Blocking sequentially processed magnetic files

S. J. Waters
LSE, London, WC2

This paper develops a mathematical algorithm that enables block sizes to be systematically calculated for sequentially processed magnetic files. A computer systems designer should find the algorithm simple to apply manually, once understood, and it is being programmed into experimental design software, which, to a large extent, automatically designs and optimises a computer system for minimum cost.

(Received January 1970)

A research project at the London School of Economics and Political Science is investigating a computer-assisted methodology of systems development; this project, known as CAM, is financed by the Science Research Council. The overall aim of the project is to demonstrate the feasibility of applying computers to the systems development function (being systems definition, design, implementation and maintenance). The present approach to systems development is manual, apart from the use of compilers and various isolated techniques, consequently systems people are involved in much routine work (e.g. areas of documentation and computer programming) but are expected, on the other hand, to solve complex technical problems (e.g. the efficient organisation of a computer system to meet complex data processing requirements). In the long term, it is anticipated that a computer-assisted methodology should result in cheaper and speedier systems development with increased flexibility and accuracy and decreased numbers of systems people.

In the short term, it is expected that immediately useful techniques will 'fall out' of the project. This paper is concerned with such a technique from the current area of research into computer-assisted systems design. This technique is part of a larger algorithm which is being developed to structure computer runs and files from defined data processing requirements and computer configurations.

A common problem in data processing systems is the choice of block sizes for magnetic files. Most of these systems currently operate in batched processing mode and many process the files sequentially (updating by the brought forward/carried forward principle). This problem is currently solved by adopting a standard block size or by trial and error calculations; both methods may produce extremely inefficient results. The following algorithm develops a near optimum solution systematically.

Central processor unit time is ignored by the algorithm as this is often immeasurable and less significant with modern computer configurations; even if this were dominant, the algorithm merely wastes spare primary storage (for a single-programming or fixed partition, multi-programming configuration).

The basic algorithm

Consider a program that sequentially processes N magnetic tape files such that:

1. Each file contains Rn records of fixed length Cn characters where $n = 1, 2, \ldots, N$. The 'information content' of file n is defined as
$$In = Rn\ Cn \qquad (1)$$
2. All magnetic tape handlers have the same characteristics:
   Transfer rate = T kc (thousand characters per second)
   Packing density = P bpi (bits per inch)
   Inter-block gap length = L c (characters)
   Inter-block start/stop time = G ms (milli-seconds).
3. All magnetic tape handlers are switched to one channel thus there is no simultaneity of information transfer between them.
4. A single buffer is to be allocated to each file, being an area of primary (e.g. core) storage large enough to contain any block of the file; thus, there is virtually no simultaneity between the magnetic tape handlers and the central processing unit.
5. The total primary storage available for these buffers is S characters (after total storage has been reduced for the operating system, the program and its work areas); S is estimated at system design and is reviewed when the program is operational.

Fig. 1 illustrates such a program that inputs a transaction file, updates a master file and outputs a result file.

The program run time is obtained by summing the following components:

1. Run set-up time (e.g. loading program and initial files) which is independent of blocking arrangements.
2. File set-up and take-down times (including rewinds) that are not simultaneous with either the run set-up or the run itself (e.g. spare handlers are not available). This time is only dependent on blocking arrangements in so far as a larger block size might reduce the number of tapes in a multi-reel file.
3. Central processing unit time which is marginally dependent on blocking arrangements in that the operating system packs and unpacks blocks.
4. Total information transfer time which is independent of blocking arrangements (in the example, this is a total of 213,000,000 characters at 40 kc giving 90 minutes, approximately).
5. Total inter-block start/stop time (in the example, this is the total number of blocks times 20 ms).

Clearly, this last component dominates the choice of blocking arrangements and the objective is to minimise the total number of blocks.

