SEQUENTIAL INPUT/OUTPUT IN OS/360

A STUDY OF EFFICIENT ALTERNATIVES TO FORTRAN INPUT/OUTPUT

Abstract

It is shown that sequential input/output as implemented in OS/360 FORTRAN does not take advantage of more efficient I/O algorithms available elsewhere in OS/360. To provide the FORTRAN user with some of these facilities, a subroutine called QSIO was written using QSAM (Queued Sequential Access Method). QSIO processes a commonly used form of Fortran unformatted record and general OS/360 logical records. A comparison of the operation of the FORTRAN and QSAM algorithms in the MVT environment is given, indicating the potential advantages of the QSAM algorithms ("normal scheduling" and "chained scheduling"). The results of a timing study using FORTRAN and QSIO to read several datasets from disk and from tape are given, demonstrating improvements in real time of 20 percent for tape and 50 percent for disk, when QSAM with chained scheduling was used.
INTRODUCTION

The problem of FORTRAN input/output efficiency arose in connection with the ON-LINE SUMX program [2]. This program implements the SUMX statistical analysis program using the 2250 scope. An interactive mode is used to prepare the control statements used by a modified version of the batch SUMX program, and to examine the output after the SUMX processing ("pass") is complete. The execution of the SUMX pass is done without user intervention: the entire input dataset is read as data for histograms and other displays is collected. Using conventional unformatted FORTRAN i/o, the time required for this processing was felt to be excessive.

For example, the time to process a dataset from disk consisting of 6,000 306-word records written with a blocking factor of 6 was 1.5 minutes or more, compared with the calculated hardware access times of 24 seconds (minimum delay) to 48 seconds (maximum delay). To the user of SUMX, pass execution time is spent idle at the scope, so that efforts to minimize it are desirable, if not necessary, to justify using ON-LINE SUMX.

Assuming that the problem can be solved, or at least partially alleviated, by improving i/o efficiency, then efficient alternatives to FORTRAN i/o are worth considering for SUMX and for other applications performing moderate to large amounts of input/output. Such efforts will help the user (and possibly improve overall system performance). Control program overhead can be reduced by more efficient channel usage, and in the case of disk i/o, hardware access time can be greatly reduced.

It is interesting to note that i/o optimization is available with standard OS/360 sequential access methods, but not widely used -- due to its inavailability with FORTRAN, and lack of documentation where it is available. With the high level access method, QSAM (Queued Sequential Access Method)\(^2\), limited optimization is automatic, and full optimization ("chained scheduling") is available by specifying a DCB parameter. QSAM is used by most OS
language processors and utilities for sequential I/O, and by PL/I for
SEQUENTIAL BUFFERED I/O (default). Chained scheduling is also available
with the low level access method, BSAM (Basic Sequential Access Method)\(^3\)
but requires a generalized multiple buffer scheduling scheme to use it.

Fortran, which uses BSAM does not have such a scheme, and cannot benefit
from chained scheduling.

To provide I/O optimization for Fortran, a set of subroutines was
written to use QSAM I/O for a commonly used Fortran record format. These
routines, called GSIO collectively, were used with OR-LEN3 SUBS to reduce the
processing time of 1,5 minutes stated above to 30-40 seconds, which is
essentially the hardware access time. To obtain a better idea of what effect
the I/O algorithms had on processing time, a timing program was written to
cconduct tests of GSIO and Fortran when reading two datasets from
disk and from tape. The results of that study will be given after a discussion
of the MFT environment and the I/O algorithms' operation within it.

1. Excluded from consideration here are gains that might be available with
multi-tasking; the interest is in methods that require only slight modification
of the user's program. Even if tasking could be used, which is unlikely
since Fortran could not be made to support it without great pain, substantial
re-organization of the user's program would still be needed.

For the UNIX problem, an approach could be to rewrite the program in PL/I,
which supports tasking, using separate tasks for scope and pass
processing to achieve overlap - but this is an enormous programming
task and may not be worth the effort.

2. QSAM provides automatic blocking and de-blocking, and automatic buffer
scheduling. The user deals only with logical records.

