On The External Merge Sorting With Large Main Memory

Hwang-Kyu Choi

대용량 주기억장치를 이용한 외부 합병정렬 방법에 관한 연구

최 황 규

요 약

본 논문은 대용량 주기억장치를 갖는 컴퓨터 상에서 효율적으로 수행 될 수 있는 외부 합병정렬 방법에 대하여 기술한다. 제시된 정렬 방법은 주기억 장치의 용량이 정렬될 화일 크기의 제곱근보다 크다는 조건하에서, 주기억 장치를 최대로 이용하여 외부 합병정렬에 소요되는 외부 합병의 횟수를 최소화 함으로써 외부 합병정렬의 성능에 가장 큰 영향을 미치는 입출력 시간을 크게 줄일수 있음을 보였다.

1. Introduction

For sorting a disk file that is too large to fit in main memory, the external merge sort[2] has been generally used. This method consists of essentially two distinct phases. First, segments of the input file are sorted using a good internal sort method, such as Quicksort[4], and then these sorted segments, called runs, are written out into disk. Second, the runs are divided into groups, and then the runs within the same group are merged together to form a larger run. This process is repeated until only one run is left. The external merge sort is I/O bound because it may require more than one merge pass in the merge phase, in which the whole file should be read from and written out into the disk, while the total CPU operations are nearly equivalent to the case of its internal sort whose comparison complexity is $O(n\log_2 n)$ for n elements to be sorted. Therefore, the I/O complexity in the merge phase is more important factor that limits the performance of the external merge sort for large files.

Most of the algorithms and analyses on external sorts are based on the assumption that the file is order magnitude much larger than main memory available, resulting in the I/O bottleneck. Thus these mainly focus on minimizing the I/O accesses. In(2), the optimal merge sequence minimizing the

number of the merge passes was extensively discussed, and it can be achieved by choosing the merge order as large as possible. On the other hand, Kwan and Baer[3] point out that selecting the merge order as large as possible is not optimal under the assumption of a more sophisticated model of the disk. But they also notice that if we assume the generalized disk model where each disk access taskes contant time, then choosing the largest merge order would be optimal. In this situation, for merging the runs with the merge order k, i.e., k-way merge, k blocks of the main memory are needed, one for each run. Therefore, as the main memory size becomes larger, the merge order increases and then the I/O accesses may be reduced.

In this paper,we consider the external merge sort problem under the assumption of a large main memory available, yet not larger than the file size, for relaxing the I/O bottleneck. We basically make the assumption that computer systems have the main memory sufficient to sort a file compeletly with only one merge pass in addition to the run generation phases. For this purpose the available main memory must be more than the *square root* of the file size (measured in blocks). Recent medium to high-end computer systems typically have memory capacities in the range of 16 to 128 megabytes, and it is projected that chip densisties will continue their current trend of doubling every year for the foreseeable future[1]. In a later section, we will show the minimum sizes main memory to satisfy our assumption for large files, and also show that these sizes are reasonabley small, which can be supported by the above computers described. Our objective, under this assumption, is to utilize the main memory effectively for reducing I/O accesses as much as possible by saving some fraction of the internally, instead of the disk.

2. Memory Requirement

When sorting N blocks of a file with M blocks of the main available, where $2 \leq M < N$, the merge order of M, i.e., M-way merge, is possible, and then the number of the merge passes needed is $\lceil \log_M R \rceil$, where R is the number of the initial runs, i.e, $\lceil \frac{N}{M} \rceil$ if M blocks of the main memory are allocated for internal sorting. Therefore, in order to merge completely with only one pass it needs more than $\sqrt{N}$ blocks of the main memory. Table 1 illustates the required memory sizes to satisfy this condition with varying the sizes of the file, when the block size is assumed to one kilobytes and 4 kilobytes. In the table, it is ture, for example, that one megabytes of the main memory can sort 256 megabytes of file when 4 kilobytes of the block size is assumed, and also proportionally two mergabytes main memory can sort one gigabytes file. Therefore, our assumption is resonable for very large files that are several order of

Table 1. The mainmemory sizes versus the sizes of the file to sort with one merge pass.

The main memory size(M)	The file size(N)	
	1kilobytes / block	4kilobytes / block
64KB	4MB	1MB
128KB	16MB	4MB
256KB	64MB	16MB
512KB	256MB	64MB
1MB	1GB	256MB
2MB	4GB	1GB

* KB(*Kilobytes*) MB(*Megabytes*) GB(*Gigabytes*)

magnitude larger than the main memory.

