Single Versus Double Buffering in Constrained Merging

William E. Wright
Department of Computer Science, Southern Illinois University at Carbondale, Carbondale, Illinois, USA

An analysis is made of a number of aspects of the buffering scheme during the merge phase of an external sort. A discussion is given of the problem of choosing between complete double buffering, complete single buffering, or something in between, assuming a constraint on the amount of internal memory available for buffering. It is shown that single buffering degrades processing time for a merge pass by an average of from 25% to 37%. It is shown that single buffering is faster than double buffering if and only if there is space available for fewer than 16 buffers.

DOUBLE BUFFERING

During the merge phase of an external sort, it is necessary to read in strings from several physical or logical units of external storage, perform merging of the strings in internal storage, and write out longer strings on other external storage units. While the data is in main memory, it is stored in buffers—input buffers for data which has been read, and output buffers in which to store the merged strings in preparation for writing. Using a merge of order $m \geq 2$, it is necessary at any one time to have at least m full input buffers in memory, one for each unit. On the other hand, only one available output buffer is necessary at any given time, because merged strings are prepared for only one output unit at a time.

Assuming that it is possible to read from only one unit at a time and write onto only one unit at a time, the technique of double buffering can greatly reduce the elapsed time during the merge by permitting a high degree of overlapping between input, output and internal processing. Double buffering of input permits one full input buffer to be available for internal processing while the other buffer is being filled (read). Double buffering of output permits one output buffer to be available for internal processing while the other full buffer is being emptied (written).

Since only one output unit is being served at any given time, only two output buffers are needed to provide double buffering and the maximum possible overlapping of output (this statement is qualified and clarified in the section on single buffering). However, since there must be m full input buffers in order for processing to continue, the maximum possible overlapping of input can be achieved only if there are at least $2m$ input buffers. In fact, the $2m$ input buffers must be filled according to an algorithm known as forecasting with floating buffers,[1] which looks at the key of the last record in each full input buffer to determine which input unit to read from next. It has been shown that if fewer than $2m$ input buffers are used, or if $2m$ buffers are used but the forecasting algorithm is not followed, then maximum overlap cannot be guaranteed.

If two buffers are used for output and $2m$ for input, then maximum overlapping of input, output and internal processing can be obtained (assuming that internal processing is faster than input and output). In this case, input and output are continuous, with the next output buffer always being full when each write is completed, and an input buffer always being empty when a read is completed. At the beginning of a merge pass only reading will occur while the first m input buffers are read and then the first output buffer is filled. At the end of a pass only writing will occur after the last input buffer has been filled. Accordingly, the time required for a merge pass will be the time to write (or read) the file once at full speed, plus the time to read m input buffers, plus the time to fill (internally) one output buffer. The nonoverlapped time at the beginning (or end) of a pass is usually insignificant in comparison with the rest of the time, so that the time for a merge pass is generally assumed to be the time to write (or read) the file at full speed.

LESS THAN DOUBLE BUFFERING

While double buffering is extremely valuable in most circumstances, it is not necessarily appropriate for certain 'constrained' environments which are less than ideal. One problem is that double buffering obviously requires more main memory than single buffering, assuming the same buffer size and merge order. Main memory may be of little concern in a large computer system, but it can be of critical importance in a mini- or microcomputer environment.

It would, of course, be possible to achieve double buffering in the same memory size as single buffering by making the buffers only half as large. In some circumstances this would be quite reasonable, but in other circumstances the disadvantages might outweigh the advantages. One disadvantage is that the access time per bit might increase, due to a constant seek and/or latency time for each access which is independent of the block (and buffer) size. Another problem is that there may be a physical minimum block size that is a function of the device, so that the buffer size cannot be smaller than this minimum. An example would be a sector on a sectored disk (e.g. 256 bytes) or a block on a magnetic bubble memory system (e.g. 144 bytes).

It would also be possible to achieve double buffering in the same memory size as single buffering, and with the same buffer size, by reducing the merge order by a factor of 2. For example, instead of doing 20-way merging with

CCC-0010-4620/82/0025-0227 $02.00

single buffering, do 10-way merging with double buffering. The advantages and disadvantages are apparent: double buffering permits overlapping of input, output and internal processing so that the pass is faster, but the lower merge order will quite likely result in more passes.