3. BSAM deals with blocks only; blocking and buffer scheduling is done by the
user. Chained scheduling can be used without special programming if the
buffer scheduling algorithm allows more than one incomplete I/O
operation.

The discussion of Fortran I/O applies to both formatted and unformatted I/O,
which are processed by the same Fortran library routine, DLCHGS.
THE NVT ENVIRONMENT AND INPUT/OUTPUT ALGORITHMS

The NVT task management algorithm suggests ways in which I/O efficiency can be improved. With this information, the three I/O methods (PORTER, normal QMA, and QMA with chained scheduling) can be compared to indicate how they should operate with NVT.

With NVT as used at SLAC, a task (usually the job stop task) will retain control of the CPU until one of the following occurs:

1) The task terminates normally or abnormally.
2) The task must wait for an event (usually an I/O operation).
3) A higher priority task becomes ready.

If a task loses control due to preemption (case 3), it is still ready.

If it must wait for an event (case 2), it becomes ready when the event is posted, e.g., I/O complete. The task will regain CPU control when it is first in the queue of ready tasks of highest priority.

I/O scheduling is performed asynchronously with task execution: When an I/O request is made to other than NVT pseudo-devices, it is entered on the appropriate channel queue. Queue entries are processed in the order presented, without regard to the priority of the originating task. When the channel signals completion of the I/O operation by interrupting the CPU, the event associated with the operation is posted.

To improve overall task execution time, one needs to reduce the number of opportunities for task preemption, and use the channel and devices more efficiently. High I/O priority will lessen preemption due to higher priority tasks, but only if the task has something to do when it has the CPU.

If it must still wait for I/O, there will be little gain.
Input/output performance is dependent on channel usage and device access time. The normal multiple-buffer scheduling scheme ("normal scheduling" in QNAM) allows I/O operations for each buffer to be scheduled independently—after a buffer is emptied (input) or filled (output), an I/O operation is scheduled for it; the task then accesses the next buffer, which has the oldest I/O request associated with it. From the host's standpoint, the other scheduling scheme ("chained scheduling") appears similar; the difference is that I/O operations are not scheduled separately, but are "chained" together until the channel becomes available. When the channel receives the I/O request, it then processes all that have been accumulated since the last access without direct CPU intervention.

Chained scheduling, without regard to device access timing considerations, is potentially more efficient than normal scheduling since less control program intervention is required; more I/O is performed per channel access. When the input/output device is tape, the benefits of chained scheduling are limited to overhead reduction, since tape access time depends only on the amount of data to be transferred and the size of the blocks. Overhead is not insignificant, however, as gains of up to 20% were experienced using chained scheduling for tape. If the device is disk, then chained scheduling provides a dramatic improvement over normal scheduling: Delay times, which depend on the positioning of the disk access mechanism, can be very greatly reduced.

The following is a more detailed description of the I/O methods. They are discussed here with regard to input only or one I/O unit, the situation for which timing data was collected.
(1) **FORTRAN I/O** -- This is a very straightforward double-buffered scheme; no more than 2 buffers are permitted. When the device (more precisely, the DBS) is opened, one buffer is filled, and a read is issued for the second buffer after the first is filled. When data in the second buffer is needed, the task waits until it is available if necessary, and issues a read for the first buffer again. Processing continues in this fashion such that there is only one incomplete read request outstanding at any time. A task with moderate to high I/O activity will never have more than one buffer of data to process at a time, so it must wait after each buffer is processed; there is no way for more than one I/O request to be serviced while the task is inactive or processing data in other buffers.