Under our assumptiom $\sqrt{N} \leq M < N$, the external merge sort is completed with only one merge pass in addition to the run generation phase. If there are more than $\sqrt{N}$ blocks of the main memory, then the excess memory not used in the merge phase exists, because the number of the initial runs, $R\left(\left\lceil \frac{N}{M} \right\rceil\right)$, is $R < \sqrt{N} < M$ and thus R blocks of the main memory are only needed to merge the runs. This excess memory can be used to save some fraction of the file, at least $M-R$ blocks, during the run generation phase. As a result, the number of blocks to be stored in the disk may be reduced and thus the number is reduced accordingly. Therefore, the number of the main memory blocks required to merge the runs is also reduced. The number of the runs(or equivalently the main memory blocks to merge the runs) to be stored in the disk is expressed by the following proposition.

Proposition. *For sorting N blocks of the file with M blocks of the main memory, where $\sqrt{N} \leq M < N$, the minimum number of the main memory blocks to merge the runs to be stored in the disk is*

$$R' = \left\lceil \frac{N-M}{M-1} \right\rceil$$

Proof. The R' is simply derived from following equation, $R' = \left\lceil \frac{N-(M-R')}{M} \right\rceil$.

Therefore, the total number of blocks to be saved into the main memory is $M - R'$

3. Algorithm

The external merge sort algorithm, using more than $\sqrt{N}$ blocks of the main memory, can be simply stated as follows:

Step 1. $N-(M-R')$ blocks of the file are constructed to runs each of which has M blocks. Possiblely the last run may have less than M blocks;

Step 2. The remaining blocks, $M-R'$ blocks, are loaded into part of the main memory which is not to be used for merging, and then these blocks are sorted internally as a run;

Step 3. The $(R'+1)$ runs, where one run resides in the main memory and others are stored in the disk, are merged together into a single sorted run with one merge pass.

4. Performance and Discussions

In the performance evaluation of our sorting algorithm, we will only consider the I/O performance because the computing complexity is nearly the same as that of its internal sorting. Our algorithm requires at least the two I/O accesses of the whole file, the initial reading and the final writing, as any other algorthms. Thus we exclude these two I/O accesses in the performance consideration.

In our algorithm, the performance benefits mainly occur in writing the runs during the run generation phase and reading the runs during the merge phase. This benefit is the ratio of the number of blocks saved in the main memory to the number of the total blocks in the file, represented by the following *benefit factor*

$$Q = \frac{M-R'}{N}$$

Then the total number of the I/O accesses reduced is turn out to be $2(1-Q)N$.

Table 2 illustrates the performance benefits of our algorithm for sorting 10 megabytes of file, varying the main memory size. We assumed the block size to 4 kilobytes. Thus the file has 2500 blocks, i.e., N=2500, and then the main memory size, M is varied from $\sqrt{N}$ to N blocks. In the table, the number of the runs stored in the disk is decreased as the main memory size increases, and, accordingly, the benefit factor is increased. As a result, the number of the I /O access is linearly decreased as the main memory size incrases.

In conclusion, our algorithm can be used to sort very large files more efficiently, using reasonablely small size of the main memory which can be supported by recent conventional computers, as long as $\sqrt{N} \leq M < N$ is satisfied. In addition, the main memory requirement to merge with one pass can be reduced to half by using the replacement selection technique[2], which can make runs twice as large as the main memory capacity, instead of the conventional sorting methods that generate runs only as large as the main memory size.

Table 2. The performance of the sort algorithm for 10 megabytes of file(N=2500), varying the main memory size(M).

The number of the main memory blocks (M)	The number the runs in the disk (R')	The performance benefit factor (Q)	The number of the I/O accesses $2(1-Q)N$
50(0.2)	50	0.000	5000
100(0.4)	25	0.030	4850
300(1.2)	8	0.117	4416
500(2.0)	5	0.198	4010
800(3.2)	3	0.319	3406
1000(4.0)	2	0.399	3004
1200(4.8)	2	0.479	2604
1400(5.6)	1	0.560	2202
1600(6.4)	1	0.640	1803
1800(7.2)	1	0.720	1402
2000(8.0)	1	0.800	1003
2100(8.4)	1	0.840	802
2200(8.8)	1	0.880	602
2300(9.2)	1	0.920	403
2400(9.6)	1	0.960	202
2500(10.0)	0	1.000	0

* The numbers in the parentheses denote the main memory sizes in megabytes.

4. References

1. M. Fishetti, Technology '86 : solid state, IEEE Spectrum 23 (3) (1986).
2. D. E. Knuth, The Art of Computer Programming, Vol. 3 : Sorting and Searching (Addision-Wesley, Reading, MA, 1973).
3. S. C. Kwan and J. L. Baer, The I/O performance of multiway merge-sort and tag sort, IEEE Trans. Comput. 34(3) (1985) 383−387.
4. R. Sedgewick, Implementing quicksort programs, Comm. ACM 21(1978) 847−857.