JOURNAL OF COMPLEXITY 6, 365–378 (1990)

Space and Time Complexities of Balanced Sorting on Processor Arrays*

FERNG-CHING LIN AND JIANN-CHERNG SHIEH

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

Received January 3, 1989

A processor is balanced in carrying out a computation if its computing time equals its I/O time. When the computation bandwidth of a processor is increased, like when multiple processors are incorporated to form an array, the critical question is to what degree the processor's memory must be enlarged in order to alleviate the I/O bottleneck to keep the computation balanced. In this paper, for the sorting problem, we present two balanced algorithms on linearly connected and mesh-connected processor arrays, respectively, and show that they reach the derived lower bounds of memory sizes. We also verify that the time complexities of the algorithms are optimal under their respective hardware constraints. © 1990 Academic Press, Inc.

1. INTRODUCTION

With the advance of technology, the computation bandwidth of a processor can be greatly increased by incorporating multiple processors and operating them in parallel. However, due to hardware limitations, the I/O bandwidth of the processor cannot be increased as easily. As a result, the computation speed of such a processor is usually confined by its I/O speed. A general approach to alleviate this problem is to reside more local memory space in the processor in order to reduce its I/O requirement (Siewiorek, *et al.*, 1982). Moreover, in real applications, we often encounter a situation in which the problem size is far larger than the processor's memory size. Under this circumstance, the computation must be decom-

* This research was partially supported by National Science Council of the Republic of China under Contract NSC77-0408-E0002-09.

0885-064X/90 $3.00

posed into subcomputations to be executed one at a time. This requires a considerable amount of I/O operations to store and retrieve intermediate results. Thus, the time spent in I/O may dominates the total execution time of the computation.

Kung (1985) proposed an information model to characterize a processor by its computation bandwidth, I/O bandwidth, and memory size. A processor is defined to be balanced for a computation if its computing time equals its I/O time. Consider a processor that can perform balanced computation to solve a given problem. Suppose the computation bandwidth of the processor is increased by a factor of α relative to its I/O bandwidth. Then in performing the same computation the processor will be unbalanced, i.e., the processor will have to wait for slower I/O and the I/O bottleneck occurs. This can be diminished by enlarging the processor's memory so that sufficient data can be prepared in time for operation during the computation. On the basis of this concept, Kung (1985) obtained some lower bound results on the memory size a processor must have in order to rebalance various computations as the processor's computation bandwidth is increased. He also claimed that we can view a collection of α identical processors as a new "larger" processor that has a computation bandwidth α times bigger. The derived memory sizes are then evenly distributed to the processors in the larger processor. However, when implementing a computation on a processor array, besides the effects of computation bandwidth, I/O bandiwidth, and memory size, we must also take into account the influence of the communication pattern among the processors. Communication may play a dominant role in computation performance when one is solving a problem like sorting.

In this paper, we investigate balanced sorting on linearly connected and mesh-connected processor arrays. The sorting model we apply here is that keys can be used for comparisons but not for manipulations. Consider a linearly connected array of α processors, each with computation bandwidth C, I/O bandwidth I, and $C/I = \log m$, where $m > \alpha$. The array's computation bandwidth is αC and its I/O bandwidth is still I. According to Kung's argument, in order to perform balanced sorting, the whole array needs $\Omega(m^\alpha)$ memory size. But we show that, under the same condition, the processor array actually requires $\Theta(\alpha m^\alpha)$ size of memory, a higher and exact bound. Next we consider a mesh-connected array of α^2 processors, each with computation bandwidth C, I/O bandwidth I, and $C/I = \log m/2$, $m > \alpha^2$. The computation bandwidth is now $\alpha^2 C$ and the I/O bandwidth becomes αI. Similarly, by Kung's argument, the processor array requires $\Omega(m^{\alpha/2})$ size of memory to perform balanced sorting. However, we show that the processor array actually requires a higher $\Theta(\alpha^2 m^{\alpha/2})$ size of memory. Our proposed balanced sorting algorithms, which are used to provide memory size upper bounds, are indeed time

optimal and exhibit asymptotic full speedups with α and α^2 processors, respectively.