(2) **Normal QMAM** -- With normal (opposed to chained) scheduling, there are n buffers and n channel programs. After open, n-1 buffers are filled. As a buffer is emptied, a read request is issued for it, and the next buffer is accessed if it is full, else the task waits. The advantage of this scheme is that more than one request may be outstanding -- it is possible that more than one request could be serviced while the task is inactive, so that when it gains control, several buffers of data will be ready. Even if this is not the case, more frequent channel access is possible since several I/O requests may be in the channel queue; waiting time will be less than with FORTRAN. With the QMAM scheme, the task must wait for the oldest request; after it is completed and the task regains control, the next oldest request for which the task will need to wait is already in the channel queue and possibly being serviced currently.
(3) QIAM with chained scheduling -- There are n buffers and n channel programs, but only one read request will be outstanding at any time. After open, n-1 buffers are filled. After a buffer is emptied, an attempt is made to join its channel program ("chain schedule") to the one currently in the channel queue or being executed (fetched) by the channel, thus creating a longer channel program. The joining is "successful" if the channel has not already fetched the last command of the outstanding channel program. If the joining is "unsuccessful," then a new channel program, i.e. read request, is issued. The next filled buffer may be obtained immediately if the commands referencing it in the current channel program have been processed by the channel (which may still be executing fill requests for subsequent buffers). Otherwise, the task may need to wait until the relevant commands are processed -- this is indicated by a special type of interrupt from the channel when it fetches the commands, which tells the CPU how far channel program processing has progressed. (This is a simplified explanation -- the actual implementation involves a complex interaction between the channel, the i/o supervisor, and the task.)

The significant difference between this scheme and normal scheduling is that the outstanding read request is "lengthened" by adding more read requests, rather than issuing more independent read requests. The effect is to reduce control program overhead, and in the case of disk i/o to provide a very large reduction in hardware delay time since successive blocks can be obtained during a single revolution; track switching on the same cylinder is performed without delay when blocks on successive tracks are accessed by chaining.

5. There is also an HW option called time-slicing, not used at MIT and SLAC, that will cause a task to lose CPU control when its time interval expires if one of the other three cases did not occur first.

6. A channel program is one or more channel command words (CCW). The term is used here to refer to the commands to access data for a single buffer, and for the entire collection of commands that the channel will process during a single channel access—the time between a start i/o instruction and a channel end interrupt.
A BRIEF DESCRIPTION OF QSIO

QSIO is a multiple-entry subroutine written to implement QSIO I/O for
PORTRAO. It may be used in two formats:

(1) Structured--this corresponds to PORTRAO unformatted I/O. Records
of the type created by a PORTRAO statement of the form

WRITE(U) N,(A(I),I=1,N)

are processed, where U is the unit, and N is an integer specifying the
number of words to be written from the fullword array A. Structured
QSIO calls may only be used with READ V or VB.

(2) Unstructured--this provides simple transmission of QSIO logical
records of any format (P,PS,V,VB,U). For example,

CALL QSIOIU(U,MAX,B,HADE)

will read a logical record from unit U, returning the number of bytes
of data read in integer B, and the data starting at HADE (usually an
array name). HADE is an integer specifying the size of the data area
in bytes (e.g., array length in bytes).

The operation of QSIO is similar to normal PORTRAO externally. The similarity
makes conversion to QSIO a simple matter of replacing PORTRAO I/O statements and
changing the corresponding 7D statements. A full description of QSIO is given in
//1/; its characteristics are summarized below:

1 PORTRAO statements and their QSIO equivalents are:

READ(U,END=endn,ERR=errn)     CALL QSIOII
       N,(A(I),I=1,P)
       no equivalent

WRITE(U) N,(A(I),I=1,N)       CALL QSIOII
       no equivalent

RESERVE U

RENAME U

BACKSPACE U

Parameters:

U     - 1st unit number.
N     - 1st word count, read/written as first data word of structured
A - any A array from which subsequent data words of the structured record are to be read/written.
B - 1st supplemental parameter for QUEM1L-bound of array A; used for error checking to determine if record is longer than array.
EBA - byte count for unstructured QSIO statements—number of bytes of data read or to be written.
ERA - array to be used as transmission area for unstructured QSIO statements.
BEND - 1st supplemental parameter for QUEM1L—length of transmission area in bytes, used for error checking like EBA.
INES - statement number to receive control on end-of-file.
ERRS - statement number to receive control on 1/o error.

(2) QSIO names are of the form QSnxxx (compared with FORTRAN FSnxxx).
The sequence number xxx is treated by QSIO in the same manner as FORTRAN.

(3) There are several restrictions:
(a) A unit may be opened for either input or output, but not both as in FORTRAN.
(b) No backspaces is permitted.
(c) When structured QSIO formats are used (QUEM1L, QUEM1L), no spanned records may be written; but any valid record, spanned or not, may be read.

