iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 **Issue:** IV **Month of publication:** April 2018

DOI: http://doi.org/10.22214/ijraset.2018.4088

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue IV, April 2018- Available at www.ijraset.com

An Overview and analysis External Equal Sorting algorithm

Gajraj Singh Pandya[1]
[1,] *Research Scholar, Faculty of Computer Science, Swami Vivekanand University, Sagar(M.P.)*

***Abstract:** This paper introduced the problem of external equal sorting. External sorting involves sorting more data than can fit in the combined memory of all the processors on the machine. It consists of using the disk as a form of secondary memory, and it presents ome fascinating challenges because of the vast difference between the bandwidths and latencies of memory and disk systems. It is an inefficient way to normalize data regarding computational complexity. But it is an entirely right way regarding software engineering because the availability of easy to use sorting routines in subroutine libraries makes the amount of coding required very small. I had already used this technique in the serial version of the speech recognition system, so it seemed natural to do the same for the parallel version. This presents efficient algorithms for external Equal sorting and remote data update.*
***Keyword:** equal sorting , Partitioning, external sorting*

I. EQUAL DISK ALLOCATION SYSTEMS

There is a large variety of different disk I/O systems available for equal computers. In some systems, each CPU has its private drive, and in others, there is a central bank of disks accessible from all CPUs. For this work I assumed the following setup: a distributed memory equal computera equal file system where all CPUs can access all the data The particular setup I used was a 128 cell AP1000 with 32 disks and the HiDIOS equal file system [Tridgell and Walsh 1996].The requirement of a file system where all CPUs ca access all the data isn't as much of an impediment as it might first seem. Almost any distributed file system can quickly be given this property using a remote access protocol coupled to the message passing library1.

1The AP1000 has only local disk access in hardware. The operating system provides remote disk access.100000,10000,1000,100,10,1,0.1

1 10 100 1000 10000 100000 1e+06 1e+07

II. DESIGNING AN ALGORITHM

An external equal sorting algorithm has design constraints entirely different from an internal algorithm. The main differences are: the average number of I/O operations per element needs minimize the I/O requests need to be gathered as far as possible into large requests the algorithm should, if possible, run-in-place to allow data sets up to the full size of the available disk space The interesting difference that the group of I/O requests can make demonstrated by the write throughput results. The graph shows the aggregate write throughput on a 128 cell AP1000 with 16 disks2. With the performance varying by four orders of magnitude depending on the size of the individual writes it is clear that an algorithm that does a smaller number of larger I/O requests is better than one which does a large number of small claims. Some events in this chapter are for a system with 32 disks, and others are for a system with 16 drives. The machine I used was reconfigured part way through my research.

III. CHARACTERIZING EQUAL SORTING

An external sorting algorithm needs to make extensive use of secondary memory (usually disk). In this chapter, I will characterize the external sorting using a parameter p where p = A where (where letA=DMN)is the number of elements b sort and M is the number of items that can fit in the internal memory of the machine. It Means that P equal to 1 implies internal sorting (as the data can provide in memory) and any value of P more significant than 1 suggests external sorting. P is rarely huge as sizeable equal disk systems are usually attached to computers with reasonably large memory subsystems. On the AP1000 available to me P is limited to 8 (a total of 2GB of RAM and 16GB of the disk). To simulate the effect of more significant value for P some of the test runs use a deliberately limited amount of memory.

IV. LIMITATIONS ON INPUT AND OUTPUT

Lower limits on the computational requirements for equal external sorting are the same as those for internal sorting. It is, however, illuminating to look at the smaller restrictions on I/O operations required for external sorting as quite precisely the number of I/O

operations needed will impact significantly on the performance of the algorithm. The most apparent lower limit is at least one read and one write per element. It Stems from the fact that every aspect has to be moved at least once to change the order of the items. Can this limit be achieved? It is relatively easy to show that it cannot. Imagine we had an algorithm that performed only one read and one write per element. As the items cannot predict in advance, we cannot place any part in its correct output position with certainty until all aspects have read. That implies that all the reads must perform before all the writes. In that case, the algorithm would have to store internally (in memory) enough information to characterize the ordering of the elements. The memory required to do this for non-compressible data is at least log(N!) bits, which is approximately N log(N). With k > 1 this cannot be achieved3. The algorithm presented in this chapter demonstrates that external equal sorting can, however, be done with an average of approximately two reads and writes per.3Strictly it could be done if the size of each element is more significant thank log N.element, at least for values of k that are practical on the machines available to me.