2. Balanced Computation

Now we formally present the information model and the concept of balanced computation introduced by Kung (1985). As illustrated in Fig. 1, a processor can be characterized by three factors:

C: the computation bandwidth, which is the number of computing operations the processor can deliver per time unit,

I: the I/O bandwidth, which is the maximum of the number of input operations and the number of output operations the processor can have per time unit, and

M: the size of the processor's memory, in terms of the number of words.

Let C_{comp} (cost for computing) denote the number of computing operations and C_{io} (cost for I/O) denote the number of I/O operations needed for a computation. A processor is balanced in carrying out the computation if its computing time equals its I/O time; i.e.,

$$\frac{C_{\text{comp}}}{C} = \frac{C_{\text{io}}}{I} \quad \text{or} \quad \frac{C}{I} = \frac{C_{\text{comp}}}{C_{\text{io}}}. \tag{1}$$

Suppose the ratio C/I is increased by a factor of α. By (1), the processor will be rebalanced if the ratio $C_{\text{comp}}/C_{\text{io}}$ is increased by the same factor. In general, this can be achieved by enlarging the size of the processor's memory. Here we take sorting as the example to show how big the new memory size must be.

Consider the problem of sorting N data by comparisons only. The problem can be solved in two phases. Phase 1 generates N/M sorted lists

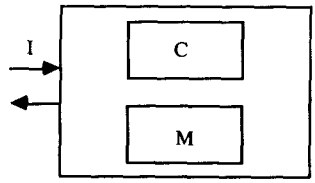

FIG. 1. Information model of a processor.

of M data each. Phase 2 merges these sorted lists by using an M-way merge algorithm (Aho *et al.*, 1976). In phase 1, producing a sorted list of M data requires $\Theta(M \log M)$ comparisons and $\Theta(M)$ I/O operations. In phase 2, we maintain a heap of M data which are the first elements of the current M sorted lists. This heap can be implemented in a memory of size $\Theta(M)$. For each I/O operation to the heap, there are $\Theta(\log M)$ comparisons to be performed to reconstruct the heap. Therefore, for both phases, we have

$$\frac{C_{\text{comp}}}{C_{\text{io}}} = \Theta(\log M). \tag{2}$$

Assume the processor is balanced for this computation; that is, $C/I = \Theta(\log M)$. Now if the ratio C/I is increased by a factor of α, then by (1), we must also increase $C_{\text{comp}}/C_{\text{io}}$ by the same factor to rebalance the computation. In other words,

$$\frac{\alpha C}{I} = \frac{\alpha C_{\text{comp}}}{C_{\text{io}}} = \Theta(\log M'), \tag{3}$$

where M' is the required new memory size. Thus, by (2) and (3), we have $M' = \Theta(M^{\alpha})$.

It was proved in (Song, 1981) that, for sorting, $C_{io} = \Omega(N \log N/\log M)$. Therefore, for the new memory size M', the new I/O requirement $C'_{\text{io}} = \Omega(N \log N/\log M')$. Let C'_{comp} be the new computation cost. Then $C'_{\text{comp}}/C'_{\text{io}} = O(C'_{\text{comp}} \cdot \log M'/N \log N)$. Since the computation is rebalanced, $C'_{\text{comp}}/C'_{\text{io}} = \alpha C/I = \Theta(\alpha \log M)$. This implies that $\alpha \log M = O(C'_{\text{comp}} \cdot \log M'/N \log N)$, and hence $\log M' = \Omega(\alpha \log M \cdot N \log N/C'_{\text{comp}})$. Suppose we want to minimize C'_{io} $(= C'_{\text{comp}} \cdot I/\alpha C)$ so as to minimize the total executing time; since $C'_{\text{comp}} = \Omega(N \log N)$ (Knuth, 1973), $\log M' = \Omega(\alpha \log M)$ or $M' = \Omega(M^{\alpha})$. Therefore, when C/I is increasing by a factor of α, $M' = \Theta(M^{\alpha})$ is the minimum memory size to keep the computation balanced and the I/O requirement minimized.

A processor array is constructed by connecting a number of processors in some interconnection pattern. Kung (1985) viewed a collection of α processors as a new larger processor that has a computation bandwidth α times bigger. With this viewpoint, the above results about a single processor were directly applied to such a larger processor. As we see later, when some specific interconnection patterns are considered, the memory sizes used for balanced sorting turn out to be higher than expected. This is proved to be necessary under the communication restrictions of the array structures.