The only restriction likely to cause problems is (c)—this can be avoided by JCL changes (to the "$S" parameter), assuming that the device can support the record length required.

(4) Several extensions of FORTRAN-type facilities may be used optionally:
(a) There is an exit for i/o errors on output, e. g., CALL QUEM1L(U, N, A, Aioerr).
(b) All calls may be written with an additional exit for non-i/o errors (e. g., no end card, record too long) --
    CALL QUEM1L(U, N, A, Alioerr, Apperr), CALL QUEM1L(U, Apperr).
(c) QSIO units may be opened explicitly with an optional DCB exit routine supplied by the user by CALL QFEM1(U, M, FOCB, Reqerr).
    M is the i/o mode (input or output), and FOCB is name of a closed
 subroutine that is called during open to inspect and modify the
DCB parameters.

QSIO was designed for simplicity--this could only be gained by restricting
processing to the single structured record format and the unstructured record
format. Generality could only be achieved by patching the FORTRAN library to
intercept calls to "OSAM units." It was felt that generality was not worth the
effort, since patching is tricky and release-dependent, and further complicated
by the presence of other patches (e.g. J. Simon's VIO22). An added benefit of
restricted processing formats is a great reduction in CPU time--about a factor of
ten over the best FORTRAN time.

7. It is possible that this restriction on spanned records can be removed
with release 17 record formats VS and V28, which use the spanning conventions
of FORTRAN.
A timing program was used to measure elapsed real time and task time required to read 2 datasets from disk and from tape. Comparisons were made using FORTRAN \( i/o \) with 2 buffers, and QIO with and without chained scheduling, using 2, 3, and 4 buffers. The FORTRAN \( i/o \) was done with the READ1 routine of G.T. Scharf \( /3/ \), which produces the best CPU time available with FORTRAN -- namely, a factor of three improvement over conventional FORTRAN \( i/o \) statements. The QIO times were obtained using QIO.

The datasets used are:

1. 100 records, each 301 words long, written unblocked. Block length is 201×4+4+(segment control)×4+(block control)×1212 bytes.
2. 6,000 records, each 291 words long, written 6 to a block. Block length is (291×4+4+(segment control))×6+(block control)×7012 bytes.

Dataset (1) represents the worst case-- short unblocked records. Dataset (2) is written optimally, with a block size equal to 2314 track capacity. Hardware access time estimates are given in table 1.

The timing program was run 7 times for disk and for tape at various priorities; the runs were:

<table>
<thead>
<tr>
<th>device</th>
<th>prty=5</th>
<th>prty=10</th>
<th>prty=12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>class B</td>
<td>class B</td>
<td>class B</td>
</tr>
<tr>
<td>disk</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>tape</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2 presents the average real and task times over all runs and for selected priorities. In some cases, very high delay times were experienced -- most likely due to waiting for the CPU-- for these cases, averages excluding them are given also.

All results show that QIO task times are much less than FORTRAN task times. This is to be expected, since QIO is a special purpose routine, while FORTRAN is very general, even with the improvement due to the user of READ1. Subsequently, only real times will be considered.

For disk \( i/o \), there was little difference between FORTRAN and normal
QSIM times. In all cases QSIM with chained scheduling reduced time by a factor of 2 or more. For the 100 record dataset, even the best times are far in excess of the hardware time (3 seconds maximum), but for the 6,000 record dataset, the best times are better than the maximum hardware time. Increasing the number of buffers did not appreciably affect the access times. From the i/o algorithms' description, it can be seen that more buffers could only improve channel access frequency with normal scheduling, while with chained scheduling, device usage could be improved also -- more blocks could be accessed without delay. For the 100 record dataset it is not clear why no improvement was experienced with more buffers -- the only explanation is that the maximum amount of channel service available was achieved with 2 buffers. For the 6,000 record dataset, little improvement was possible -- even with 2 buffers, 2 tracks (12 records) would be accessed per channel access with chained scheduling.