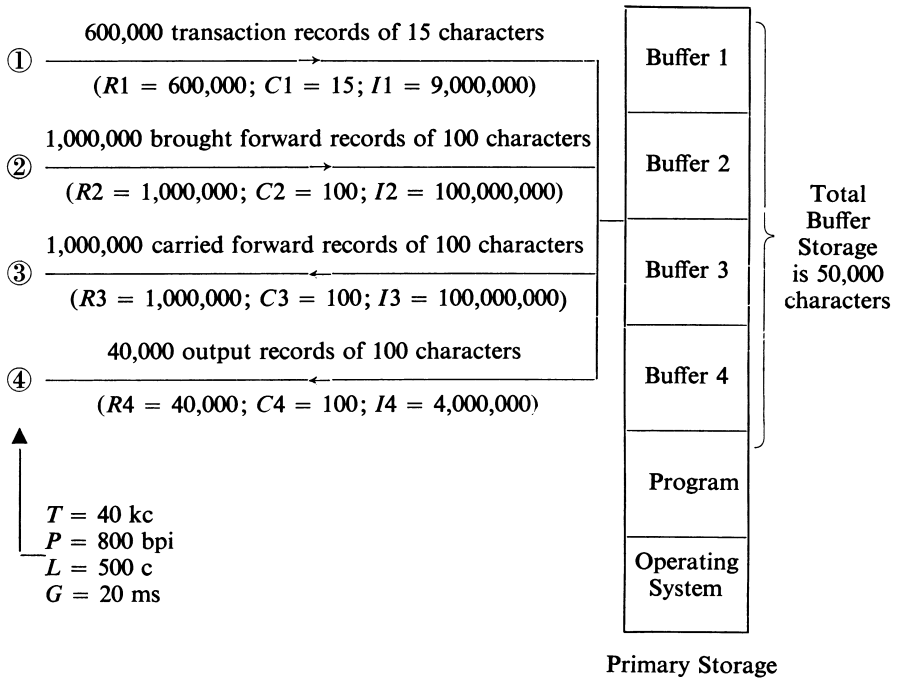

Fig. 1. Example of an update program

Let the number of records per block for each file be Bn, giving block size $Bn\,Cn$, then the total number of blocks, B to be minimised is given by

$$B = \sum_{n=1}^{n=N} \frac{Rn}{Bn} \tag{2}$$

subject to the constraint that total buffer storage must not exceed available storage,

$$S \geqslant \sum_{n=1}^{n=N} Bn\,Cn \tag{3}$$

The Appendix derives the solution as

$$Bn = S\sqrt{\frac{Rn}{Cn}} \Big/ \sum_{n=1}^{n=N} \sqrt{Rn\,Cn} \tag{4}$$

$$B = \frac{1}{S}\left(\sum_{n=1}^{n=N} \sqrt{In}\right)^2 \tag{5}$$

Fig. 2 illustrates the result of applying this basic algorithm to the example; there is a significant saving in run time and tapes. Even if only 4,000 characters were available for buffer storage, the algorithm saves 20 minutes run time and two tapes over standard blocking.

Note that the length of a file is approximately $Rn/P(Cn + L/Bn)$ inches which may be converted to a number of reels by dividing by the length of one magnetic tape.

Multi-reel files

Each tape of a multi-reel file requires loading, rewinding and unloading which often increases information transfer time significantly (e.g. by a third); thus, blocking arrangements should aim to reduce the number of tapes in such files. The basic algorithm achieves this as, from (1), (4), the buffer size is given by

$$Sn = S\sqrt{In} \Big/ \sum_{n=1}^{n=N} \sqrt{In} \tag{6}$$

which clearly weights large files (e.g. note the tape savings in the worked example).

Thus, the algorithm is applied and the number of tapes in each file is calculated. Any spare handlers are allocated one

		Single blocking	Standard blocking	Optimised blocking
File 1	Buffer size	15 charas	1,000 charas	6,000 charas
	Number of tapes*	14	1	1
Files 2, 3	Buffer size	100 charas	1,000 charas	20,000 charas
	Number of tapes*	27	7	5
File 4	Buffer size	100 charas	1,000 charas	4,000 charas
	Number of tapes*	1	1	1
Total number of blocks		2,640,000	213,000	12,500
Total start/stop time		14 hrs. 40 mins.	1 hr. 11 mins.	4 mins.

*Magnetic tape length is taken as 2,400 feet.

Fig. 2. Worked example of basic algorithm

each to the larger files so that, by 'ping-ponging', the load, rewind and unload overheads are effected simultaneously to information transfer; the first tape load and the last tape rewind and unload contribute to run time. The overheads for all remaining files, including multi-reel files on one handler, also contribute to run time.

Varying handler performances

The basic algorithm assumes that all magnetic tape handlers have the same performance which is the usual situation; however, some hardware configurations include low-speed and high-speed handlers. In this case, those files with the higher information content are usually allocated to the higher performance handlers.