3. Balanced Sorting Algorithms

In recent years, there have been many parallel algorithms proposed for sorting on linearly connected or mesh-connected arrays of processors (Chen and Nussbaum, 1985; Lang *et al.*, 1985; Miranker *et al.*, 1983; Nassimi and Sahni, 1979; Owens and Ja'Ja', 1985; Thompson and Kung, 1977). If we carefully probe these designs, however, we find that there are some problems incurred:

(1) It was commonly assumed that the computation bandwidth of a processor is equal to its I/O bandwidth, and the I/O bottleneck problem is ignored.

(2) It was taken for granted that the processor array always has adequate space to hold all the data involved in the computation.

(3) It was often assumed that the number of processors in the array is proportional to the problem size.

In this section, we present algorithms for balanced sorting on linearly connected and mesh-connected processor arrays under practical hardware conditions; namely, there is a bandwidth ratio between I/O and computation, and memory size and processor number are not arbitrarily large.

3.1 *Balanced Sorting on a Linearly Connected Processor Array*

Suppose we have a linearly connected array of three identical processors, as depicted in Fig. 2. Each processor has a computation bandwidth C and an I/O bandwidth I, and $C/I = \log m$, where m is a positive constant. On this array, we again do the sorting in two phases. Phase 1 consists of $N/(3m^3)$ rounds, each generating three sorted lists of m^3 data.

Initially, we preload three groups of m^3 data into the local memories MD_{31}, MD_{21}, and MD_{11}, respectively. When processor P_i sorts its m^3 data in MD_{i1}, $i = 1, 2, 3$, $3m^3 \log m$ comparisons are performed. We can

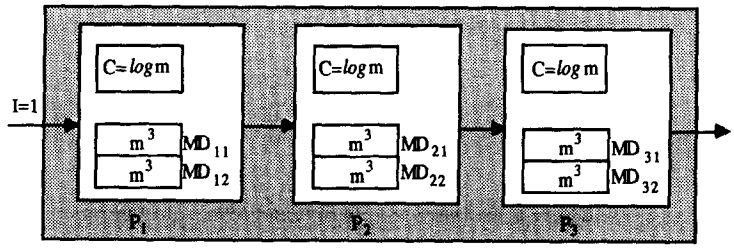

FIG. 2. Linearly connected array of three processors.

simultaneously input $3m^3\log m/\log m = 3m^3$ data into MD_{32}, MD_{22}, MD_{12}. That is, at the end of this round there are m^3 data in each MD_{i2}, $i = 1, 2, 3$. In the next round, each P_i sorts the m^3 data in MD_{i2}, and the next m^3 input data will be loaded into MD_{i1}. In this time period, we can also output the three sorted lists in MD_{i1}, $i = 1, 2, 3$, requiring $3m^3$ I/O time. The rest operations of phase 1 can be deduced accordingly. At the end of this phase, we have N/m^3 sorted lists.

In phase 2, we repeatedly apply the m^3-way merge algorithm in the processors to merge the sorted lists until the final result is obtained. For each m^3-way merge in P_i, $i = 1, 2, 3$, we maintain a heap of m^3 elements. Initially in each MD_{i1}, there are m^3 elements of the first elements of the current m^3 sorted lists. These $3m^3$ data are loaded into the processors during the last round of phase 1. Then in each P_i, we establish a heap of m^3 data. Since it costs $m^3\log m^3/\log m = 3m^3$ computing time, we can simultaneously load $3m^3$ data of the second elements of the current three sets of m^3 sorted lists into MD_{32}, MD_{22}, and MD_{12}. After P_i outputs the top element of the heap in MD_{i1}, it takes a specific element in MD_{i2} to replenish MD_{i1}. During the $\log m^3/\log m = 3$ computing time of reheaping, the output data are dispatched to notify the host to supply three definite elements as fillers to furnish MD_{i2}, $i = 1, 2, 3$. The rest of phase 2 can be continued accordingly until we finally get the desired sorted list of N data.

It is clear that the computing time and the I/O time of the above computation are equalized and the total size of local memories used is $3 \cdot (2m^3)$. On the basis of the same idea, the general balanced sorting algorithm on a linearly connected processor array of arbitrary length can be derived.