The tape i/o results show less marked differences among the i/o methods. For the 100 record dataset, the FORTRAN times show no regularity, but the best time is comparable with the QSIM times. When accessed with QSIM, the times showed much less variation, with a slight improvement when more buffers were used, and somewhat larger improvement when chained scheduling was used. The findings for the 6,000 record dataset are similar. Generally, with the tape times, the differences in the various QSIM options are too small to be considered greatly significant--they might be due to perturbations in system loading, but there is a consistent "improvement" which gives them some validity. Another factor is that the best QSIM times are only slightly more than the hardware times: for the 100 record dataset, 2.24 sec vs. 1.80 sec.; for the 6,000 record dataset, 68.14 sec vs. 66.20 sec. There is little room for improvement. Since times of this magnitude were obtained with only 2 buffers, additional buffers could be of small benefit. From the description of the i/o algorithms, it can be seen that additional buffers could only improve frequency of channel access, which was almost as good as could be desired for these runs.
CONCLUSIONS

It must be acknowledged that the test conditions are somewhat unrealistic -- there was no computational activity and only one i/o unit was in use -- which is the limiting case of an i/o dominated task. Yet here, it was found that QSAM was faster than FORTRAN and subject to less variation in different runs. Chained scheduling afforded only slight improvement for tape as would be expected, and great improvement for disk since device access is more efficient. These results are consistent with the i/o algorithm descriptions. The descriptions also imply that increasing the number of buffers should provide more improvement -- this was not realised (use of 2 buffers approached hardware times). It appears that the maximum channel access frequency was obtained even with two buffers. The number of buffers would be more significant if there were more computational activity and/or channel access were less frequent -- their use would allow more i/o per channel access if chaining were used.

Also, task priority made little difference, which implies that the other tasks in the system did not hold the CPU for long intervals. For the i/o study task, the CPU processing was so small that preemption did not limit CPU access. It might be possible for a CPU dominated task (e.g. a geometrical reconstruction program), using QSAM with chained scheduling and several buffers, to hold the CPU indefinitely if run at high priority!

In summary, it can be noted that QXIO does provide the FORTRAN user with an efficient, easily usable alternative to conventional FORTRAN i/o. The selection of the optimal combination of parameters -- blocking, chained scheduling, and a number of buffers -- will depend on the application and system load.
### Table 1

**Hardware Access Time Estimates**

1) **DISK -- model 231b**

- Transfer rate = $3.125 \times 10^5$ bytes/sec
- Seek time = 25 ms minimum; 15 ms maximum; 75 ms average
- Rotational delay = 0 ms minimum; 25 ms maximum; 12.5 ms average

#### a) 100 record dataset -- 100 blocks, 1212 bytes/block

<table>
<thead>
<tr>
<th></th>
<th>transfer</th>
<th>seek</th>
<th>rot. delay</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>0.39</td>
<td>0.025</td>
<td>0</td>
<td>0.425</td>
</tr>
<tr>
<td>avg</td>
<td>0.39</td>
<td>0.075</td>
<td>1.25</td>
<td>1.725</td>
</tr>
<tr>
<td>max</td>
<td>0.39</td>
<td>0.133</td>
<td>2.5</td>
<td>2.92</td>
</tr>
</tbody>
</table>

There are 6 blocks/track, so the dataset resides on 17 tracks of one cylinder. Transfer time is 0.39 sec. Total time, if one seek is required, can be computed using minimum, average, and maximum delay times (sec):

b) **6,000 record dataset -- 1,000 blocks, 7,012 bytes/block**

There is one block/track, thus 1,000 tracks or 50 cylinders are occupied by the dataset. Transfer time is 22.1 sec. Total time, with 50 seeks, is:

<table>
<thead>
<tr>
<th></th>
<th>transfer</th>
<th>seek</th>
<th>rot. delay</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>22.1</td>
<td>1.25</td>
<td>0</td>
<td>23.35</td>
</tr>
<tr>
<td>avg</td>
<td>22.1</td>
<td>1.28</td>
<td>12.5</td>
<td>36.88</td>
</tr>
<tr>
<td>max</td>
<td>22.1</td>
<td>1.4</td>
<td>25.0</td>
<td>48.5</td>
</tr>
</tbody>
</table>

II) **Tape -- model 2802, 16000psi, 9 track**

- Transfer rate = $1.20 \times 10^5$ bytes/sec
- Interblock gap = 8 ms

#### a) 100 record dataset -- 100 blocks, 1212 bytes/block

<table>
<thead>
<tr>
<th>Transfer time</th>
<th>Gap time</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8</td>
<td>1.8 sec</td>
</tr>
</tbody>
</table>