V. OVERVIEW OF THE ALGORITHM

The external sorting algorithm works by mapping the file onto a 2-dimensional k × k grid in a snake-like fashion4. The columns and rows of the network are then alternately sorted using the internal equal sorting algorithm described above. It can be demon-strated [Schnorr and Shamir 1986] that the upper limit on the number of column and row sorts required is d log key + 1. This basic algorithm augmented with some optimizations which significantly improve the efficiency of the algorithm in the average case. In particular, the grid squares further divided into slices which contain enough elements to fill one processor. The top and bottom component of each slice are kept in memory allowing the efficient detection of slices for which all items are in their correct final position. In practice, this means that most slices completely sorted after just one column and row sort, and they can ignore in the following operations The algorithm also uses dynamic allocation of processors to slices, allowing for an even distribution of work to processors as the number of unfinished slices decreases. The proof that the algorithm does indeed sort and the upper limit on the number of column and row sorts comes from the sheer sort algorithm[Schnorr and Shamir 1986].

VI. PARTITIONING

The external sorting algorithm starts off by partitioning the file to be sorted into a two-dimensional grid of size k × k. Each grid square is further subdivided into some slices so that the number of elements in a slice does not exceed the amount that can hold in one processor's memory. An example of this partitioning shown in Figure 2.2 for k = 3 and P = 6. Each slice in the figure labeled with two sets of coordinates. One is its slice number, and the other is its (row, column) coordinates. The slice number uniquely defines the slice whereas the (row, column) coordinates shared between all slices in a grid square.

VII. COMPLETION PROCESS

An important part of the algorithm is the detection of completion of the sort. Although it would be possible to use the known properties of shearsort to guarantee comple-tion by just running the algorithm for dlog *k* + 1e passes, it is possible to improve the average case enormously by looking for early completion. Early completion detection is performed on a slice by slice basis, rather than on the whole grid. Completion of the overall sorting algorithm is then defined to occur when all slices have been completed. The completion of a slice is detected by first augmenting each slice number with a copy of the highest and lowest element in the slice. The last processor to write the slice holds these elements. At the end of each set of column or row sorts these sentinel elements are then gath-ered in one processor using a simple tree-based gathering algorithm. This processor then checks to see if the following two conditions are true to determine if the slice has completed: the smallest element in the slice is larger than or equal to the largest element in all preceding slices; and the largest element in each slice is smaller than or equal to the smallest element in each of the following slices. If these two conditions are true then all elements in the slice must be in their correct final positions. In this case the elements need never be read or written again and the slice is marked as finished. When all slices are marked as finished the sort is complete.

VIII. PROCESSOR ALLOCATION IN A MATHEDOLOGY

As slices are marked complete the number of slices in a row or column will drop below the number of processors *P*. This means that if a strictly serialized sorting of the rows or columns was made then processors would be left idle in rows or columns which have less than *P* slices remaining. To take advantage of these processors the allocation of slices to processors is made dynamically. Thus while one row is being sorted any unused processors can begin the task of sorting the next row. Even if too few unused processors are available to sort the next row a significant saving can be made because the data for the slices in the

next row can be loaded, thus overlapping I/O with computation. This dynamic allocation of processors can be done without additional communi-cation costs because the result of the slice completion code is made available to all processors through a broadcast from one cell. This means that all processors know which slices have not completed and can separately calculate the allocation of proces-sors to slices.