Suppose we have a linearly connected array of α processors. The computation bandwidth of the processor array becomes αC but its I/O bandwidth is still I. Phase 1 is completed in $N/\alpha m^\alpha$ rounds, each producing α sorted lists of m^α data. In phase 2, we repeatedly apply the m^α-way merge algorithm in the processors to merge the sorted lists.

LEMMA 3.1. *The execution time of the proposed algorithm for balanced sorting on the linearly connected array of α processors is $O(N \log N/\log m^\alpha)$.*

Proof. In phase 1, the array takes in N data and produces N/m^α sorted lists in N I/O time. Let us count the merging of sorted lists of $m^{k\alpha}$ data into sorted lists of $m^{(k+1)\alpha}$ data, $1 \le k < \log N/\log m^\alpha$, as one iteration. The number of iterations for merging is $O(\log N/\log m^\alpha)$, and each iteration costs $O(N)$ I/O time. Thus, phase 2 needs $O(N(\log N/\log m^\alpha))$ I/O time. Therefore the total execution time of the computation is $O(N(\log N/\log m^\alpha) + N) = O(N \log N/\log m^\alpha)$. Q.E.D.

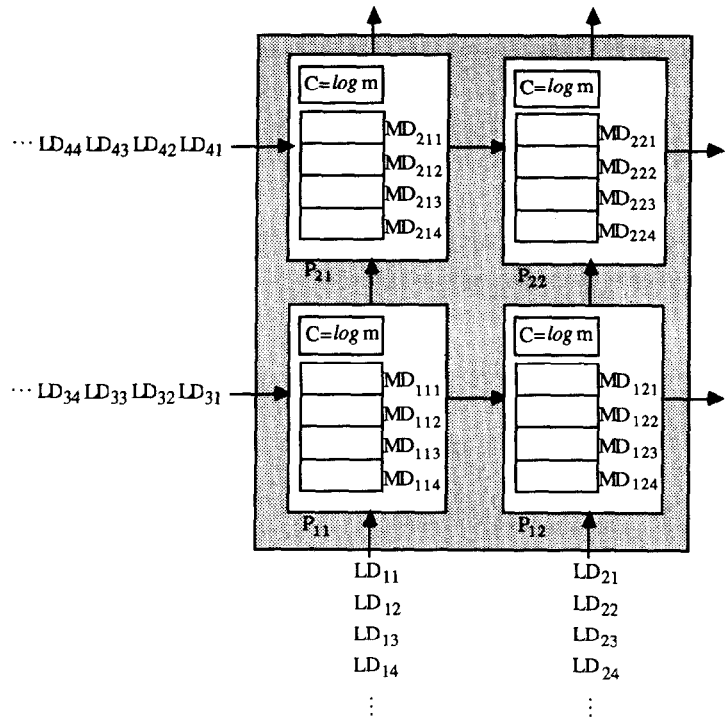

FIG. 3. Mesh-connected array of four processors.

LEMMA 3.2. *The memory size used in the proposed algorithm for balanced sorting on the linearly connected processor array of α processors is $O(\alpha m^{\alpha})$.*

Proof. For the m^{α}-way merge in a processor, we maintain a heap of m^{α} elements, therefore the total size of local memories used in the array is $\alpha \cdot (2m^{\alpha})$. Q.E.D.

3.2. *Balanced Sorting on a Mesh-Connected Processor Array*

Next we consider a mesh-connected array of four processors, P_{11}, P_{12}, P_{21}, P_{22}, each with computation bandwidth C and I/O bandwidth I, and $C/I = \log m/2$. In fact, we may assume that $C = \log m$ and $I = 2$. The sorting is also done in two phases. Phase 1 is executed in $N/(4 \cdot 2 \cdot m)$ rounds, each generating $4 \cdot 2$ sorted lists of m data. That is, each processor generates two sorted lists of m data in each round. The operations of the processors in the array can be described with the help of Fig. 3.