It is not obvious as to what choice should be made. An example should illustrate the problem. Suppose that at the beginning of a merge pass there are 14 physical strings residing on a single random access device such as a disk or magnetic bubble memory system. Suppose also that there are 5000 bytes of main memory space available (assume a minicomputer environment) for the input buffers. Assuming that the physical block size for the device is 256 bytes or a multiple of 256, then there is enough main memory for 19 single block buffers.

Completing the sort in one merge pass would require 14-way merging. This would in turn require 28 input buffers if double buffering were used. This design is thus infeasible since the maximum number of buffers is 19. Therefore two passes will be required if double buffering is used. Since $\sqrt{14} \simeq 4$, four-way merging could be used for each pass. Eight buffers would be required, either all having a size of two blocks (with three blocks left over), or five with a size of two blocks and three with a size of three blocks. The time required for the pass would be the time required to read the file once.

An alternative design would be to go ahead with 14-way merging so as to finish in one pass, and not use double buffering. One technique would use forecasting with 19 single-block floating buffers.[2, 4] The merge pass would be longer since overlapping is not complete, but there would only be one such pass.

Still another scheme would also do 14-way merging, but this time with just 14 buffers. Nine of the buffers would be 1 block long, and the other five would be 2 blocks long. There would be no overlapping of input and internal processing, since processing would stop when a buffer became exhausted and was being read into. On the positive side, the sort would be completed in only one pass, the larger buffers would take longer to exhaust, and the access time per bit would be smaller for the longer reads.

In addition to these problems relating to the extra main memory requirements for double buffering, there is another problem that greatly reduces the benefits of double buffering under some circumstances. The problem is that double buffering loses most of its advantage if input and output cannot be overlapped with each other. For example, if the merge is being performed on a single disk device, divided into the various logical units, then input and output cannot occur simultaneously. Even if there are two or more devices but a single controller, input and output cannot occur simultaneously. (It might be possible to do independent seeking, but this does not require double buffering.) Even in the case where there are multiple controllers but all connected to a single channel, input and output cannot occur simultaneously.

Hence we see that in many (very common) situations, double buffering does not bring about an overlapping of input and output time. It does permit overlapping of input–output with internal processing, but such an overlap can largely be achieved with fewer than $2m$ input buffers, especially if internal procssing is very fast compared to input–output.

SINGLE BUFFERING

From the preceding discussion it is apparent that in many circumstances consideration should be given to performing merges using fewer than $2m$ input buffers (where m is the merge order). Again, there must be at least m buffers, and there is no benefit in having more than $2m$. Let n denote the number of input buffers in excess of m. Then $0 \leq n \leq m$ and the total number of input buffers is $m + n$.

Let $r = (m + n)/m$ i.e. the ratio of the total number of input buffers to the merge order. Then $1 \leq r \leq 2$, complete double buffering corresponds to $r = 2$, and complete single buffering corresponds to $r = 1$. A large value of r has the advantage of a greater overlapping of input, output and internal processing, other things being equal. A small value of r has the potential advantage of a larger buffer size (i.e. slower exhaustion of buffers and faster access time per bit) and a higher merge order (i.e. fewer merge passes), other things being equal.

As noted earlier, a discussion of merging with from $m + 1$ to $2m$ input buffers (i.e. $1 \leq n \leq m$) is given in Refs 2 and 4. An approximate model for the system is presented. We shall present here an analysis (based on some reasonable assumptions) for the special case of $n = 0$, i.e. complete single buffering. We shall show the degradation in time for a complete merge pass, caused by nonoverlapped input, as a function of the merge order. We shall show under what circumstances single buffering is superior to double buffering, assuming a constant memory size. We shall assume that input and output can proceed independently and simultaneously, since the analysis is trivial otherwise. For simplicity, we shall restrict our attention to the case where all input and output buffers are the same size.

Let us begin by defining the random variable X to be the number of input buffers exhausted during the process of filling a single output buffer. Clearly, $X = 0, 1, 2, \ldots, m$. We shall assume that the original data was random, so that the buffers are exhausting in a uniform fashion. We shall also make the simplifying assumption that they are exhausting independently of each other. In reality, the buffer exhaustions are not independent since, for example, two buffers cannot exhaust at exactly the same time. The assumption is quite reasonable, however, especially for large m.

