Appeared in "Journal of Parallel and Distributed Computing, July 1995 "

Overlapping Computations, Communications and I/O in Parallel Sorting †

Mark J. Clement
Computer Science Department
Brigham Young University
Provo, Utah 84602-6576
801-378-7608
clement@cs.byu.edu

Michael J. Quinn
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331-3202
503-737-5572
quinn@cs.orst.edu

August 4, 1995

Abstract

In this paper we present a new parallel sorting algorithm which maximizes the overlap between the disk, network, and CPU subsystems of a processing node. This algorithm is shown to be of similar complexity to known efficient sorting algorithms. The pipelining effect exploited by our algorithm should lead to higher levels of performance on distributed memory parallel processors. In order to achieve the best results using this strategy, the CPU, network and disk operations must take comparable time. We suggest acceptable levels of system balance for sorting machines and analyze the performance of the sorting algorithm as system parameters vary.

1 Introduction

Sorting is one of the most studied problems in computer science. It is also of great practical importance because of the number of sorting operations that occur in database manipulations. Many algorithms have been suggested that perform efficient sorting on parallel computers. The algorithm presented here maximizes the overlap between computations, communications and I/O in distributed memory parallel processors. This overlap enables the algorithm to achieve speedup values that would be unattainable with a non-overlapped version.

We will address the problem of sorting a data set of N tuples which is small enough to reside in internal memory, but begins and ends on disk. We use the term Internal Sorting of External Data (ISED) to refer to this problem. Initially, the data in the file to be sorted is distributed throughout the local disks connected to the processing elements. This distributed memory, distributed disk processor model has been suggested previously for high performance database use [5, 13]. At the conclusion of the algorithm, the sorted file will also be distributed in non-overlapping sorted runs on the processors' disks. We will assume a multicomputer model where

†This research was supported by Sequent Computer Systems, Inc. and the Oregon Advanced Computing Institute.

each processor has a local disk, local memory and a network connection with other processors. Most of the current generation of multicomputers have DMA controllers to transfer data between processors and disk controllers. These controllers can perform the transfer of data from the disk to memory without direct processor intervention. We will assume this model for our analysis.

Several approaches have been used to achieve parallel solutions to the sorting problem. Divide and conquer or merge-based sorts begin with an equal number of elements distributed to each of the processors. Each processor sorts its part of the total file and then each pair of processors merge their sorted lists. This process continues until the final processor performs a sequential merge of two sorted halves of the file. The speedup attainable through this approach is limited by the large ($O(N)$) sequential component of the algorithm in the final merge [15].

Partition-based sorts avoid the sequential merge by distributing the data in such a way that each processor's part of the data does not overlap with any other processor's data. The performance of partition sorts is limited by how evenly the data is divided among the processors. Our algorithm is similar to the quickmerge algorithm proposed by Quinn [15] and relies on the statistical sampling theory presented in DeWitt [5]. Our main contribution lies in the customization of the algorithm so that it overlaps communication, computation, and I/O. Our overlapped sorting algorithm (OVS) holds a comparative advantage over non-overlapped algorithms in the following areas:

- Because of the overlap between computation and I/O, OVS has the potential of nearly doubling the speedup attainable on parallel machines.
- Interprocessor communication in the algorithm does not degrade the speedup of the algorithm because it occurs while other computations are being performed.
- The traffic on the communication network is distributed throughout the run time of the algorithm. OVS performs communications throughout the algorithm instead of having all processors compete for the network at once.

In the first phase of the OVS algorithm, all of the processors take a statistical sample of the data contained on their local disk. They then sort the samples and determine splitting vectors for their sample of the file. All of the processors then communicate to build a global splitting vector to be used throughout the sort.

During the second phase, the processors divide the remainder of the file to be sorted into k blocks. Each processor will perform three overlapping activities for iterations $0 \leq i \leq k$:

- Initiate a read operation to retrieve the ith block of unsorted data.
- Start the DMA write of sorted subblocks from the sort in iteration $(i-1)$ to their destination processors. Also start the DMA read of sorted subblocks from other processors.
- Sort the block of data from the $(i-1)$th disk read and merge sorted subblocks from the $(i-1)$th iteration.

The disk read, network communication and computation steps are pipelined so that they can occur at the same time. During the last phase of the algorithm, the sorted runs from each iteration are merged and written to the disk.

Through overlapping computations, communications and I/O, significant improvements can be made in the speedup attainable in sorting on parallel computers. The algorithm presented here is tailored to exploit this concurrency while adding minimal additional computational complexity. We review previous sorting algorithms which have contributed to this algorithm in Section 2. In Section 3 we present a formal description of the sorting algorithm. A complexity analysis of the algorithm is performed in Section 4. Section 5 compares this algorithm to other parallel algorithms for varied levels of system balance.

2 Related Research

Several articles [3, 5, 15] and books [1, 9] survey previous research in sorting algorithms. We will not attempt to review all of the related work here, but will provide an overview of the sorting algorithms which have contributed to this research and will compare them to the OVS algorithm.

2.1 Parallel Sorting by Regular Sampling (PSRS)

Parallel sorting by regular sampling [17] is a partition-based sort which has demonstrated good speedup characteristics for the internal sorting problem. The regular sampling algorithm was presented as an internal sorting algorithm but could be customized to perform internal sorting of external data. Figure 1 shows the sequence of operations the algorithm performs to to sort N keys on p processors.

1. During the first phase of the algorithm each of the p processors sorts a contiguous list of size N/p using sequential quicksort. From this list, $p-1$ evenly spaced samples are taken on each of the p processors. This sampling strategy insures that the number of records to be merged by the processor with the most records will be no more than twice the average number of records merged by a processor. Minimizing this data skew has been shown to be extremely important in parallel database algorithms [6, 10].

2. During the second phase, the $p(p-1)$ samples are sorted and $p-1$ evenly spaced pivots are taken from the set of samples. Each processor then finds where each of the $p-1$ pivots divides its list using $p-1$ binary searches. Each processor then sends a subset of the sorted list to each of the other processors, using the $p-1$ pivot values. Processor q will receive sorted sublists with values between pivot q and $q+1$.