We first individually preload m data into MD_{ij1} and MD_{ij2}, $i, j = 1, 2$. Now each P_{ij} starts to sort its data in MD_{ij1}. Because it requires $m \log m$

comparisons, the processor will take $m \log m/\log m = m$ computing time to accomplish this task. We simultaneously load in four data groups LD_{11}, LD_{21}, LD_{31}, LD_{41} of m data each to MD_{214}, MD_{224}, MD_{123}, MD_{223}, respectively. In the following m I/O time, P_{ij} sorts its data in MD_{ij2}, and in the mean time, the next four data groups LD_{12}, LD_{22}, LD_{32}, LD_{42} are sent to MD_{114}, MD_{124}, MD_{113}, MD_{213}, respectively. In the next m computing time, P_{ij} sorts its data in MD_{ij3}. During this time period, we output four of the sorted lists which were generated before. The sorted lists in MD_{212} and MD_{222} are output through the two upper ports and the sorted lists in MD_{121} and MD_{221} are output through the two right ports. Meanwhile, the current input data LD_{13}, LD_{23}, LD_{33}, LD_{43} are loaded into MD_{212}, MD_{222}, and MD_{121}, MD_{221}, respectively.

Next, each P_{ij} begins to sort its data in MD_{ij4}. We can simunltaneously output the sorted lists in MD_{111} and MD_{211} through the two right ports and output the sorted lists in MD_{112} and MD_{122} throught the two upper ports. Certainly, we still keep inputting data. This time the input data LD_{14}, LD_{24}, LD_{34}, LD_{44} are loaded into MD_{112}, MD_{122}, MD_{111}, MD_{211}, respectively. The rest of phase 1 can be deduced accordingly. When this phase is finished, we will have N/m sorted lists of m data each.

In phase 2, we iteratively merge the sorted lists until we get the final result. In each P_{ij}, we apply the m-way merge algorithm on both MD_{ij1} and MD_{ij3}. Initially, there are m elements of the first elements of the current m sorted lists in MD_{ij1} and also in MD_{ij3}, $i, j = 1, 2$. These $4 \cdot (2 \cdot m)$ data can be loaded into the processors during the last round of phase 1. Then we establish two heaps of m data in each processor. Since it takes $2m \log m/\log m = 2m$ computing time, we can simultaneously load $4 \cdot (2m)$ data of the next elements of current $4 \cdot 2$ sets of m sorted lists into MD_{ij2} and MD_{ij4}. These data should be sent to the processors that their parents, the first elements, stay in. After each P_{ij} outputs the top elements of heaps in MD_{ij1} and MD_{ij3}, we take definite elements in MD_{ij2} and MD_{ij4} to replenish MD_{ij1} and MD_{ij3}. The output data which are generated in MD_{ij1} are dispatched out by using the two right ports to notify the host to supply four specific elements to furnish MD_{ij2}. Similarly, the output data which are generated in MD_{ij3} are sent out by using the two upper ports to notify the host to supply four other specific elements to furnish MD_{ij4}. Since it needs $2 \log m/\log m = 2$ computing time to reheap MD_{ij1} and MD_{ij3}, the work of reheaping and supplying fillers can be concurrently performed. The rest of phase 2 can be continued accordingly until we acquire a sorted list of N data.

It should be clear that the computing time and the I/O time are equalized in the above algorithm. The total size of local memories used is $4 \cdot (4m^{2/2})$. The algorithm can be generalized. Suppose we have a mesh-connected array of α^2 processors. The computation bandwidth and I/O

bandwidth of this processor array are $\alpha^2 \log m$ and 2α, respectively. Phase 1 is executed in $N/(2\alpha^2 m^{\alpha/2})$ rounds, each produces $2 \cdot \alpha^2$ sorted lists of $m^{\alpha/2}$ data. In phase 2, each processor P_{ij} applies the $m^{\alpha/2}$-way merge algorithm on both MD_{ij1} and MD_{ij3} to iteratively merge the sorted lists until the final result is obtained.

LEMMA 3.3. *The execution time of the proposed algorithm for balanced sorting on the mesh-connected processor array is* $O(N \log N/(2\alpha \log m^{\alpha/2}))$.