An input buffer can exhaust at most once while an output buffer is being filled, since even if it started with no records, and if all records were selected from it, the output buffer would be filled by the time the input buffer exhausted a second time. Thus the exhaustion or nonexhaustion of an input buffer while the output buffer is being filled is a Bernoulli trial.[3] Since records inserted into the output buffer are being drawn randomly from m input buffers, the probability that any given input buffer will exhaust is $1/m$. Therefore, since the buffers are exhausted independently, the number of buffers to be exhausted (i.e. the random variable X) is a binomial random variable with parameters $1/m$ and m.[3]

Consider the exact moment when we have just filled the previous output buffer, initiated the write, and are ready to begin filling the next output buffer (we assume double buffering on the output). Let the random variable D denote the time required to read or write one buffer. We shall assume that D has a lower bound of $d_0 > 0$ and

a mean (E[D]) of d. Let $D_0, D_1, D_2, \ldots$ have the same distribution as D. We shall let D_0 denote the time required to write the output buffer, and D_i, $i \geq 1$, denote the time to read the ith input buffer to exhaust, assuming that at least i input buffers become exhausted before filling the output buffer.

Let the random variable T denote the time until the write for the next output buffer is initiated. Then

$$\mathrm{E}[T] = \sum_{i=0}^{m} \mathrm{E}[T \mid X = i]\mathrm{P}[X = i] \tag{1}$$

We shall assume that internal processing time is negligible in comparison to input-output time. Thus,

$$\mathrm{E}[T \mid X = 0] = \mathrm{E}[D_0] = d \tag{2}$$

Similarly, for $1 \leq i \leq m$,

$$\begin{aligned}
&\mathrm{E}[T \mid X = i] \\
&\quad = \mathrm{E}[\max\{D_0, D_1 + \cdots + D_i\}] \\
&\quad = \mathrm{E}[D_1 + \cdots + D_i + \max\{D_0, D_1 + \cdots + D_i\} \\
&\qquad - D_1 - \cdots - D_i] \\
&\quad = id + \mathrm{E}[\max\{D_0, D_1 + \cdots + D_i\} - D_1 - \cdots - D_i] \\
&\quad = id + \mathrm{E}[\max\{D_0, D_1 + \cdots + D_i\} - D_1 - \cdots - D_i \mid \\
&\qquad D_1 + \cdots + D_i \geq D_0] \\
&\qquad \mathrm{P}[D_1 + \cdots + D_i \geq D_0] + \mathrm{E}[\max\{D_0, \\
&\qquad D_1 + \cdots + D_i\} - D_1 - \cdots - D_i \mid \\
&\qquad D_0 > D_1 + \cdots + D_i]\,\mathrm{P}[D_0 > D_1 + \cdots + D_i] \\
&\quad = id + \mathrm{E}[D_1 + \cdots + D_i - D_1 \\
&\qquad - \cdots - D_i \mid D_1 + \cdots + D_i > D_0] \\
&\qquad \mathrm{P}[D_1 + \cdots + D_i \geq D_0] + \mathrm{E}[D_0 - D_1 \\
&\qquad - \cdots - D_i \mid D_0 > D_1 + \cdots + D_i] \\
&\qquad \mathrm{P}[D_0 > D_1 + \cdots + D_i] \\
&\quad = id + \mathrm{E}[D_0 - D_1 - \cdots - D_i \mid D_0 > D_1 + \cdots + D_i] \\
&\qquad \mathrm{P}[D_0 > D_1 + \cdots + D_i]
\end{aligned} \tag{3}$$

Combining (1)–(3) we get

$$\begin{aligned}
\mathrm{E}[T] &= d\mathrm{p}_x(0) + \sum_{i=0}^{m} id\mathrm{p}_x(i) + \\
&\quad \sum_{i=1}^{m} \mathrm{E}[D_0 - D_1 - \cdots - D_i \mid \\
&\quad D_0 > D_1 + \cdots + D_i] \\
&\quad \mathrm{P}[D_0 > D_1 + \cdots + D_i]\mathrm{P}_x(i) \\
&= d(\mathrm{p}_x(0) + \mathrm{E}[X]) + \sum_{i=1}^{m} b_i \mathrm{p}_x(i)
\end{aligned} \tag{4}$$

where

$$b_i = \mathrm{E}[D_0 - D_1 - \cdots D_i \mid D_0 > D_1 + \cdots + D_i] \quad \mathrm{P}[D_0 > D_1 + \cdots + D_i] \tag{5}$$