3. During the last phase of the algorithm, each processor merges the $p-1$ sorted sublists. The pivot values established in the second phase of the algorithm insure that the final sorted lists on each processor will be non-overlapping. Since the whole data set has been used to come up with the pivot values, no processor will have more than $2N/p$ elements to merge in the final phase of the algorithm.

The principle limitation of customizing the PSRS algorithm for internal sorting of external data is that the sequential sort in phase one can not start until all of the N/p keys are available. The customized algorithm would have to read all of the N records from disk before phase one could begin. This eliminates the opportunity for overlapping computation and disk I/O. Another drawback of the PSRS algorithm in external sorting is that the network communication of sorted sublists is concentrated at the end of phase two and cannot be overlapped with sorting computations. The OVS algorithm to be presented in this paper eliminates these two barriers.

2.2 Probabilistic Splitting (PS)

The probabilistic splitting [5] algorithm is another partition-based algorithm which relies on a probabilistic sample of the unsorted data to determine pivot values. Good load balance is crucial to the performance of parallel sorting algorithms. PSRS insures a good balance by sampling the entire data set. The PS algorithm relies on the premise that there is a high probability that a much smaller number of samples can be used to achieve an equivalent level of load balance. The PS algorithm has three phases.

1. The first phase of the algorithm takes a sample of the data items on the local disk of the processor. Instead of a simple random sample of size ps on a p processor system, a sample consisting of p simple random samples of size s is taken, one from each processor. This technique is known as "stratified random sampling" [16]. Let α, for skew, denote the ratio of the maximum number of records sorted by a processor over the average number of records sorted by a processor. Then with N records to be sorted on p processors the maximum number of records to be sorted by one processor is $\alpha N/p$. Blelloch et al. [3] proved that if s is the number of samples each processor takes from the data file, the probability P_r that there will be more than $\alpha N/p$ records on any processor is given by:

   $$P_r \leq N e^{-(1-1/\alpha)^2 \alpha s/2}$$

   Solving for the number of samples required for a confidence level of $(1 - P_r)$:

   $$s = \frac{2\ln(N/P_r)}{(1-1/\alpha)^2 \alpha}$$

   These samples are then sent to a central processor which sorts the samples and broadcasts the resultant $p - 1$ pivot values to the other processors.

2. Once the pivot values are known, all processors read the records from their local disks and put them into buckets corresponding to the processors that the records are destined for. The PS algorithm is designed for external sorting, where the data to be sorted does not fit into memory. When memory is full of partitioned data, the processors redistribute the records so that each key is at the appropriate destination processor.

3. Once the data has been distributed, the processors use a sequential sorting algorithm to sort at most $\alpha N/p$ records which are then written to disk.

One of the main assumptions made by the PS algorithm is that there is no overlap between I/O operations, CPU operations or network traffic. The statistical sampling nature of the algorithm, however, makes it a good candidate for overlapped execution. A good overlapped sorting algorithm will determine the pivot values as early as possible in the execution of the algorithm so that network communications and I/O can be overlapped.

2.3 Other Sampling Issues

Although the PS algorithm has many attractive properties, the bound on the number of samples taken to build the pivot values is still too large to allow us to overlap I/O and sorting effectively. Since the samples must be taken at random, they will all require a separate disk I/O. For a 1,000,000 record file, the PS bound would suggest that we need 201 random samples to achieve a 95% confidence level that the skew would be less than 1.5. During a disk read operation, the entire sector must be transferred from disk into memory before the data can be accessed. The time required to read in a single record on a disk is about the same as that required to read a whole sector. For common database parameters of 8 Kbyte sectors and 100 byte tuples [10], the PS algorithm will read 64% of the records during the sampling phase for a 40 processor system. For 60 processors, 96% of the file will have been read in the sampling phase. This strategy is clearly not suited well to machines with large numbers of processors.

Page level stratified sampling (PLSS) has been suggested by Seshadri as a more efficient method of determining pivot values for a data set [16]. By using all of the records in the disk sector or "page," equivalent confidence levels in skew values can be obtained using fewer disk I/O operations. Seshadri proves that using all of the keys in a sector as sample values will always result in lower skew values than tuple level stratified sampling. He also derives a new bound on the number of samples necessary to achieve a given skew. The number of samples required by Seshadri's bound is dependent on the number of processors and the size of the file to be sorted and requires significantly fewer samples for a given confidence level. For these reasons we will use page level stratified sampling with Seshadri's bound to determine the pivot values for our sorting algorithm.

The probability P_r that stratified sampling will result in pivot values with more than $\alpha N/p$ records between pivots is given as [16]

$$P_r = \min(P_1, P_2)$$

where

$$P_1 = p\alpha^s \left(\frac{p-a}{p-1}\right)^{ps-s},$$

$$P_2 = pA(\alpha N/p)\left(1 - r^{N-\alpha N/p}\right)/(1-r)$$

and

$$r = \frac{(\alpha N/p+1)(N-\alpha N/p-1-(ps-s))}{(\alpha N/k+1-(s-1))(N-\alpha N/k-1)}.$$

$A(m)$ is defined as

$$A(m) = \begin{pmatrix} ks \\ s-1 \end{pmatrix} \left(\frac{m^{\underline{s-1}}(N-m)^{\underline{ks-(s-1)}}}{N^{\underline{ks}}}\right)\left(\frac{ks-(s-1)}{N-m}\right),$$

where $n^{\underline{m}}$ is defined as the falling factorial: $n^{\underline{m}} = n(n-1)...(n-m+1)$ with m a positive integer. Tables 1 and 2 show the number of samples required for Seshadri's bound as the number of processors and the number of records in the file varies. The percentage of the file sampled was calculated by assuming that each tuple was 100 bytes long and that the whole 8 Kbyte sector was read for each sample. Using page level sampling, a good set of pivot values can be gathered while performing a small percentage of the total I/Os required to read in the file.

2.4 External Sorting

Other researchers have attempted to overlap computations and disk accesses in external sorting problems. A parallel file sorting algorithm has been implemented on the JASMIN hardware and software system which utilizes input streams to pipeline I/O and computation phases [2]. This external sort is characteristic of other external sorting algorithms in that it utilizes a merge sort in the final phase of the algorithm. The performance for these algorithms is again limited by the sequential merge of $O(N)$ elements.