Proof. Since phase 1 takes in N data by using 2α ports and produces $N/m^{\alpha/2}$ sorted lists, it costs $N/2\alpha$ I/O time. In phase 2, the number of iterations for merging is $O(\log N/\log m^{\alpha/2})$; each iteration takes $O(N/2\alpha)$ I/O time. Thus, phase 2 needs $O((N/2\alpha) \cdot (\log N/\log m^{\alpha/2})) = O(N \log N/(2\alpha \log m^{\alpha/2}))$ I/O time. Therefore the total execution time is $O(N \log N/(2\alpha \log m^{\alpha/2}) + N/2\alpha) = O(N \log N/(2\alpha \log m^{\alpha/2}))$. Q.E.D.

LEMMA 3.4. *The memory size used in the proposed algorithm for balanced sorting on the mesh-connected processor array of* α^2 *processors is* $O(\alpha^2 m^{\alpha/2})$.

Proof. For the $m^{\alpha/2}$-way merge in a processor, we maintain a heap of $m^{\alpha/2}$ elements. So the total size of local memories is $\alpha^2(4m^{\alpha/2})$. Q.E.D.

4. COMPLEXITIES OF BALANCED SORTING

In this section, we show that the algorithms presented in the previous section really achieve their respective memory size lower bounds and exhibit asymptotic full speedups. But first we need present a general result of I/O complexity of sorting on processor arrays.

4.1. *I/O Complexity of Sorting*

Suppose sorting is implemented in a system that consists of β modules, each having M places to hold data, where $M \gg \beta$ and $N > \beta M$. We also assume that in an I/O operation, every module can receive at most t data from outside, where $1 \leq t \leq M$. The following lemma is a stronger version of Song's result on I/O complexity of sorting (Song, 1981).

LEMMA 4.1. *On the system defined above, the number of required I/O operations for sorting is* $\Omega(N \log N/(t \log M))$.

Proof. Since we are proving the lower bound for I/O, we may assume that in every I/O operation the computing power of a module is only limited by the number of data it encounters during that period of time. As explained below, this implies that in the first $\beta M/t$ I/O operations, there

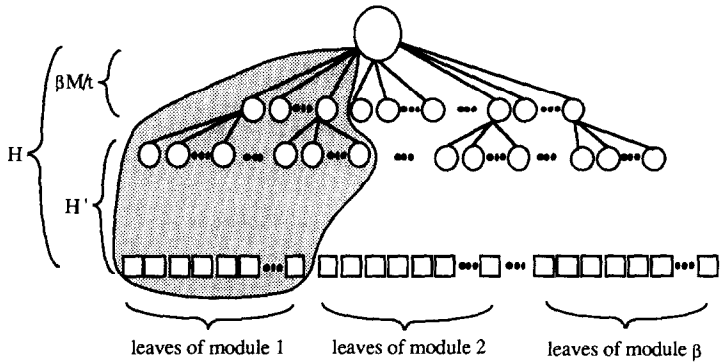

FIG. 4. Tree representation of sorting on β modules.

might be as many as $b(M!)(M^t)^{(\beta M/t - M/t)}$ possible outcomes generated by the system.

Consider a particular module. In the first M/t I/O operations, since there are at most M data that can be transmitted into the system, the module will encounter at most M data. The number of possible outcomes produced by this module is at most $M!$. In the next I/O operation, this module can receive at most t data, thus there are at most M^t possible outcomes generated by it. Therefore, at most $(M!)M^t$ outcomes are generated after $(M/t) + 1$ I/O operations, at most $(M!)M^{2t}$ outcomes are generated after $(M/t) + 2$ I/O operations, and so on. Since we have β modules, after the first $\beta M/t$ I/O operations, at most $\beta(M!)(M^t)^{(\beta M/t - M/t)}$ possible outcomes are generated.

Then in the $((\beta M/t) + 1)$th I/O operation, each module receives at most t data and generates at most M^t possible outcomes, and so on. Sorting on such a system can be represented by a tree as shown in Fig. 4. In the figure, each leaf corresponds to an outcome indicating an ordering of the initial N data. (The leaves in different modules may represent the same permutation.) We are therefore looking for a number H' such that

$$\beta(M!)(M^t)^{(\beta M/t - M/t)} \cdot (M^t)^{H'} \geq N! \qquad \text{or}$$

$$\log \beta + \log M! + (\beta - 1)M \log M + H't \log M \geq \log N!.$$

Using Stirling's approximation for $\log N!$ and $\log M!$, we have

$$\begin{aligned} H't \log M &\geq N \log N - N + O(\log N) - (\log \beta + M \log M - M \\ &\quad + O(\log M) + (\beta - 1)M \log M) \\ &\geq N \log N + \text{lower-order terms in } N + (\log \beta + M \log M \\ &\quad + (\beta - 1)M \log M) + \text{lower-order terms in } M. \end{aligned}$$