#### b) 6000 record dataset -- 1000 blocks, 7012 bytes/block

<table>
<thead>
<tr>
<th>Transfer time</th>
<th>Gap time</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>58.2</td>
<td>0.9</td>
<td>59.1 sec</td>
</tr>
</tbody>
</table>

Reference: /7/
### DISK I/O 100 RECORD UNBLOCKED DATASET

<table>
<thead>
<tr>
<th>METHOD-UPDFN:</th>
<th>TIMES(IDOTH SEC):</th>
<th>F-2</th>
<th>Q-2</th>
<th>Q-3</th>
<th>Q-4</th>
<th>QC-2</th>
<th>QC-3</th>
<th>QC-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>REN PRTY 121 12</td>
<td>5662</td>
<td>114</td>
<td>5261</td>
<td>59</td>
<td>5204</td>
<td>44</td>
<td>5359</td>
<td>56</td>
</tr>
<tr>
<td>101 10</td>
<td>5411</td>
<td>106</td>
<td>5416</td>
<td>54</td>
<td>5604</td>
<td>43</td>
<td>5363</td>
<td>49</td>
</tr>
<tr>
<td>102 10</td>
<td>5989</td>
<td>113</td>
<td>5908</td>
<td>56</td>
<td>6336</td>
<td>52</td>
<td>6106</td>
<td>53</td>
</tr>
<tr>
<td>51 5</td>
<td>5737</td>
<td>121</td>
<td>8794</td>
<td>43</td>
<td>7059</td>
<td>53</td>
<td>6015</td>
<td>51</td>
</tr>
<tr>
<td>52 9</td>
<td>7753</td>
<td>113</td>
<td>7167</td>
<td>49</td>
<td>6493</td>
<td>66</td>
<td>6275</td>
<td>44</td>
</tr>
<tr>
<td>53 9</td>
<td>5833</td>
<td>118</td>
<td>9317</td>
<td>56</td>
<td>5333</td>
<td>48</td>
<td>5323</td>
<td>48</td>
</tr>
<tr>
<td>AVERAGE ALL</td>
<td>5989</td>
<td>114</td>
<td>6209</td>
<td>52</td>
<td>5696</td>
<td>50</td>
<td>5983</td>
<td>50</td>
</tr>
<tr>
<td>ALL EXCL. #53</td>
<td>5986</td>
<td>114</td>
<td>3160</td>
<td>53</td>
<td>5994</td>
<td>51</td>
<td>6077</td>
<td>50</td>
</tr>
<tr>
<td>ONLY PRTY 12 &amp; 10</td>
<td>5647</td>
<td>111</td>
<td>5629</td>
<td>57</td>
<td>5691</td>
<td>46</td>
<td>5617</td>
<td>52</td>
</tr>
<tr>
<td>ONLY PRTY 5</td>
<td>6291</td>
<td>117</td>
<td>6645</td>
<td>49</td>
<td>6054</td>
<td>54</td>
<td>6258</td>
<td>48</td>
</tr>
<tr>
<td>PRTY 5 EXCL. #53</td>
<td>6291</td>
<td>117</td>
<td>7092</td>
<td>49</td>
<td>6298</td>
<td>55</td>
<td>6537</td>
<td>47</td>
</tr>
</tbody>
</table>