3 Overlapped Sorting (OVS)

The overlapped sorting algorithm presented here is designed to maximize the overlap between I/O, network and CPU functions. Our algorithm contains features from the PSRS and PS algorithm along with significant modifications to enable overlap. An overview of the algorithm is given in Figure 2.

1. During the first stage of the algorithm, the pivot values are determined by collecting a page level stratified sample. Each processor reads s pages from the disk and sorts the samples. The samples are sent to one processor which merges the ps elements and picks $p-1$ evenly spaced pivot values and broadcasts the pivot values to the rest of the processors.

2. During the second phase of OVS, each processor divides the unsorted data into k blocks and for each iteration i, where $1 \leq i \leq k$, the processors overlap the following three operations:

   (a) Initiate a read of the ith $(N/p)/k$ records from the disk.

   (b) Sort the block of data read in iteration $i-1$ using sequential quicksort and merge the $p-1$ sorted subblocks received from the network in iteration $i-1$.

   (c) Initiate $p-1$ network sends of the data sorted in iteration $i-1$ and initialize the DMA hardware to receive the data sent by other processors.

   Figure 3 illustrates the execution of the second phase of the OVS algorithm. A pipeline is formed with data originating on disk being fed into the sort operation of the next iteration. Sorted data is then output to the network stage of the pipeline. Records received from the network are then merged to form one of k sorted sublists.

3. At the end of the second phase of the algorithm, each processor will have k sorted sublists of maximum length $\frac{\alpha N}{kp}$. During the third phase, each processor will merge the k sorted

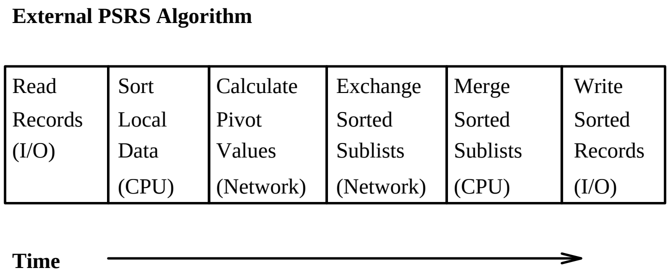

Figure 1: Phases of the parallel sorting by regular sampling (PSRS) algorithm modified for internal sorting of external data.

Number of processors	Number of samples required	Fraction of file read in sampling phase
10	29	2.3%
20	37	5.9%
40	44	14.1%
80	51	23.0%
100	52	41.6%

Table 1: Samples per processor for $\alpha = 1.5$, $N = 10^6$, 80 records per page, p varying [16] .

N	Number of samples required	Fraction of file read in sampling phase
800,000	46	23.3%
1,600,000	47	11.9%
6,400,000	47	2.9%
12,800,000	48	1.5%

Table 2: Samples per processor for $p = 50$, $\alpha = 1.5$, 80 records per page, N varying [16].

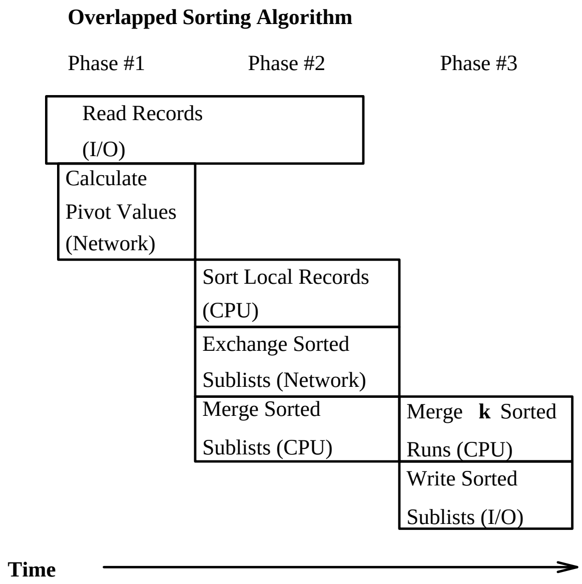

Figure 2: Top level view of the overlapped sorting algorithm (Phases of the algorithm are not drawn to scale).

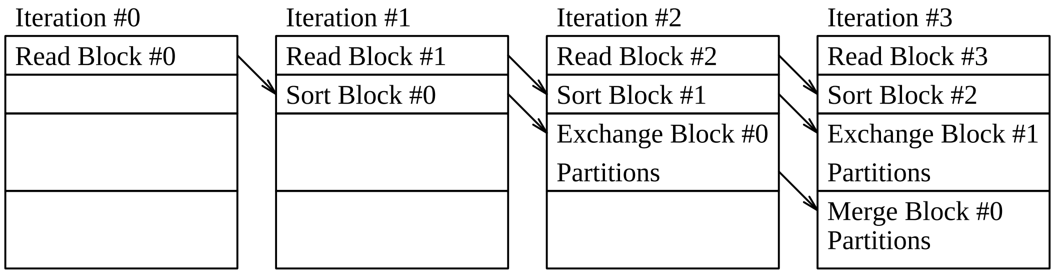

Figure 3: Execution of the second phase of the overlapped sorting algorithm.

sublists to create the final sorted run. The sorted records can be written to disk as they are merged so that CPU and I/O times can be overlapped.

4 Complexity Analysis

In this section we will examine differences in the complexity between the PSRS algorithm and the OVS algorithm. The number of iterations k used in the algorithm will be specified as a function of other system parameters. We will also see that the number of computations in the various phases of the OVS algorithm are within a constant of the corresponding phases in the PSRS algorithm. *Unit* will be defined as the time necessary to perform a comparison between two keys. We will denote C_s as the constant (in *Units*) for the sorting operation and C_m as the constant for merging. Given similar complexity values, the OVS algorithm holds significant advantages, since it is able to overlap execution of various functions.

4.1 Sampling

The sampling phase of the OVS algorithm must be performed before the pipeline of the last two phases can be started. During the sampling phase each processor will read s random records, sort these records and send the s records to a single processor where they will be merged. Evenly spaced pivot values will then be broadcast to all of the processors and phase 2 of the algorithm will begin.

Parallel sorting is particularly attractive for large databases. For data sets with greater than 1,000,000 tuples per processor the sampling phase of the OVS algorithm takes an insignificant fraction of the total time for the algorithm. Table 3 shows the percentage of network, disk and CPU time spent in the sampling phase of the algorithm for $N = 1,000,000 * p$ with a 95% confidence level in a skew of 1.5 or less. For the networking figures we assumed that ps tuples would have to pass through the bisectional bandwidth of the network during the sampling phase compared to N records worst case during the whole algorithm. Bisectional bandwidth is defined as the rate at which communication can take place between one half of a computer and the other [18]. The percentage of disk time compares the number of tuples which were read in during the sampling phase (assuming 80 tuples per sector) with the total number of tuples. The CPU column in Table 3 compares the sorting and merging time spent during the sampling phase to the total computational time of the algorithm. For more than 1024 processors a parallel merge scheme would be more a more efficient way to generate the sample. Table 3 shows that for large databases, the time spent in the sampling phase of the OVS algorithm is small compared to the rest of the algorithm. We will ignore it in our overall complexity analysis.

4.2 Disk I/O

The PSRS algorithm requires all data to be available in memory when the sort begins. In applying PSRS to the problem of ISED we must extend the algorithm to include disk accesses. The most efficient way to read the records from the disk is to read them all in a single block transfer. With D_s as the startup time (in *Units*) for a disk I/O and D_r as the number of *Units* necessary to transfer a single record from disk to memory, the time to read a single processor's share of the data file assuming contiguous records $T^{PSRS}_{diskread} = D_s + (N/p)D_r$. Time values will be specified in terms of the time necessary to perform a comparison between key values on the given CPU in order to simplify other analysis.

The OVS algorithm performs k separate disk I/O operations to read in the data for an I/O complexity of

$$\begin{aligned} T^{OVS}_{diskread} &= k\left(D_s + \frac{N}{kp}D_r\right) \\ &= kD_s + \frac{N}{p}D_r \end{aligned}$$

The OVS algorithm performs $(k-1)$ more disk I/O startups than the PSRS algorithm during the read phase. The writes to disk could be overlapped with either algorithm, resulting in time

$$T^{OVS}_{diskwrite} = kD_s + \frac{\alpha N}{p}D_r$$

The total disk time for disk I/O with the overlap algorithm is

$$T^{OVS}_{disk} = 2kD_s + \frac{N(\alpha+1)}{p}D_r$$

4.3 Sorting

The sorting phase of the PSRS algorithm requires average time

$$T^{PSRS}_{sort} = C_s\frac{N}{p}\log\left(\frac{N}{p}\right)$$

to sort the N/p records on each processor. The OVS algorithm performs k sorts of length $\frac{N}{kp}$ for average time complexity

$$\begin{aligned} T^{OVS}_{sort} &= kC_s\frac{N}{kp}\log\left(\frac{N}{kp}\right) \\ &= C_s\frac{N}{p}\log\left(\frac{N}{kp}\right) \end{aligned}$$

We will simplify later calculations by noting that

$$T^{OVS}_{sort} \approx C_s\frac{N}{p}\log\left(\frac{N}{p}\right) \text{ when } k \ll p$$

With a data distribution which produces $O(n^2)$ comparisons for quicksort,

$$T^{PSRS}_{sort} = C_s\frac{N^2}{p^2}$$

and

$$T_{sort}^{OVS} = kC_s \frac{N^2}{(kp)^2} = C_s \frac{N^2}{kp^2}.$$

With either average or worst case data distribution, the number of computations in the OVS sorting phase is less than or equal to that of the PSRS algorithm for positive integer values of k.

4.4 Network Transfers

In the worst case data distribution for sorting, the network time will be limited by the bisectional bandwidth of the interconnection network. We will define β_s as the network message startup time expressed in number of *Units*. β_r is the inverse of the bisectional bandwidth of the interconnection network. It will be measured in the number of *Units* necessary to transfer one record through the bisectional bandwidth of a multicomputer. The time necessary for network transfers in the PSRS algorithm is $T_{net}^{PSRS} = p\beta_s + N(\beta_r)$. The OVS algorithm takes

$$\begin{aligned} T_{net}^{OVS} &= k\left(p\beta_s + \frac{N}{k}\beta_r\right) \\ &= kp\beta_s + N(\beta_r) \end{aligned}$$

for $(k-1)$ times more network startup times than are required for the PSRS algorithm. We will show in later sections that the message startup time is not significant when compared to the transfer time in the sorting algorithm.

4.5 Merging

In the merging phase for the the regular sampling algorithm, p sorted sublists of length N/p^2 are merged using a heap merge algorithm to form a per-processor sorted segment of the data. The regular sampling algorithm guarantees that the skew will be less than 2. The merging complexity is

$$T_{merge}^{PSRS} = C_m \frac{\alpha N}{p} \log(p).$$

For each of the k iterations in phase two of the OVS algorithm, p subblocks of maximum length $\frac{\alpha N}{kp^2}$ will be received over the network. At some point during the algorithm, kp of these subblocks must be merged together to form a single sorted list of length no more than $\frac{\alpha N}{p}$. We have considered three methods for performing this merge.

1. The approach used in the OVS algorithm consists of a merge of subblocks from each iteration to form k sublists of length $\frac{\alpha N}{kp}$. The heap merge of each of these subblocks during phase two of the algorithm will take time

$$T_{merge}^{phase2} = kC_m \frac{\alpha N}{kp} \log(p) = C_m \frac{\alpha N}{p} \log(p).$$

These k sublists are then merged to form a single sorted list during phase three. The complexity of this final stage is

$$T_{merge}^{final} = C_m \frac{\alpha N}{p} \log(k).$$

The expected total time for this method

$$T_{merge}^{OVS} \quad = \quad C_m \left(\frac{\alpha N}{p} \log(p) + \frac{\alpha N}{p} \log(k) \right).$$

As long as $k \leq p$ the merge phase of the OVS and PSRS algorithms will both be of order $N \log(p)$.

2. A second method for merging the subblocks is to keep a single, ever-growing sorted list which contains all of the records received up to that point in the algorithm. For each iteration, the algorithm merges the small subblocks using a heap merge algorithm and then merges this combined list with the growing sorted list. This merging algorithm has complexity $kC_m \frac{\alpha N}{kp} \log(p)$ for the small lists and $C_m \frac{\alpha N}{kp}(1 + 2 + \cdots + k)$ for the two way merge for a total of

$$C_m \left(\frac{\alpha N}{p} \log(p) + \frac{k(k+1)\alpha N}{kp} \right) = C_m \left(\frac{\alpha N}{p} \log(p) + \frac{(k+1)\alpha N}{p} \right).$$

   With k a positive integer, method one has lower complexity since

$$\frac{\alpha N}{p} \log(p) + \frac{\alpha N}{kp} \log(k) < \frac{\alpha N}{p} \log(p) + \frac{(k+1)\alpha N}{p}.$$

3. A third method for merging the subblocks is to merge the p sorted subblocks of length at most $\frac{\alpha N}{kp^2}$ with the growing sorted list in a $p+1$ way heap merge each iteration. This merging algorithm has complexity $C_m k(k+1) \frac{\alpha N}{kp} \log(p+1)$. Comparing the complexities of method one and method three:

$$\frac{\alpha N}{p} \log(p) + \frac{\alpha N}{kp} \log(k) < k \frac{\alpha N}{p} \log(p)$$

   since $\log(p) + \frac{1}{k} \log(k) \leq k \log(p)$ for all $k < p$. Again method 1 has lower complexity and was chosen for the merging method in the OVS algorithm.

4.6 Totals

Given the complexity for each phase of the OVS algorithm, we can now evaluate the overall algorithm. The CPU time in phase two of the OVS algorithm is

$$T_{sort+merge}^{phase2} = C_s \frac{N}{p} \log\left(\frac{N}{p}\right) + C_m \frac{\alpha N}{p} \log(p)$$

assuming $k \ll p$ using average case analysis. Combining the disk and network time with CPU time

$$
\begin{aligned}
T^{OVS}_{phase2} &= \max(T^{OVS}_{diskread}, T^{phase2}_{sort+merge}, T^{OVS}_{net}) \\
&= \max\left(kD_s + \frac{N}{p}D_r, C_s\frac{N}{p}\log\left(\frac{N}{p}\right) + C_m\frac{\alpha N}{p}\log(p), kp\beta_s + N(\beta_r)\right) \qquad (1)
\end{aligned}
$$

The total expected time for the OVS algorithm is

$$
\begin{aligned}
T^{OVS}_{total} &= T^{OVS}_{phase2} + \max\left(T^{OVS}_{diskwrite}, T^{final}_{merge}\right) \\
&= \max\left(kD_s + \frac{N}{p}D_r, C_s\frac{N}{p}\log\left(\frac{N}{p}\right) + C_m\frac{\alpha N}{p}\log(p), kp\beta_s + N(\beta_r)\right) + \\
&\quad \max\left((kD_s + \frac{\alpha N}{p}D_r), \left(C_m\frac{\alpha N}{p}\log(k)\right)\right). \qquad (2)
\end{aligned}
$$

In the asymptotic case with $N \gg p$ the computational complexity of the sorting and merging operations performed by the OVS algorithm is

$$
\begin{aligned}
T^{OVS}_{sort} + T^{OVS}_{merge} &= C_s\frac{N}{p}\log(\frac{N}{p}) + C_m\left(\frac{\alpha N}{p}\log(p) + \frac{\alpha N}{p}\log(k)\right) \\
&= C_s\frac{N}{p}\log(N) - C_s\frac{N}{p}\log(p) + C_m\frac{\alpha N}{p}\log(kp) \\
&= O\left(\frac{N}{p}\log(N)\right).
\end{aligned}
$$

which is optimal for the comparison based sorting problem [4].

4.7 Selecting k

We can use pipeline theory [7] to determine the appropriate number of iterations (k) for phase 2. The disk read, sorting, network transfer and merging operations can be likened to four instructions being executed by a CPU pipeline. The number of iterations can be likened to the number of stages in the pipeline. We will use $f(N)$ to denote the time necessary to complete the sorting operation and will assume that the disk, network and merging operations take similar time. The time to complete all four operations in terms of $f(N)$ is

$$
T_{pipeline} = \begin{cases} 4f(N) & \text{as } k \to 1 \\ (4 + (k-1))\frac{f(N)}{k} & \text{where } 1 < k < \infty \\ f(N) & \text{as } k \to \infty \end{cases}. \qquad (3)
$$

Given the expression for pipeline time, we can develop the expected decrease in execution time due to the increased number of iterations.

$$
\begin{aligned}
T^{k+1}_{pipeline} &= T^{k}_{pipeline} \frac{(4+k)\frac{f(N)}{k+1}}{(4+(k-1))\frac{f(N)}{k}} \\
&= T^{k}_{pipeline} \frac{k^2+4k}{k^2+4k+3} \qquad (4)
\end{aligned}
$$

Hence

$$
\begin{aligned}
T^{k}_{pipeline} - T^{k+1}_{pipeline} &= T^{k}_{pipeline} - T^{k}_{pipeline}\left(\frac{k^2+4k}{k^2+4k+3}\right) \\
&= T^{k}_{pipeline}\left(1 - \frac{k^2+4k}{k^2+4k+3}\right).
\end{aligned}
$$

The system parameters must be somewhat balanced in order for the pipelining to have an effect. If we assume that CPU time dominates during phase two of the OVS algorithm, we can derive the optimal value for the number of iterations k.