Since $M \gg \beta$, this can be rewritten as

$$H' \geq (N \log N - \beta M \log M)/t \log M + \text{lower-order terms in } N + \text{lower-order terms in } M.$$

That is,

$$H = H' + \beta M/t \geq ((N \log N - \beta M \log M) + \beta M \log M)/t \log M + \text{lower-order terms in } N + \text{lower-order terms in } M.$$

Since $N > M$, this implies

$$H \geq (N \log N/t \log M) + \text{lower-order terms in } N.$$

So we have $H = \Omega(N \log N/t \log M)$. Q.E.D.

4.2. *Memory Size Lower Bounds*

We now employ Lemma 4.1 to show the minimum sizes of local memories for linearly connected and mesh-connected processor arrays to minimize the I/O requirements in performing balanced sorting.

THEOREM 4.1. *For a linearly connected array of α processors, each with computation bandwidth C, I/O bandwidth I, and $C/I = \log m$, the processors individually need $\Omega(m^{\alpha/t})$ size of local memory to minimize the array's I/O requirement in balanced sorting, where t is the number of data each I/O operation can handle.*

Proof. It is known that the number of comparisons needed to sort N data is $\Omega(N \log N)$ or $C_{\text{comp}} = \Omega(N \log N)$ (Knuth, 1973). The array's computation bandwidth C' is αC and I/O bandwidth is still I. Assume M is the processors' individual local memory size required. Since the processor array is balanced for sorting, by (2),

$$C_{\text{comp}}/C_{\text{io}} = C'/I = \alpha C/I = \alpha \log m.$$

But, by Lemma 4.1, we have $C_{\text{io}} = \Omega(N \log N/(t \log M))$. This implies that

$$\alpha \log m = C_{\text{comp}}/C_{\text{io}} = O(C_{\text{comp}} \cdot t \log M/N \log N).$$

So $\log M = \Omega(\alpha \log m \cdot N \log N/(tC_{\text{comp}}))$. Since we want to minimize C_{io} and hence C_{comp}, and since $C_{\text{comp}} = \Omega(N \log N)$, we have $M = \Omega(m^{\alpha/t})$. Q.E.D.

For the linearly connected array in Section 3.1, since $C/I = \log m$ and $t = 1$, by Theorem 4.1, we know that each processor needs $\Omega(m^\alpha)$ local memory size for balanced sorting. The proposed algorithm actually achieves this lower bound.

THEOREM 4.2. *For a mesh-connected array of α^2 processors, each with computation bandwidth C, I/O bandwidth I, and $C/I = \log m$, the processors need $\Omega(m^{\alpha/t})$ size of individual local memory to minimize the array's I/O requirement in balanced sorting, where t is the number of data each I/O operation can handle.*

Proof. The array's computation bandwidth C' is $\alpha^2 C$ and I/O bandwidth I' is αI. Assume M is the processors' individual local memory size required. Since the processor array is balanced for sorting, by (2),

$$C_{\text{comp}}/C_{\text{io}} = C'/I' = \alpha^2 C/(\alpha I) = \alpha \log m.$$

But, by Lemma 4.1, we have $C_{\text{io}} = \Omega(N \log N/(t \log M))$. This implies that

$$\alpha \log m = C_{\text{comp}}/C_{\text{io}} = O(C_{\text{comp}} \cdot t \log M/N \log N).$$

So $\log M = \Omega(\alpha \log m \cdot N \log N/(tC_{\text{comp}}))$. Since we want to minimize C_{io} and hence C_{comp}, and since $C_{\text{comp}} = \Omega(N \log N)$, we have $M = \Omega(m^{\alpha/t})$. Q.E.D.

For the mesh-connected array in Section 3.2, since $C/I = \log m/2 = \log m^{1/2}$ and $t = 1$, by Theorem 4.2, we know that each processor needs $\Omega((m^{1/2})^{\alpha/t}) = \Omega(m^{\alpha/2})$ local memory size for balanced sorting. The proposed algorithm actually achieves this lower bound.