### DISK I/O 6000 RECORD BLOCKED DATASET

<table>
<thead>
<tr>
<th>METHOD-UPDFN:</th>
<th>TIMES(IDOTH SEC):</th>
<th>F-2</th>
<th>Q-2</th>
<th>Q-3</th>
<th>Q-4</th>
<th>QC-2</th>
<th>QC-3</th>
<th>QC-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>REN PRTY 121 12</td>
<td>5734</td>
<td>1386</td>
<td>5812</td>
<td>78</td>
<td>5686</td>
<td>96</td>
<td>5473</td>
<td>91</td>
</tr>
<tr>
<td>101 10</td>
<td>5520</td>
<td>1457</td>
<td>5698</td>
<td>146</td>
<td>5666</td>
<td>131</td>
<td>5577</td>
<td>128</td>
</tr>
<tr>
<td>111 10</td>
<td>9121</td>
<td>1307</td>
<td>6507</td>
<td>125</td>
<td>6325</td>
<td>108</td>
<td>6177</td>
<td>104</td>
</tr>
<tr>
<td>51 5</td>
<td>6589</td>
<td>1432</td>
<td>7063</td>
<td>103</td>
<td>5609</td>
<td>95</td>
<td>5684</td>
<td>76</td>
</tr>
<tr>
<td>52 5</td>
<td>9277</td>
<td>1404</td>
<td>8315</td>
<td>108</td>
<td>9035</td>
<td>94</td>
<td>6671</td>
<td>79</td>
</tr>
<tr>
<td>53 5</td>
<td>5758</td>
<td>1424</td>
<td>7311</td>
<td>84</td>
<td>1347</td>
<td>103</td>
<td>6221</td>
<td>83</td>
</tr>
<tr>
<td>54 5</td>
<td>6913</td>
<td>1414</td>
<td>6641</td>
<td>111</td>
<td>7465</td>
<td>115</td>
<td>6611</td>
<td>96</td>
</tr>
<tr>
<td>AVERAGE ALL</td>
<td>6913</td>
<td>1414</td>
<td>6552</td>
<td>115</td>
<td>6501</td>
<td>117</td>
<td>6110</td>
<td>95</td>
</tr>
<tr>
<td>ALL EXCL. #53</td>
<td>6913</td>
<td>1414</td>
<td>6552</td>
<td>115</td>
<td>6501</td>
<td>117</td>
<td>6110</td>
<td>95</td>
</tr>
<tr>
<td>ONLY PRTY 12 &amp; 10</td>
<td>6502</td>
<td>1406</td>
<td>6259</td>
<td>110</td>
<td>6088</td>
<td>126</td>
<td>5870</td>
<td>122</td>
</tr>
<tr>
<td>ONLY PRTY 5</td>
<td>7324</td>
<td>1420</td>
<td>6962</td>
<td>112</td>
<td>6496</td>
<td>106</td>
<td>7161</td>
<td>77</td>
</tr>
<tr>
<td>PRTY 5 EXCL. #53</td>
<td>7324</td>
<td>1420</td>
<td>6846</td>
<td>121</td>
<td>6915</td>
<td>107</td>
<td>6341</td>
<td>75</td>
</tr>
</tbody>
</table>
### TAPE 1/0 100 RECORD UNBLOCKED DATASET

<table>
<thead>
<tr>
<th>METHOD-BUENGI</th>
<th>TIME(100TH SEC):</th>
<th>F-2 REAL TASK</th>
<th>Q-2 REAL TASK</th>
<th>Q-3 REAL TASK</th>
<th>Q-4 REAL TASK</th>
<th>GC-2 REAL TASK</th>
<th>GC-3 REAL TASK</th>
<th>GC-4 REAL TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFN</td>
<td>PRTY</td>
<td>101</td>
<td>10</td>
<td>1108</td>
<td>31</td>
<td>275</td>
<td>7</td>
<td>296</td>
</tr>
<tr>
<td>102</td>
<td>10</td>
<td>255</td>
<td>29</td>
<td>222</td>
<td>5</td>
<td>243</td>
<td>4</td>
<td>221</td>
</tr>
<tr>
<td>51</td>
<td>5</td>
<td>2295</td>
<td>26</td>
<td>1103</td>
<td>3</td>
<td>254</td>
<td>3</td>
<td>214</td>
</tr>
<tr>
<td>52</td>
<td>5</td>
<td>242</td>
<td>31</td>
<td>222</td>
<td>7</td>
<td>230</td>
<td>6</td>
<td>292</td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td>1131</td>
<td>31</td>
<td>355</td>
<td>10</td>
<td>330</td>
<td>6</td>
<td>292</td>
</tr>
<tr>
<td>54</td>
<td>5</td>
<td>255</td>
<td>28</td>
<td>222</td>
<td>4</td>
<td>223</td>
<td>4</td>
<td>222</td>
</tr>
<tr>
<td>55</td>
<td>5</td>
<td>425</td>
<td>1</td>
<td>223</td>
<td>6</td>
<td>224</td>
<td>1</td>
<td>215</td>
</tr>
</tbody>
</table>