IX. ALGORITHM PERFORMANCE

The problem with measuring the speed of external sorting is finding enough disk space. The AP1000 has 2GB of internal memory, which means you need to sort con-siderably more than 2GB of data to reach reasonable values of *P* Unfortunately only 10GB (of a possible 16GB) is allocated to the HiDIOS file system, so a *P* above 5 is not possible when the full internal memory of the machine is used. In practice a *k* of above

X. WORST CASE

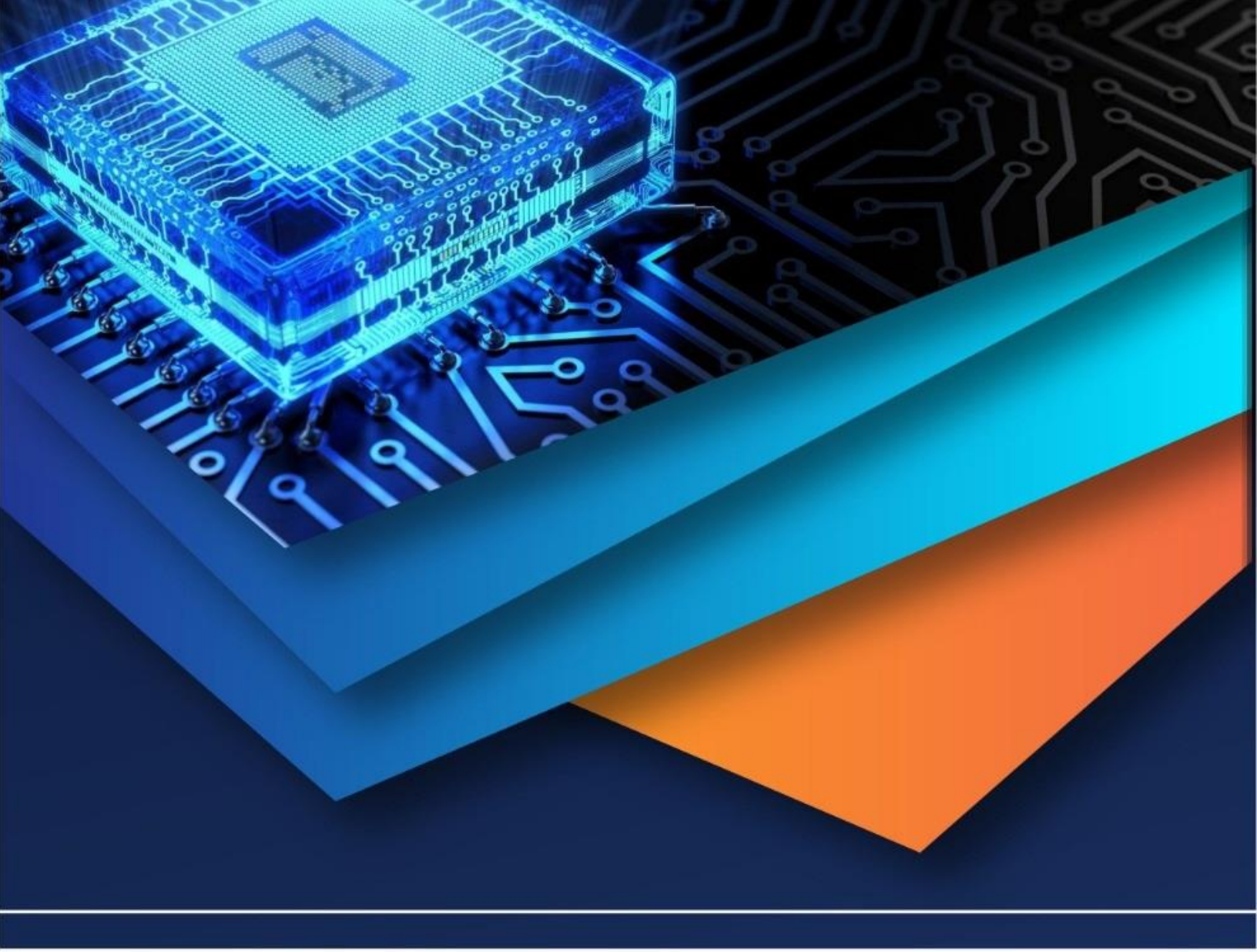
An essential property of a sorting algorithm is the worst case performance., Ideally, the worst case is not much worse than the standard fact, but this is hard to achieve. The worst case for the external sorting algorithm is when the data starts out sorted by the column number that assigns after the partitioning stage. It will mean that the first column sort will not achieve anything. The worst case of log k + 1 passes can then made. Additionally, the data distribution required for the worst example of the internal sorting algorithm should use.