Theorem 4.1 *The optimal value for the number of iterations* (k) *for the OVS algorithm is*

$$
k = \left\lfloor -2 + \sqrt{1 + 3\frac{C_s \frac{N}{p}\log\left(\frac{N}{p}\right) + C_m \frac{\alpha N}{p}\log(p)}{\max(D_s, p\beta_s)}} \right\rfloor .
$$

Proof:

We can see from Equation 1, as the the number of iterations is increased by one, there is an additional overhead of $\max(D_s, p\beta_s)$ due to the added disk startup time and network message startup time. As k becomes large, the time for the algorithm will increase. Equation 3 shows that as $k \to 1$, the expected time also increases. Hence the function is concave up. We will derive the value for k which produces the minimum value for execution time. The optimal value for the number of iterations occurs when the decrease in execution time due to the increased pipeline depth is smaller that the additional overhead. This occurs when

$$
T^{OVS}_{phase2}\left(1 - \frac{k^2+4k}{k^2+4k+3}\right) < \max(D_s, p\beta_s). \qquad (5)
$$

Setting the two sides of Equation 5 equal to each other, we can derive the near optimal value for k given the other system parameters.

$$
T^{OVS}_{phase2}\left(1 - \frac{k^2+4k}{k^2+4k+3}\right) = \max(D_s, p\beta_s)
$$

Substituting CPU time for T_{phase2}^{OVS}:

$$\left(C_s\frac{N}{p}\log\left(\frac{N}{p}\right)+C_m\frac{\alpha N}{p}\log(p)\right)\left(1-\frac{k^2+4k}{k^2+4k+3}\right) = \max(D_s, p\beta_s)$$

$$\left(C_s\frac{N}{p}\log\left(\frac{N}{p}\right)+C_m\frac{\alpha N}{p}\log(p)\right)(3) = (k^2+4k+3)\max(D_s, p\beta_s)$$

$$(k^2+4k+3) = 3\left(\frac{C_s\frac{N}{p}\log\left(\frac{N}{p}\right)+C_m\frac{\alpha N}{p}\log(p)}{\max(D_s, p\beta_s)}\right).$$

Using the quadratic equation

$$k = \left\lfloor -2+\sqrt{1+3\frac{C_s\frac{N}{p}\log(\frac{N}{p})+C_m\frac{\alpha N}{p}\log(p)}{\max(D_s, p\beta_s)}}\right\rfloor.$$

□

The following example shows how Theorem 4.1 can be used to calculate k. We will use experimental results from a study of sorting on a SPARCstation 2. With the system configuration and GNU C compiler used in the testing we found that $C_s = 115$ and $C_m = 241$ CPU cycles. With $N = 100,000,000$, $p = 100$, $\alpha = 1.5$, $D_s = 5380$ and $\beta_s = 53$ instruction times, Theorem 4.1 would specify $k = 20$.

We have shown that the OVS algorithm performs a similar number of disk, network and CPU operations to the PSRS algorithm. If the subsystems of the target processing node are well balanced, the overlapped sorting algorithm should be able to perform better than non-overlapped sorting algorithms. In the next section we examine the effect of varying system parameters on the performance of the OVS algorithm.

5 System Balance

ISED sorting is a disk intensive problem. N records must be read and written to disk while $O(N\log(N))$ comparison operations are performed. Additionally, in the worst case, all N records will have to pass through the bisectional bandwidth of the interconnection network. If the disk and network are not well balanced with the processor speed, the time spent reading and writing data will often dominate [12]. There are few guidelines as to how well balanced the subsystems should be in order to achieve acceptable performance. In this section we will suggest system parameters which are well suited to the overlapped sorting algorithm. We will assume that $C_m = C_s$ in deriving expressions for system balance. These constants may differ in practice depending upon the machine architecture and compiler used.

We have shown in our sampling analysis that the last two phases of the overlapped sorting algorithm take the majority of the time for the algorithm. In phase two the disk, network and CPU operations can be overlapped. The best system parameters for this phase of the algorithm would have the time for each of these operations be equal or $T^{phase2}_{sort+merge} = T^{OVS}_{net} = T^{OVS}_{diskread}$. During the third phase of the algorithm, the disk and merging time should be equal for best system utilization; i.e., $T^{phase3}_{merge} = T^{OVS}_{diskwrite}$.

The conditions for phase two are satisfied when both the network and disk times are balanced with the CPU time.

$$\begin{aligned} T^{phase2}_{sort+merge} &= T^{OVS}_{net} \\ \Rightarrow \frac{N}{p}\log\left(\frac{N}{p}\right) + \frac{\alpha N}{p}\log(p) &= kp\beta_s + N\beta_r \end{aligned}$$

and

$$\begin{aligned} T^{phase2}_{sort+merge} &= T^{OVS}_{diskread} \\ \Rightarrow \frac{N}{p}\log\left(\frac{N}{p}\right) + \frac{\alpha N}{p}\log(p) &= kD_s + \frac{N}{p}D_r. \end{aligned}$$

If we make the assumption that the data set will be large enough so that there will be at least 1,000,000 records for each processor, we can simplify these equations in order to examine system balance.

The network time consists of time for starting up a message and time for the data to pass through the bisectional bandwidth of the network. When $N = 1,000,000p$, the startup time and transfer time are equal if $p\beta_s = \frac{N}{k}\beta_r = \frac{1,000,000p}{k}\beta_r$. The message startup time does not dominate the transfer time until $\beta_s > \frac{1,000,000}{k}\beta_r$. Even with the high latency of TCP traffic over ethernet the ratio is closer to $\beta_s = 2000\beta_r$. Assuming that the bandwidth component of the network time dominates, the network and CPU time for phase two are balanced when

$$\begin{aligned} T^{phase2}_{sort+merge} &= T^{OVS}_{net} \\ \Rightarrow \frac{N}{p}\log\left(\frac{N}{kp}\right) + \frac{\alpha N}{p}\log(p) &= N\beta_r \\ \Rightarrow \beta_r &= \frac{\frac{N}{p}\log\left(\frac{N}{kp}\right) + \frac{\alpha N}{p}\log(p)}{N} \\ &= \frac{1}{p}\left(\log\left(\frac{N}{kp}\right) + \alpha\log(p)\right). \end{aligned}$$

The disk time can also be divided into startup and transfer time. Our disk startup time D_s includes the seek time and operating system latency to start the transfer. With $N = 1,000,000p$

the disk startup time will not dominate the transfer time until $D_s > (1,000,000/k)D_r$. For fast SCSI disks with 30$m$sec seek time and 3MByte/sec transfer rate $D_s \approx 30m$sec and $D_r \approx$ 100Byte/3Mbytes/sec $\approx 33\mu$sec. For this configuration $D_s \approx 1,000D_r$ which is within our limit of $D_s < 50,000D_r$ for $k = 20$ [7]. Assuming that the transfer time dominates the disk startup time the disk and CPU time for phase two are balanced when

$$\begin{aligned} T^{phase2}_{sort+merge} &= T^{OVS}_{diskread} \\ \Rightarrow \frac{N}{p}\log\left(\frac{N}{kp}\right) + \frac{\alpha N}{p}\log(p) &= \frac{N}{p}D_r \\ \Rightarrow D_r &= \log\left(\frac{N}{kp}\right) + \alpha\log(p). \end{aligned}$$

