIMD mesh. Their algorithm sorts $N = n^2$ inputs into snake-like order on a $n \times n$ mesh in $3n + o(n)$ steps. They also provide a lower bound, independently discovered by Kunde [Ku], for this problem of $3n - o(n)$ steps. The model for the lower bound puts no limit on the power of the processors but forces the output to be a permutation of the input, eliminating the possibility that the outputs be copies of the inputs. The idea of their lower bound is to show that for any algorithm we can fool some input into taking a long route to its final destination.

In this paper we consider a more realistic model in which a processor is allowed to perform only simple operations and communicate a limited amount of information to its neighbours during a single step. In this model we provide an algorithm for sorting $N = n^2$ integers in the range $[1 \dots N]$ on a $n \times n$ mesh in $2n + o(n)$ steps. This would seem to contradict the lower bound mentioned above. However, as in the sequential case, we are able to achieve this bound by taking advantage of our knowledge of the range inputs. The idea behind our algorithm is to gather together enough information about the inputs into the center portion of the mesh so that when this information is broadcast the processors can determine their correct output value.

The rest of the paper is divided into three sections. In the next section we give a more detailed description of our model and a number of facts which will be useful for proving our results. Section 3 develops our main result in simple stages. The last section contains conclusions and suggests lines of enquiry for future work. Throughout the paper logarithms are to the base 2 and expressions take integral values whenever this is convenient and otherwise insubstantial.

2 Preliminaries

The $n \times n$ mesh-connected array has $N = n^2$ processors arranged in a two dimensional grid (without wrap-around edges). The processors reside at the vertices of the grid and each has bidirectional communication links to its

neighbours defined by the grid edges. The processors run in synchronous MIMD fashion. On each step they are able to send and receive at most one packet from each of their neighbours and perform a constant number of simple operations (e.g., addition, comparison, concatenation) on the contents of these packets, where the size of the packets is at most $O(\log N)$ bits. In all the algorithms below the processors are never required to store more than $O(k + \log N)$ bits where k is the size (in bits) of their initial input.

Initially each processor contains an integer in some fixed range. The goal of the computation is to sort the integers into snake-like order (or some other fixed order). That is, at the end of the computation the i th processor in the snake-like ordering of the mesh should contain the i th smallest integer in the input. The results below are stated for the snake-like ordering but they are easily seen to hold for many other orderings. In corollary 1 we consider the case where each processor initially contains $r > 1$ integers in the given range and at the end of the computation the integers are sorted with r per processor in the obvious manner.

A subroutine which will be very useful in the algorithms below is the parallel prefix operation. Let $\oplus$ denote a binary associative operation defined on pairs of elements from some domain. Assume each processor i , $1 \leq i \leq N$, numbered in snake-like ordering, contains an input x_i from the domain. The parallel prefix operation computes each of the partial sums $y_i = x_1 \oplus \cdots \oplus x_i$, $1 \leq i \leq N$, and stores y_i in processor i . Assume $\oplus$ can be computed using a constant number of operations in our model. Then

Lemma 1 *The parallel prefix of $N = n^2$ inputs can be computed in $O(n)$ steps on a $n \times n$ mesh.*

Proof: In $n-1$ steps compute the parallel prefix of each row, computing from left to right on odd numbered rows and from right to left on even numbered rows, i.e. sum along the snake. Route the sum of each row to the opposite end of the row in $n-1$ steps. Compute the parallel prefix of the first and last column downward in a further $n-1$ steps. Finally in $n-1$ steps distribute in every row (again along the snake) the sum of the previous rows. $\square$

A second operation we will require below is block broadcasting. Imagine an $n \times n$ mesh partitioned into $\frac{n^2}{m^2} m \times m$ blocks of processors. For the block broadcasting operation the processors in a designated block each initially contain an $O(\log N)$ bit input. The result of the operation is each of the blocks contain a copy of the inputs of the designated block with corresponding processors containing the corresponding inputs.

Lemma 2 *The block broadcast of an $m \times m$ block of an $n \times n$ mesh may be performed in $2(n - m)$ steps.*

Proof: Broadcast the block along its row of blocks and then each block in the row broadcasts along its column. The worst case is broadcasting a corner block, in which case each phase requires $n - m$ steps to complete. $\square$

The following facts have been established elsewhere (see [SS] and [LMT]).

Fact 1 *General inputs (containing $O(\log N)$ bits) may be sorted in $O(n)$ steps on a $n \times n$ mesh using constant size queues.*

Fact 2 *A partial permutation of packets (containing $O(\log N)$ bits), i.e., at most one packet originating at and destined to a processor, may be routed in $O(n)$ steps on a $n \times n$ mesh using constant size queues.*

3 Sorting Integers on a Mesh

As a first simple step towards proving our main result we show how to sort N bits into snake-like order in $2n - 2$ steps.