Let the inter-block start/stop time for each file be Gn then the objective is not to minimise the total number of blocks but the total start/stop time; thus, the objective function (2) must be modified as

$$B = \sum_{n=1}^{n=N} \frac{Gn\,Rn}{Bn}$$

Clearly, formulae (4), (5) are modified by simply replacing Rn by $Gn\,Rn$.

Variable length records

The basic algorithm requires modification for files containing variable length records. Let such files have normal record length Cn characters and maximum record length Mn characters then the basic algorithm is applied to optimise the normal situation.

However, some operating systems impose the further constraint that block size must not be exceeded by the maximum record size

$$\therefore Mn \leqslant Bn\,Cn$$

If this fails after applying the algorithm, then allocate a block size of Mn characters and reapply the algorithm to the remaining files after reducing the total buffer storage S by these allocated buffers of Mn characters. It may be shown that this second optimisation yields smaller block sizes for the remaining files than the first optimisation; thus, it is in order to omit all files failing the constraint together and the constraint must be reapplied after the second (and possibly successive) optimisations.

Multiple buffering

The basic algorithm assumes that each file is single-buffered which is rarely the case. Generally, files are double-buffered to achieve simultaneity between file handling and processing; occasionally, files are multiple-buffered to smooth an uneven relationship between file handling and processing.

Let the number of buffers allocated to each file be Dn then constraint (3) becomes

$$S \geqslant \sum_{n=1}^{n=N} Dn\,Bn\,Cn$$

Clearly, formulae (4), (5) are modified by simply replacing Cn by $Dn\,Cn$. Alternatively, if all files are to be double-buffered, the S factor may be replaced by $S/2$ throughout.

Note that if, as a result of multiple buffering, the magnetic tape moves continuously without starting and stopping between blocks then the inter-block start/stop time should be replaced by the inter-block transfer time; in the example, 20 ms is replaced by 500c/40 kc secs = 12·5 ms.

Multiple channels

The basic algorithm assumes that all magnetic tape handlers are on one channel but they are usually allocated to two, sometimes more, channels so that the handlers on one channel may operate simultaneously with those on another; note that multiple (e.g. double) buffering is usually necessary to achieve this simultaneity.

The algorithm is applied and the handling time of each file is calculated. The files are then allocated to channels so that the total handling times of the channels are as close as possible; the run time contribution is the total handling time of those files on the dominant channel.

Direct access devices

The basic algorithm and its extensions have been developed for sequentially processed magnetic tape files; there now remains the task of interpreting the various formulae for sequentially processed direct access device files (updated by the brought forward/carried forward principle).

The characteristics of direct access devices may be compared to those of magnetic tape handlers as follows:

1. Transfer rate is typically much faster.
2. Packing density is measured as number of characters per track.
3. Inter-block gap length is typically much smaller.
4. Inter-block start/stop time is replaced by the following components:

(*a*) Seek time (e.g. disc head movement and selection). This is usually negligible if only one file is required from the device; even if several files are to be sequentially processed in parallel from the device, they may be arranged to share cylinders so that seek time is again negligible. However, seek time is sometimes significant and must be included (e.g. if the direct access device is shared by several programs run together in a multi-programming environment).

(*b*) Search time (e.g. latency or rotational delay), being half a track revolution on average. Note that if the optional check facility is taken on an output file then an extra track revolution must be included for reading after writing.

Thus, the algorithm is equally valid for sequentially processed direct access device files if the objective is to minimise run time.

Most computer systems impose the further constraint that block size must not exceed track size

$$\therefore P \geqslant Bn\,Cn$$

If this fails after applying the algorithm, then allocate a block size of P characters and reapply the algorithm to the remaining files after reducing the total buffer storage S by these allocated buffers of P characters. It may be shown that this second optimisation yields larger block sizes for the remaining files than the first optimisation; thus, it is in order to omit all files failing the constraint together and the constraint must be reapplied after the second (and possibly successive) optimisations.

Some operating systems further insist that block size must equal one of a restricted set of values (e.g. one eighth, one quarter, one half or a complete track). The algorithm may be used to establish which value is closest to the optimum.

Note that the size of a direct access device file is approximately $Rn/P(Cn + L/Bn)$ tracks.