Phase three of the algorithm overlaps disk and merging operations. This phase will achieve maximum overlap when

$$\begin{aligned} T^{final}_{merge} &= T^{OVS}_{diskwrite} \\ \Rightarrow \frac{\alpha N}{p}\log(k) &= \frac{\alpha N}{p}D_r \\ \Rightarrow D_r &= \log(k) \end{aligned}$$

5.1 Example

A concrete example will aid in understanding the optimal system balance. We have examined the performance of quicksort on a SPARCstation 2. We determined the *Unit* time by dividing the elapsed time for a sort of internal data by the number of comparisons which were counted during the sort. Using a *Unit* time of 2.29 microseconds, 100 processors, $k = 20$, $\alpha = 1.5$ and $N = 100,000,000$

$$\begin{aligned} \beta_r &= \frac{1}{p}\left(\log\left(\frac{N}{kp}\right) + \alpha k\log(p)\right) \\ &= 2.15 \text{ comparisons/tuple} \end{aligned}$$

Assuming a 100 byte tuple, the bisectional bandwidth would need to be approximately 20Mbytes/second. This bandwidth can be achieved in many of the networks currently available on multicomputers.

Using the same assumptions

$$\begin{aligned} D_r &= \log(k) \\ &= 4.3 \text{ comparisons/tuple} \end{aligned}$$

The disk bandwidth necessary for the algorithm to run efficiently on a SPARCstation 2 is 4.5 Mbytes/second. Some current synchronous SCSI implementations have achieved higher bandwidths than this, so this does not seem unreasonable.

5.2 Model

An analytical model was developed to show that the OVS algorithm is practical on existing parallel machines, and to compare the OVS and PSRS algorithms on machines with varying system parameters. A sort of random integer values was performed to determine the exact number of comparisons which occurred for the OVS algorithm. Figure 4 shows the speedup values predicted on 100 processors with a 100,000,000 tuple data set and skew equal to 1.5 as the network and disk speeds vary. The system parameters of several actual machine architectures have been labeled to show the estimated performance of the OVS algorithm on actual machines. All of the machine plots were made assuming SCSI disks on each node with a 3Mbyte/sec transfer speed. Figure 5 shows a close up view of the region of maximum speedup.

Several existing machines are examined to determine architectures where the OVS algorithm is practical. The Intel Paragon is shown to be on the plateau of maximum performance, but it is near the point where severe performance degradation would occur if network transmission time was increased. The network on the Paragon was assumed to have 200Mbyte/sec links configured in a mesh [8]. The IBM SP1 is plotted assuming a 6Mbyte/sec omega network [11]. The Meiko CS-2 has a logarithmic network where bisectional bandwidth increases linearly with the number of processors. The Meiko plot assumes that each node has a 50Mbytes/second link [14]. The IBM SP1 appears to have higher disk transmission times because of the faster CPU speed. Both the network and disk axes are given in terms of the CPU speed. The ethernet plot in Figure 4 pertains to a network of SPARCstation 2 workstations with SCSI disks connected with ethernet. It is obvious that the network bisectional bandwidth must be significantly increased before a network of workstations can efficiently execute the OVS algorithm.

The same sort was modeled using a modified External PSRS algorithm in order to compare the algorithms. PSRS was modified to read data from disk at the beginning of the algorithm and then to overlap writes to disk and the merging operation at the end of the algorithm. Figure 6 compares the speedup results for the two algorithms. OVS is shown to have higher speedup values on architectures with reasonable levels of system balance. The total number of comparisons for the OVS algorithm was less than the PSRS algorithm (5273 million compares vs 7492 million compares). The lower number of comparisons, coupled with the increased level of overlap leads to the more favorable speedup curve for the OVS algorithm.

Proper system balance is important to achieving acceptable speedup. Modeling algorithmic behavior can help system designers to know what effect their design decisions will have on the efficiency of an algorithm family. The overlap sorting algorithm should perform well on a wide variety of parallel systems.

6 Conclusions

Many parallel sorting algorithms have been developed which give varying levels of performance on parallel computers. Our overlapped sorting algorithm is more amenable to internal sorting of external data than many existing algorithms because it allows computations, network com-

Number of Processors	Number of samples	Percent of Network Time	Percent of Disk read	Percent of CPU Time
8	26	0.003%	0.21%	.006%
16	35	0.004%	0.28%	.025%
32	43	0.004%	0.35%	.076%
64	50	0.005%	0.40%	.211%
128	58	0.006%	0.47%	.573%
256	65	0.006%	0.53%	1.469%
512	72	0.007%	0.58%	3.66%
1024	79	0.008%	0.64%	8.929%

Table 3: Percentage of resources used during the sampling phase for $N = 1000000 * p$, $\alpha = 1.5$, confidence level = 90%, 80 records per page.

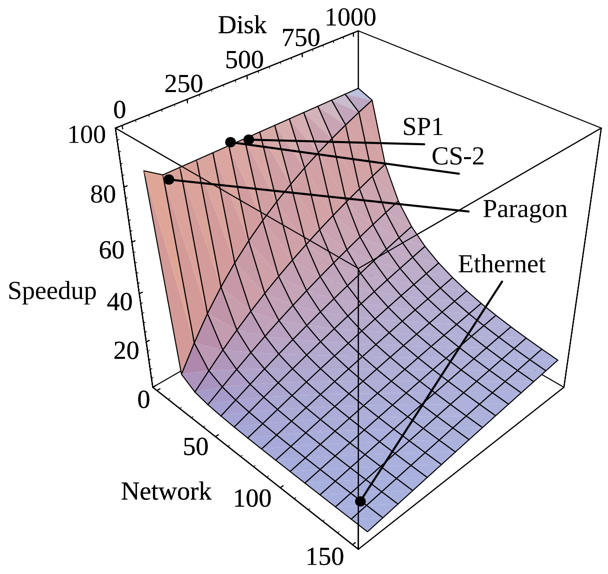

Figure 4: Predicted speedup results for sorting 100 million records with the OVS algorithm on 100 processors as disk and network speeds vary. The Disk and Network axis are given in number of *Units* (comparison times) to transfer a 100 byte record. Skew is assumed to be 1.5. Points are plotted for the Intel Paragon, IBM SP1, Meiko CS-2 and a network of SPARCstation 2 workstations connected by ethernet.