Theorem 1 *$N = n^2$ bits (i.e., inputs chosen from $\{0, 1\}$) can be sorted into snake-like order in $2n - 2$ steps on a $n \times n$ mesh.*

Proof: The processors on the left half of the mesh send their input to the right and those on the right half of the mesh send their input to the left and the central processors sum the inputs. After $\frac{n}{2} - 1$ steps the processors in column $\frac{n}{2}$ have computed the number of 1's in the left half of their row and the processors in column $\frac{n}{2} + 1$ have computed the number of 1's in the right half of their row. We repeat this process along the central columns. After another $\frac{n}{2} - 1$ steps the four central processors contain the total number of 1's in their quadrant of the mesh. In two steps (exchange along rows and then columns) they are each able to calculate the total number of 1's in the entire mesh equal to t . Using lemma 2 with $m = 1$ they are able to broadcast t to each processor in their quadrant in $2(\frac{n}{2} - 1)$ steps. Each processor compares $N - t$ to their index in the sorted order. If their index is strictly greater than $N - t$ they output 1. Otherwise they output 0. The total number of steps required is $2n - 2$. $\square$

The above theorem already indicates the difference in strength of the Schnorr-Shamir sorting model and the model considered here since even this

task would require $3n - o(n)$ steps in the Schnorr-Shamir model. Note that the obvious diameter lower bound of $2n - 2$ shows that the result is optimal.

The next step in our program is to extend theorem 1 to sorting integers in the range $[1 \dots N^\epsilon]$, $\epsilon < 1$. This is done by partitioning the processors into square blocks and by performing the above algorithm on the blocks.

Theorem 2 *$N = n^2$ integers in the range $[1 \dots N^\epsilon]$, $\epsilon < 1$, can be sorted into snake-like order in $2n + o(n)$ steps on a $n \times n$ mesh.*

Proof: Partition the mesh into $m \times m$ blocks where $m = n^\epsilon$. Sort the inputs in each block into snake-like order using fact 1. Each processor checks the value received by the next processor in the ordering against the value it received. If the values differ the processor is called a leader. The last processor in the ordering is always a leader. Perform a parallel prefix using the operation maximum and a processor's input being its index if it is a leader and 0 otherwise. From this computation the leaders can calculate how many other processors in the block contain the same value as they do. Using fact 2 the leaders send this number to the processor whose index is the leaders value. At this point processor i contains the number of processors in the block with input i , for $1 \leq i \leq N^\epsilon = n^{2\epsilon}$. (If a processor receives no packet during the routing it outputs 0 at this point.)

Analogous to the proof of theorem 1 the four central blocks compute the total number of i -valued inputs in their quadrant, for $1 \leq i \leq N^\epsilon$ in $2(\frac{n}{2} - m)$ steps, share these totals in $2m$ steps, and then broadcast the complete mesh totals using lemma 2 in a further $2(\frac{n}{2} - m)$ steps.

Now each block contains full information about the sorted order of the input. Using parallel prefix with the operation addition and the processors input being its total, processor i can calculate the position of the first and last occurrence of the value i in the total sorted order. From this it can calculate the first occurrence of the value i in its own block, if any. Using fact 2 it can send a packet containing the value to this first processor called a leader. Note that the processors having the same output are contiguous in snake-like order inside the block. A straight-forward application of parallel prefix allows the leaders to broadcast their value to each of the processors between them and the next leader. The inputs in this case are ordered pairs (x, y) where x equals 1 if the processor is a leader and 0 otherwise and y equals the processors value if it is a leader and 0 otherwise. The (non-commutative) operation is defined by $(x_1, y_1) \oplus (x_2, y_2) = (x_1, b)$ where $b = y_2$ if $x_2 = 1$ and $b = x_1$ otherwise.

Each of the computations done inside the blocks require $O(m)$ steps. The summing of the blocks in the central processors and the broadcasting of the result requires $2(n - m)$ steps. Therefore the total number of steps required is $2n + O(m)$. $\square$

The algorithm is optimal to within an additive factor of $O(n^\epsilon)$. By having one processor simulate the work of a number of processors the theorem may be extended to sort more than N numbers in a smaller range in asymptotically the same time bound.