**AVERAGE ALL**

1322 | 29 | 403 | 6 | 310 | 5 | 292 | 4 | 222 | 3 | 222 | 3 | 222 | 3 | 222 | 5 |

**PRTY 10 ONLY**

1879 | 30 | 248 | 6 | 240 | 2 | 237 | 4 | 237 | 3 | 212 | 1 | 215 | 5 | 215 | 5 |

**PRTY 5 ONLY**

1068 | 29 | 466 | 6 | 336 | 6 | 314 | 4 | 226 | 3 | 226 | 4 | 226 | 5 |

### TAPE 1/0 6000 RECORD BLOCKED DATASET

<table>
<thead>
<tr>
<th>METHOD-BUENGI</th>
<th>TIME(100TH SEC):</th>
<th>F-2 REAL TASK</th>
<th>Q-2 REAL TASK</th>
<th>Q-3 REAL TASK</th>
<th>Q-4 REAL TASK</th>
<th>GC-2 REAL TASK</th>
<th>GC-3 REAL TASK</th>
<th>GC-4 REAL TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFN</td>
<td>PRTY</td>
<td>101</td>
<td>10</td>
<td>10212</td>
<td>1396</td>
<td>9296</td>
<td>113</td>
<td>7289</td>
</tr>
<tr>
<td>102</td>
<td>10</td>
<td>8312</td>
<td>1424</td>
<td>6016</td>
<td>128</td>
<td>6818</td>
<td>135</td>
<td>6857</td>
</tr>
<tr>
<td>51</td>
<td>5</td>
<td>13751</td>
<td>1421</td>
<td>1881</td>
<td>134</td>
<td>32507</td>
<td>133</td>
<td>7798</td>
</tr>
<tr>
<td>52</td>
<td>5</td>
<td>7753</td>
<td>1424</td>
<td>6507</td>
<td>129</td>
<td>6886</td>
<td>126</td>
<td>6822</td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td>8895</td>
<td>1394</td>
<td>7332</td>
<td>129</td>
<td>7170</td>
<td>134</td>
<td>7044</td>
</tr>
<tr>
<td>54</td>
<td>5</td>
<td>8672</td>
<td>1376</td>
<td>8094</td>
<td>99</td>
<td>6909</td>
<td>76</td>
<td>6794</td>
</tr>
<tr>
<td>55</td>
<td>5</td>
<td>11875</td>
<td>222</td>
<td>11006</td>
<td>236</td>
<td>12258</td>
<td>247</td>
<td>11788</td>
</tr>
</tbody>
</table>

**AVERAGE ALL**

9325 | 1412 | 9127 | 136 | 11173 | 140 | 7387 | 140 | 8033 | 121 | 7848 | 124 | 7767 | 117 |

**ALL EXCL. #5 & 55**

9408 | 1410 | 7429 | 129 | 6996 | 123 | 6931 | 120 | 6905 | 113 | 6818 | 110 | 6814 | 106 |

**PRTY 10 ONLY**

9262 | 1410 | 8051 | 124 | 7093 | 131 | 6972 | 124 | 6805 | 116 | 6808 | 123 | 6803 | 111 |

**PRTY 5 ONLY**

9337 | 1413 | 8958 | 141 | 13783 | 144 | 8113 | 142 | 8524 | 124 | 8264 | 125 | 8152 | 120 |

**P=5 EXCL. #5 & 55**

7844 | 1411 | 7015 | 117 | 6992 | 117 | 6904 | 110 | 6972 | 112 | 6325 | 110 | 6821 | 102 |

**Key:** Method codes: **F**-FIFO, **Q**-QDIS, **GC**-GCQ, **QAQ**-QAQ with normal scheduling, **QAQAC**-QAQ with chained scheduling.

RFN: Arbitrary reference number to refer to results of each run.
References


3. G.T. Scharf, Improving FORTRAN I/O Timing. User note #26, SLAC Facility, Stanford Computation Center. READI is documented as SLAC Library Program #11,43.