Letting $b = \sum_{i=1}^{m} b_i \mathrm{p}_x(i)$, we get

$$\mathrm{E}[T] = d\left(\left(1 - \frac{1}{m}\right)^m + 1\right) + b \tag{6}$$

The values $b, b_1, \ldots, b_m$ cannot of course be determined more specifically without knowing the distribution of D. Obviously, $\mathrm{P}[D_0 > D_1 + \cdots + D_i]$ decreases as i increases. This probability would usually be very small, since $D_0, \ldots, D_i$ have the same distribution. In many cases it would become 0 very quickly (e.g. $i > 2$). If D is a constant, then $b_i = b = 0$ for $1 \leq i \leq m$, and Eqn (6) becomes

$$\mathrm{E}[T] = d\left(\left(1 - \frac{1}{m}\right)^m + 1\right) \tag{7}$$

Generally, the smaller the variance of D, the smaller the value of b. For magnetic tape devices, D would be essentially constant. For random access devices such as disks, drums, and magnetic bubble memory, D would typically consist of a constant component, corresponding to transfer time, and a random component, corresponding to seek and/or latency time. For disks, transfer time would probably correspond to a fraction of a rotation, or possibly as much as a full rotation. Latency would vary from 0 to a full rotation. Seek time would vary from 0 to a fraction of a rotation to possibly a few rotations.

One technique to reduce or eliminate the influence of b would be to use more than double buffering on output. The extra output buffers could be used to negate the effect of wide fluctuations in access time. Three or four output buffers, for example, would greatly reduce b. With an infinite number of output buffers, E[T] would be given by Eqn (7) (m output buffers would not be sufficient, in theory). Obviously, the use of additional output buffers tends to defeat the purpose of single buffering on input. It nevertheless provides a useful insight into the influence of b on E[T].

Accordingly, we shall ignore b and look at Eqn (7) as a function of the merge order m. Table 1 gives some sample values of E[T] as a function of m, under the various assumptions presented and the additional convention that $d = 1$. Note that

$$\lim_{m \to \infty} \left(1 - \frac{1}{\mathrm{m}}\right)^m = \mathrm{e}^{-1} \simeq 0.36788$$

which gives the bottom entry in the table.

The table suggests that E[T] is an increasing function of m, and we would like to prove this result.

$$\begin{aligned}
d_m(\mathrm{E}[T]) &= d_m\left(\left(1 - \frac{1}{m}\right)^m + 1\right) \\
&= m\left(1 - \frac{1}{m}\right)^{m-1} \frac{1}{m^2} + \left(1 - \frac{1}{m}\right)^m \ln\left(1 - \frac{1}{m}\right) \\
&= \left(1 - \frac{1}{m}\right)^{m-1} \left(\frac{1}{m} + \left(1 - \frac{1}{m}\right) \ln\left(1 - \frac{1}{m}\right)\right)
\end{aligned} \tag{8}$$

Table 1. Expected degradation caused by single buffering

m	E[T]
2	1.250
3	1.296
5	1.328
10	1.349
100	1.366
∞	1.36788$\cdots$

It is obvious that $(1-1/m)^{m-1}$ is positive for $m \geq 2$, therefore we wish to show that the other factor in Eqn (8) is positive for $m \geq 2$.

$$\lim_{m\to\infty}\left(\frac{1}{m}+\left(1-\frac{1}{m}\right)\ln\left(1-\frac{1}{m}\right)\right)=0+1\cdot 0=0 \quad (9)$$

$$\tfrac{1}{2}+\tfrac{1}{2}\ln\tfrac{1}{2}\simeq 0.1534>0 \quad (10)$$

$$d_m\left(\frac{1}{m}+\left(1-\frac{1}{m}\right)\ln\left(1-\frac{1}{m}\right)\right)$$
$$=\frac{1}{m^2}\ln\left(1-\frac{1}{m}\right)<0 \quad \text{for } m\geq 2 \quad (11)$$