Corollary 1 $N^{1+\beta} = n^{2+2\beta}$ integers (N^β per processor) in the range $[1 \dots N^\epsilon]$, $\epsilon + 2\beta < 1$, can be sorted into snake-like order in $2n + o(n)$ steps on a $n \times n$ mesh. $\square$

Again this result is optimal to within an additive factor (in this case $O(n^{\epsilon+2\beta})$). It provides a more drastic example of the difference between the model used here and the Schnorr-Shamir model. In the straightforward extension of the Schnorr-Shamir model to sorting more than N numbers a lower bound on the above result would be $\Omega(n^{1+2\beta})$. (To see this, consider the permutation which sends the numbers on the left half of the mesh to the right half. There are only n edges joining the halves and each of $\Theta(n^{2+2\beta})$ numbers must be transferred across these edges, one number per edge per time step.)

Theorem 2 does not immediately extend to the case of sorting integers in the range $[1 \dots N]$. To achieve our main result we make use of a representation of N numbers in the given range which requires only $2N$ bits. This may be stored over $O(\frac{N}{\log N})$ processors with $O(\log N)$ bits per processor.

Theorem 3 $N = n^2$ integers in the range $[1 \dots N]$ can be sorted into snake-like order in $2n + o(n)$ steps on a $n \times n$ mesh.

Proof: A sorted list of k integers in the range $[1 \dots N]$ may be represented by a string of $k + N$ bits. N 0's are used as markers to separate the integers into N buckets, one for each possible value. Each of the k integers is represented by a single 1 between the appropriate markers. (See [LLMPW] for more on compact representations of sorted integers.) For $k = O(N)$ this representation may be distributed over $\frac{N}{\log N}$ processors with $O(\log N)$ bits per processor. The i th processor contains the bits $(i - 1)\frac{k+N}{N} \log N + 1$ to $i\frac{k+N}{N} \log N$ and the number of 0's occurring in the string before that point.

This auxiliary value will simplify the explanation below by enabling the processors to interpret correctly the portion of the string they hold. Note that given a distribution of the string the processors can calculate this value using a parallel prefix operation.

We are now ready to present an overview of our sorting algorithm.

1. Partition the mesh into $m \times m$ blocks, $m = \frac{n}{\sqrt{\log N}}$. Each block contains $\frac{N}{\log N}$ processors.
2. Sort the blocks into snake-like order using fact 1.
3. Convert the result of 2 to the compact representation described above.
4. Along each row of blocks repeat the following for $i = 1, \dots, \log(\sqrt{\log N}) - 1$:
 - (a) Move block j over block $j + 2^{i-1}$, if $j < \frac{N}{2}$, over block $j - 2^{i-1}$, if $j > \frac{N}{2} + 1$.
 - (b) Merge the compact representation of the pairs of overlapping blocks to form a single compact representation.

At this point the central blocks in each row contain a representation of all the integers in their half of the row of blocks.

5. Repeat 4 along the central columns. At this point the four central blocks contain a representation of all the integers in their quadrant.
6. Using two more exchanges and merges the four central blocks contain a compact representation of all the integers in sorted order.
7. Using fact 2 broadcast the result of 6 to all the blocks.
8. Inside each block decode the compact representation so that each processor may calculate its final output.

The total number of routing steps performed in steps 4-7 is $2(n - m)$. The total number of mergings that take place in steps 4-6 is $O(\log \log N)$. To complete the proof of the theorem we must show that step 3, the mergings performed in steps 4 - 6 and step 8 may all be performed in $O(m)$ steps.

Claim 1: Step 3 may be performed in $O(m)$ steps.

Proof: As in the proof of theorem 2 each processor at the end of a block of similar values is elected leader of the block. Using a parallel prefix operation the leaders can calculate (a) the number of processors with their same input, (b) the number of leaders below them in the ordering, and (c) the value of the next leader in the ordering. From this information the leaders can calculate a portion of the compact representation consisting of the 1's in their bucket followed by the 0's up to the next bucket. At this point the leaders contain the complete compact representation if their results were to be combined. Note that a leader may have calculated a large portion (larger than $O(\log N)$ bits) of the compact representation but it can store this portion using two $\log N$ bit integers. The leaders are also able to calculate the beginning position of their portion of the string. The leaders break their portion into $t = \log N + 1$ bit chunks corresponding to pieces that processors will eventually contain. Some of the chunks thus created will contain fewer than t bits. Using a parallel prefix operation these small chunks can be accumulated into single t bit piece at the next highest leader. In the worst case this partitioning will result in four types of pieces: A single piece consisting of t 0's and 1's, a sequence of pieces containing t 1's, a single piece of length t consisting of some 1's followed by some 0's, and a sequence of pieces containing t 0's. Using four routing phases the leaders can route the pieces to the processors which are to contain them along with the number of 0's appearing below them in the ordering. In the case of the sequences of pieces containing t 1's or t 0's the leader routes the piece to the first processor in the sequence along with the number of pieces in the sequence. A further parallel prefix operation is then used to distribute this information to the remaining processors. $\square$

Claim 2: The merging of compact representations performed in steps 4-6 may be performed in $O(m)$ steps.