4.3. *Asymptotic Full Speedups*

THEOREM 4.3. *The proposed algorithm on the linearly connected processor array exhibits an asymptotic full speedup with α processors.*

Proof. Consider sorting N data on a single processor with $\Theta(m)$ size of memory. By taking $\beta = 1$ and $t = 1$ in Lemma 4.1, we know that the number of I/O operations and hence the execution time is $\Omega(N \log N/\log m)$. From Lemma 3.1, we know that the proposed algorithm on the linearly connected processor array takes $O(N \log N/(\log m^\alpha)) = O((N \log N/\log m)/\alpha)$ execution time to accomplish sorting. Thus the algorithm exhibits an asymptotic full speedup with α processors. Q.E.D.

THEOREM 4.4. *The proposed algorithm on the mesh-connected processor array exhibits an asymptotic full speedup with α^2 processors.*

Proof. Again, a single processor takes $\Omega(N \log N/\log m)$ execution time. By Lemma 3.3, the proposed algorithm on the mesh-connected

processor array takes $O(N \log N/(2\alpha \log m^{\alpha/2})) = O((N \log N/\log m)/\alpha^2)$, and thus exhibits an asymptotic full speedup with α^2 processors.

Q.E.D.

5. Conclusion

In this paper, we have derived the memory siz lower bounds required to balance sorting on a linearly connected processor array and a mesh-connected processor array. They are higher than what were claimed by Kung (1985), reflecting the fact that communication among the processors indeed influences the design of balanced computations.

It is important to design balanced algorithms on processor arrays and analyzing the factors for achieving balanced computations provides insight into the design of high performance architectures. We constantly emphasize that for balanced computation, computing time and I/O time must be equalized. However, in real situations, it will be meaningless not to consider the amount of time spent in the computation. It is easy to schedule a computing process such that the computing time equals the I/O time by allowing the CPUs to be not so active. So besides equalizing the computing time and the I/O time, it is necessary to minimize the total execution time. This is exactly the place where the memory size plays its role.

For balancing sorting on a processor, since the size of the processor's memory must be increased exponentially as computation bandwidth increases, it may become unrealistically large. Kung (1985) thus claimed that, for the sorting problem, one should not expect any substantial speedup without a significant increase in the processor's I/O bandwidth. Given the results of this paper, we can conclude further that, for sorting on a processor array, one should not expect any essential speedup without significant increases in both the I/O bandwidth of the array and the communication bandwidth among the processors.

References

Aho, A., Hopcroft, J., and Ullman J. (1976), "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, MA.

Chen, P. Y., and Nussbaum, M. (1985), Sorting with systolic architecture, *in* "Proceedings of the 1985 International Conference on Parallel Processing," pp. 865–868.

Knuth, D. E. (1973), "The Art of Computer Programming," Vol. 3, "Sorting and Searching," Addison-Wesley, Reading. MA.

Kung, H. T. (1985), Memory requirements for balanced computer architecture, *J. Complexity* **1,** 147–157.

Lang, H. W., Schimmler, M., Schmeck, H., and Schroder, H. (1985), Systolic sorting on a mesh-connected network, *IEEE Trans. Comput.* **C-34,** 652–658.

MIRANKER, G., TANG, L., AND WONG, C. K. (1983), A zero-time VLSI sorter, *IBM J. Res. Develop.* **27,** 140–148.

NASSIMI, D., AND SAHNI, S. (1979), Bitonic sort on a mesh-connected parallel computer, *IEEE Trans. Comput.* **C-27,** 2–7.

OWENS, R. M., AND JA'JA', J. (1985), Parallel sorting with serial memories, *IEEE Trans. Comput.* **C-34,** 379–383.

SIEWIOREK, D. P., BELL, C. G., AND NEWELL, A. (1982), "Computer Structures: Principles and Examples," McGraw–Hill, New York.

SONG, S. W. (1981), "On a High-Performance VLSI Solution to Database Problems," Ph.D. thesis, Department of Computer Science, Carnegie–Mellon University.

THOMPSON, C. D., AND KUNG, H. T. (1977), Sorting on a mesh-connected parallel computer, *Comm. ACM* **20,** 263–271.