Equations (9)–(11) show that this factor is a decreasing function for $m \geq 2$, with an initial value (at $m = 2$) which is positive, and a limiting value (as m approaches infinity) of 0. Hence, $1/m + (1-1/m)\ln(1-1/m) > 0$ for $m \geq 2$. Hence $d_m(\mathrm{E}[T]) > 0$ for $m \geq 2$, and therefore $\mathrm{E}[T]$ is an increasing function of m. Moreover, since $\lim_{m\to\infty}\mathrm{E}[T] = 1-\mathrm{e}^{-1} \simeq 1.36788$, we have shown that this limit is an upper bound for $\mathrm{E}[T]$.

The primary conclusion to be drawn from this development is that the degradation (increase) in run time due to the use of single buffering instead of double buffering is less than 37%. Note that $\mathrm{E}[T]$ would be simply d if double buffering with forecasting were used for input. (To be precise, the expectation would be somewhat larger than this, just as for single buffering, unless the access times were constant or there were an infinite number of output buffers.) The random variable T, of course, represents the time to process only one buffer of output, but extending the expectation to the entire merge pass would yield the same coefficient $(1-1/m)^m + 1$.

SINGLE VERSUS DOUBLE BUFFERING

An obvious question arising from this development is under what circumstances single buffering would be preferable to double buffering. Let us accordingly look at the specific case in which main memory available for input buffering is constant, and a choice must be made between double buffering with m buffers (and m-way merging), or single buffering with $2m$ buffers (using $2m$-way merging). We assume that the merge passes are repeated until the sort is complete. Obviously, double buffering will produce faster passes, and single buffering will produce fewer passes.

Let n be the number of strings to be merged, let $\mathrm{p}(x)$ be the number of passes required using x-way merging, let R be the expected time required to write (read) the file once at full speed, and let $S(x)$ be the expected total run time for the merge using x-way merging. Then we get

$$\begin{aligned}\mathrm{p}(2m) &= \lceil \log_{2m} n\rceil\\ \mathrm{p}(m) &= \lceil \log_m n\rceil\\ &= \lceil \log_{2m} n\cdot\log_m 2m\rceil\\ &\simeq \mathrm{p}(2m)\cdot(1+\log_m 2)\\ S(m) &= \mathrm{p}(m)\cdot R\\ &\simeq \mathrm{p}(2m)(1+\log_m 2)R \quad (12)\end{aligned}$$

$$S(2m) \simeq \mathrm{p}(2m)\cdot\left(1+\left(1-\frac{1}{m}\right)^m\right)R \quad (13)$$

From Eqns (12) and (13) we see that single buffering is faster than double buffering if and only if

$$\log_m 2 > \left(1-\frac{1}{m}\right)^m$$

i.e.

$$\left(1-\frac{1}{m}\right)^m \log_{10} m < \log_{10} 2 \simeq 0.30103$$

i.e. $m < 8$.

This analysis shows then, that under the various assumptions made, and for a given buffer size, if we do not have room for at least 16 input buffers then we are better off to single buffer than to double buffer. For example, if we have room for only 14 buffers, then it is faster to use single buffering and 14-way merging than to use double buffering and 7-way merging. On the other hand, if we have room for 16 or more input buffers, then double buffering is faster.

SUMMARY

We have analyzed a number of aspects of single input buffering during the merge phase of an external sort procedure. We have shown that it is a nontrivial problem to choose between complete single buffering, complete double buffering, or something in between, assuming some constraint on the amount of internal memory available for buffering. We have shown that single buffering degrades processing time for a merge pass by an average of from 25% to 37%. We have shown that single buffering is faster than double buffering if and only if there is space available for fewer than 16 buffers.

REFERENCES

1. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, pp. 324–325. Addison-Wesley, Reading, Massachusetts (1973).
2. Ref. 1, pp. 343–345.
3. E. Parzen, *Stochastic Processes*, pp. 13–15, Holden-Day, San Francisco (1965).
4. L. J. Woodrum, A model of floating buffering, *IBM Systems Journal* **9**, 118–144 (1970).

Received May 1981