Proof: Given compact representations of two sorted lists of integers the 1's in the strings represent distinct integers but the 0's occur twice, once in each string. To perform a merging of the representations we must align the 0's and remove one of these copies. Each processor contains a portion of the compact representation of two lists. Call the lists A and B and mark the pieces as to which list they represent. Sort the pieces by the number of 0's occurring in the list before the piece. This sorting which results in two pieces per processor in snake-like order can be performed in $O(m)$ steps using an obvious extension of fact 1. Processors declare themselves leaders if they contain a piece from the A list and the next piece in the sorted order is a piece from the B list. Using a parallel prefix operation the leaders distribute their A piece to each of

the processors containing B pieces up to the next A piece. These processors simply align their 0's with the 0's in the leaders piece and merge the 1's into a single representation. At this point the processors contain a correct complete representation of the string though some processors will contain more bits than others (by at most a factor of two). Using the same technique as in claim 1 these bits can be evenly distributed among the processors to form the required compact representation described above. $\square$

Claim 3: Step 8 may be performed in $O(m)$ steps.

Proof: Using a parallel prefix operation the processors calculate the number of 1's occurring in the output before their piece of the compact representation. Each processors then checks if the piece of the compact representation it has represents the output values of processors in its block. If so, in a single routing phase, the relevant pieces are routed to the first (of at most $O(\log N)$) processor in the block which is represented by the piece. In a further $O(\log N)$ steps the first processors in each contiguous segment of processors represented by a piece distributes the relevant information to its neighbors. $\square$

Claims 1-3 along with the observations above complete the proof of the theorem. $\square$

4 Conclusions

The results presented above are analogous to what is known about sequential sorting in that it is shown that taking advantage of our knowledge of the range of the inputs allows us to improve the efficiency of sorting. In this case the small range is used to reduce the amount of storage required to represent the inputs. The results also indicate a difference between the lower bound model of Schnorr and Shamir and the more realistic MIMD model above.

The techniques of theorem 3 do not obviously extend to the case of sorting integers in the range $[1 \dots N^c]$, for $c > 1$. This leads us to ask whether there is a $2n + o(n)$ algorithm for this problem or if a $3n$ lower bound may be shown in this case. It may also be of interest to investigate the problem of sorting integers in a fixed range on other networks such as the hypercube, butterfly and shuffle-exchange.

References

- [AHU] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [AS] W. C. ATHAS AND C. L. SEITZ, *Multicomputers: Message-Passing Concurrent Computers*, *Computer*, **21**, 8, 1988, pp. 9 - 24.
- [Ba] K. BATCHER, *Design of a Massively Parallel Processor*, *IEEE Trans. Comp.*, **29** (1980), pp. 836-840.
- [Da] W. DALLY, *Wire-Efficient VLSI Multiprocessor Communication Networks*, *Advanced Research in VLSI*, MIT Press, 1987, pp. 391-415.
- [Fl] C. M. FLAIG, *VLSI Mesh Routing Systems*, CalTech Computer Science Technical Report, 5241:TR:87, 1987.
- [Ku] M. KUNDE, *Lower Bounds for Sorting on Mesh-connected Architectures*, *Acta Informatica* **24**, 1987, pp. 121-130.
- [LLMPW] T. LEIGHTON, C. E. LEISERSON, B. MAGGS, S. PLOTKIN AND J. WEIN, *Lecture Notes for Theory of Parallel and VLSI Computation*, MIT/LCS/Research Seminar Series 1, 1988.
- [LMT] T. LEIGHTON, F. MAKEDON AND I. TOLLIS, *A $2n - 2$ Step Algorithm for Routing in an $n \times n$ Array with Constant Size Queues*, *Symp. on Parallel Algorithms and Architectures*, 1989, pp. 328-335.
- [SS] C. P. SCHNORR AND A. SHAMIR, *An Optimal Sorting Algorithm for Mesh Connected Computers*, *Proc. of 18th ACM Symp. on Theory of Computing*, 1986, pp. 255-263.
- [TK] C. D. THOMPSON AND H. T. KUNG, *Sorting on a Mesh-Connected Parallel Computer*, *CACM* **20** (1977), pp. 263-270.