The complete algorithm

The basic algorithm and all its extensions are now combined into a unified algorithm for a single program that sequentially processes N magnetic files; file n contains Rn records of normal length Cn characters and maximum length Mn characters ($Mn = Cn$ if the records are fixed length) therefore its normal information content is $In = Rn\,Cn$.

The procedure to minimise the run time of the program is:

1. Allocate the files to magnetic devices; often, the higher

information content files are allocated to the faster devices (note that replaceable disc devices are commonly the fastest and that a costed saving in run time may outweigh the higher cost of disc packs compared to magnetic tapes). The device characteristics *Tn*, *Pn*, *Ln*, and *Gn* are thus defined for each file.

2. Allocate the devices to channels.
3. Allocate a number of buffers, *Dn*, to each file.
4. Apply the following algorithm to optimise block sizes as

$$Sn = BnCn$$
$$= S\sqrt{Gn\,Rn.\,Dn\,Cn}/Dn \sum_{n=1}^{n=N} \sqrt{Gn\,Rn.\,Dn\,Cn}$$
$$= S\sqrt{Gn\,Dn\,In}/Dn \sum_{n=1}^{n=N} \sqrt{Gn\,Dn\,In}$$

5. Apply any direct access device constraints that

$$Pn \geqslant Sn$$

and reoptimise if necessary.

6. Apply any variable length record constraints that

$$Mn \leqslant Sn$$

and reoptimise if necessary.

7. Calculate run time.

Device, buffer and channel allocation may be varied and the algorithm reapplied to assess the affect on run time.

Finally, having optimised each program of a computer system in isolation, the interactive constraints of the entire system must be applied as follows:

1. Runs must be device compatible (e.g. a file output to a disc device must be subsequently input from a disc device).
2. Runs must be block size compatible (e.g. a file output with a particular block size must be subsequently input with that block size). If several runs handle a particular file and their optimisations result in different block sizes for that file, then the smallest block size may be chosen.

Note that if a file passes from one program to another via a sort program then it escapes the above constraints as its device and block size may vary between sorting.

Conclusion

The author would be pleased to assist any organisation wishing to apply this algorithm and to be informed of results obtained from its use.

Finally, the author wishes to acknowledge the assistance of his colleagues at the LSE, particularly Mr. F. F. Land and Dr. A. H. Land of the Statistics, Mathematics, Computing and Operational Research Department.

Appendix

To solve (2) and (3) ,let the equality apply then

$$S - \sum_{n=1}^{n=N} Bn\,Cn = 0 \qquad (1.1)$$

Using a constant Lagrange multiplier, λ, then (2) may be written as

$$B = \sum_{n=1}^{n=N} \frac{Rn}{Bn} - \lambda\left(S - \sum_{n=1}^{n=N} Bn\,Cn\right)$$
$$= -\lambda S + \sum_{n=1}^{n=N} \left(\frac{Rn}{Bn} + \lambda Bn\,Cn\right)$$

The minimum value of this function is found by equating all partial derivatives to zero

$$\therefore 0 = \frac{\partial B}{\partial Bn}$$
$$= -\frac{Rn}{Bn^2} + \lambda Cn$$
$$\therefore Bn = \sqrt{\frac{Rn}{\lambda Cn}}$$

noting that positive values must be taken. From (1.1),

$$\sqrt{\lambda} = \frac{1}{S} \sum_{n=1}^{n=N} \sqrt{Rn\,Cn}$$
$$\therefore Bn = S\sqrt{\frac{Rn}{Cn}} \Big/ \sum_{n=1}^{n=N} \sqrt{Rn\,Cn} \qquad (1.2)$$
$$B = \frac{1}{S}\left(\sum_{n=1}^{n=N} \sqrt{Rn\,Cn}\right)^2 \qquad (1.3)$$

It can be shown that the true minimum has been derived and not merely a local minimum.

Some operating systems require that the *Bn* are integral so that records do not spill from one block to the next. One method of choosing such integral values is to ignore the fractional parts of the *Bn* by truncating to the integer below; it may be shown that this increases *B* by no more than

$$\frac{1}{100S} \sum_{n=1}^{n=N} Cn\%$$

(which is often less than 10% in practice).

Footnote: The basic algorithm was also developed by Ewing S. Walker and published in the August/September, 1970 issue of Software Age (while this paper was still being refereed).