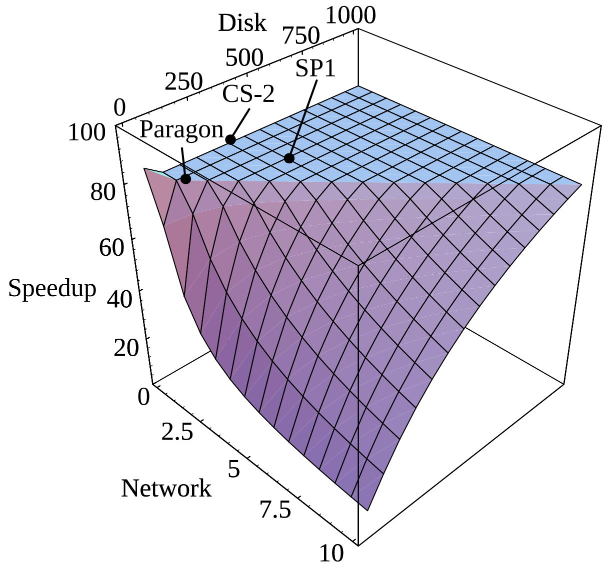

Figure 5: Closer view of the maximum speedup plane from Figure 4. The Disk and Network axis are given in number of *Units* (comparison times) to transfer a 100 byte tuple. Skew is assumed to be 1.5. Points are plotted for the Intel Paragon, IBM SP1 and Meiko CS-2.

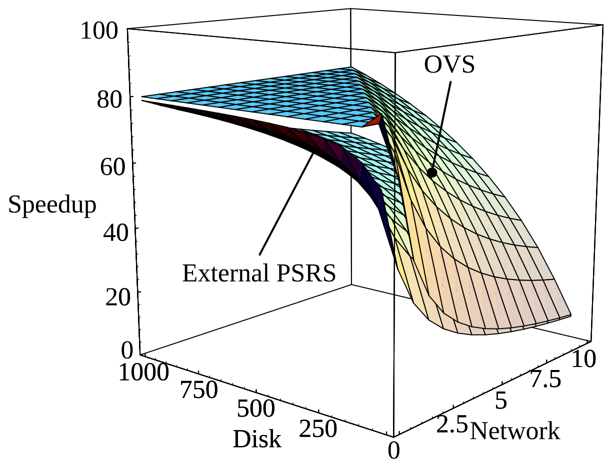

Figure 6: Predicted speedup for sorting 100 million records on 100 processors with the PSRS and OVS algorithms as disk and network speeds vary. The OVS algorithm holds a clear advantage on systems where the network, disk, and CPU speeds are well balanced.

munications and disk I/O to occur simultaneously. Overlapping the activities of these three subsystems improves performance.

We have shown that the complexity of this algorithm compares favorably with other efficient parallel sorting algorithms and have developed a computational model for the execution time of the new sorting algorithm. The overlapped sorting algorithm is sensitive to imbalance in system parameters; we have analyzed how variance in these system parameters affects performance.

The overlapped sorting algorithm provides a new way of taking advantage of the parallel disk, network and CPU subsystems on contemporary multicomputers.

References

[1] Akl, S. G. . *Parallel Sorting Algorithms.* Academic Press, Orlando, FL, 1985.

[2] Beck, M. , Bitton, D. , and Wilkinson, W. K. . Sorting large files on a backend multiprocessor. *IEEE Transactions on Computers*, 37(7):769–778, July 1988.

[3] Blelloch, G. E. , Leiserson, C. E. , Maggs, B. M. , Plaxton, C. G. , Smith, S. J. , and Zagha, M. . A comparison of sorting algorithms for the connection machine CM-2. In *Proceedings of the 3rd Annual ACM SPAA*, 1991.

[4] Cormen, T. H. , Leiserson, C. E. , and Rivest, R. L. . *Introduction to Algorithms.* McGraw-Hill, New York, 1990.

[5] DeWitt, D. J. , Naughton, J. F. , and Schneider, D. A. . Parallel sorting on a shared-nothing architecture using probabilistic splitting. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*, pages 280–291, December 1991.

[6] Evans, D. J. . A parallel sorting-merging algorithm for tightly coupled multiprocessors. *Parallel Computing*, 14(1):111–121, 1990.

[7] Hennessy, J. L. and Patterson, D. A. . *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[8] Intel Corporation. *Paragon OSF/1 C Compiler User's Guide*, January 1993.

[9] Knuth, D. E. . *Sorting and Searching.* Addison-Wesley, Reading, Massachusetts, 1973.

[10] Lakshmi, M. S. and Yu, P. S. . Effect of skew on join performance in parallel architectures. In *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems*, pages 107–120, December 1988.

[11] Leighton, F. T. . *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1992.

[12] Leopold, C. . A fast sort using parallelism within memory. In *Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, pages 326–333, 1992.

[13] Lorie, R. A. and Young, H. C. . A low communication sort algorithm for a parallel database machine. In *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, pages 125–135, 1989.

[14] Meiko World Incorporated. *Computing Surface 2 Overview Documentation Set*, 1993.

[15] Quinn, M. J. . Parallel sorting algorithms for tightly coupled multiprocessors. *Parallel Computing*, 6(3):165–177, March 1988.

[16] Seshadri, S. and Naughton, J. F. . Sampling issues in parallel database systems. In *Proceedings of the 3rd International Conference on Extending Database Technology*, pages 328–343, March 1992.

[17] Shi, H. and Schaeffer, J. . Parallel sorting by regular sampling. *Journal of Parallel and Distributed Computing*, 14(4):361–372, April 1992.

[18] Wilson, G. V. . A glossary of parallel computing terminology. *IEEE Parallel and Distributed Technology*, 1(1):52–67, February 1993.