XI. CONCLUSIONS

The external equal sorting algorithm presented in this chapter has some quite strong similarities to the internal equal sorting algorithm from the previous section. Both get most of their work done with a first pass that leaves the vast majority of elements in their correct final position. In the case of the external algorithm, this first pass is the natural first step in the overall algorithm whereas for the internal algorithm the first pass is logically separated from the cleanup phase. The external algorithm also makes very efficient use of the disk subsystem by using I/O operations that are of a size equal to the internal memory size of each of the processors. It makes an enormous difference to the practical effect of the algorithm.

REFERENCES

[1] AKL, S. G. 1985. Parallel Sorting Algorithms. Academic Press, Toronto. (p. 47) BATCHER, K. E. 1968. Sorting networks and their applications. In Proc. AFIPS Spring Joint Computer Conference, Volume 32 (1968), pp. 307 – 314. (pp. 7, 14) BELL, T., CLEARY, J., AND WITTEN, I. 1990. Text Compression. Prentice Hall.(p. 86
[2] BLELLOCH, G. E., LEISERSON, C. E., MAGGS, B. M., PLAXTON, C. G., SMITH, S. J., AND ZAGHA, M. 1991. A comparison of sorting algorithms for the connection machine CM-2. In Proc. Symposium on Parallel Algorithms and Architectures (Hilton Head, SC, July 1991). (p. 9
[3] BURROWS, M. AND WHEELER, D. 1994. A block-sorting lossless data compression algorithm. Technical Report SRC Research Report 124 (May), Digital Systems Research Center. (p. 7
[4] CALLAGHAN, B. 1998. Network file system version 4.(p9
[5] ELLIS, J. AND MARKOV, M.1998.A fast, in-place, stable merge algorithm.(p. 6
[6] FENWICK, P. 1996. Block sorting text compression. In Proc. 19th Australasian Computer Science Conference, Melbourne, Australia (January 1996). (pp. 88, 90
[7] FOX, G. C., JOHNSON, M. A., LYZENGA, G. A., OTTO, S. W., SALMON, J. K., AND WALKER, D. W. 1988. Solving Problems on Concurrent Processors, Volume 1. Prentice-Hall. (pp. 10, 15, 21
[8] GAILLY, J. AND ADLER, M. 1998. zlib. (pp. 55, 75, 86
[9] GNU.1998.The GNU project GUTENBERG. 1998. Project gutenberg. (p. 79) HELMAN, D., BADER, D., AND JA`JA`, J. 1996. A randomized parallel sortin
[10] algorithm with an experimental study. Technical Report CS-TR-3669, Institute for Advanced Computer Studies, University of Maryland. (p. 32)
[11] HOFF, A. AND PAYNE, 1997.Generic diff format specification.(p. 93
[12] HUANG, B. AND LANGSTON, M. A.1988. Practicalin placemerging. Communications of the ACM 31, 348 – 35
[13] ISHIHATA, H., HORIE, T., INANO, S., SHIMIZU, T., KATO, S., AND IKESAKA,
[14] 1991. Third generation message passing computer AP1000. In International Symposium on Supercomputing (November 1991), pp. 45 – 55. (pp. 3, 10
[15] ISHIHATA, H., HORIE, T., AND SHIMIZU, T. 1993. Architecture for the AP1000 highly parallel computer. Fujitsu Sci. Tech. J. 29, 6 – 14. (p. 1
[16] KARP, R. AND RABIN, M. 1987. Efficient randomized pattern-matching algorithms. IBM J. Research and Development 31, 249 – 260. (p.5
[17] KNUTH, D. E. 1981. The Art of Computer Programming, Sorting and Searching. Addison-Wesley. (pp. 4, 7, 10, 21
[18] RONROD, M. A. 1969. An optimal ordering algorithm without a field of operation (in Russian). Dokl.Akad. Nauk SSSR 186, 1256 – 125
[19] LEACH, P. AND NAIK, D. 1998. Common internet filesystem protocol.. (p. 9
[20] MACDONALD, J. 1998. Versioned file archiving, compression and distribution. (p. 84
[21] MCILROY, D., MCILROY, P., AND BOSTI, K. 1993. Engineering radix sort Computing systems 6, 1, 5 – 27. (pp. 32, 43)



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY