CGTM No. 81
John Ahern
Nov. 1969

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 improvementsin 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 6,000 302-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[1]. 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.[4] These routines, called QSIO collectively, were used with ON-LINE SUMX 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 conduct tests of QSIO and FORTRAN when reading two datasets from disk and from tape. The results of that study will be given after a discussion of the MVT 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 SUMX 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 deblocking, 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.

4. The discussion of FORTRAN i/o applies to both formatted and unformatted i/o, which are processed by the same FORTRAN library routine, IHCFIOSH.

THE MVT ENVIRONMENT AND INPUT/OUTPUT ALGORITHMS

The MVT task management algorithm suggests ways in which i/o efficiency can be improved. With this information, the three i/o methods (FORTRAN, normal QSAM, and QSAM with chained scheduling) can be compared to indicate how they should operate with MVT.

With MVT as used at SLAC, a task (usually the job step task) will retain control of the CPU until one of the following occurs:[5]

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 lost 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 HASP 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 job step 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 QSAM) 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 task's standpoint, the other scheduling scheme ("chained scheduling") appears similiar; 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 on 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 dataset (more precisely, the DCB) 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 QSAM -- With normal (opposed to chained) scheduling, there are n buffers and n channel programs[6], 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 QSAM 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) QSAM 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 MVT option called time-slicing, not used at 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 QSAM i/o for FORTRAN. It may be used in two formats:

(1) Structured--this corresponds to FORTRAN unformatted i/o. Records of the type created by a FORTRAN 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 RECFM V or VB.

(2) Unstructured--this provides simple transmission of OS/360 logical records of any format (F,FB,V,VB,U). For example,

```
CALL QRDLR1(U,BMAX,B,BADDR)
```

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

The operation of QSIO is similiar to normal FORTRAN externally. The similiarity makes conversion to QSIO a simple matter of replacing FORTRAN i/o statements and changing the corresponding DD statements. A full description of QSIO is given in /1/; its characteristics are summarized below:

(1) FORTRAN statements and their QSIO equivalents are:

READ(U,END=endsn,ERR=errsn) N,(A(I),I=1,N)	CALL QREAD1 (U,NMAX,N,A,&endsn,&errsn)
WRITE(U) N,(A(I),I=1,N)	CALL QWRIT1(U,N,A)
no equivalent	CALL QRDLR1 (U,BMAX,B,BA,&endsn,&errsn)
no equivalent	CALL QWRLR1 (U,B,BA)
ENDFILE U	CALL QCLOS1(U)
REWIND U	CALL QREW1(U)
BACKSPACE U	not permitted

Parameters:

U - I*4 unit number,
N - I*4 word count, read/written as first data word of structured

	logical record.
A	- any*4 array from which subsequent data words of the structured record are to be read/written.
I	- I*4 implied do loop index in FORTRAN i/o statements.
NMAX	- I*4 supplemental parameter for QREAD1--bound of array A; used for error checking to determine if record is longer than array.
B	- I*4 byte count for unstructured QSIO statements--number of bytes of data read or to be written.
BA	- any type array to be used as transmission area for unstructured QSIO statements.
BMAX	- I*4 supplemental parameter for QRDLR1--length of tranmission area in bytes, used for error checking like NMAX.
endsn	- statement number to receive control on end-of-file.
errsn	- statement number to receive control on i/o error.

(2) QSIO ddnames are of the form QTuuFnnn (compared with FORTRAN FTuuFnnn). The sequence number nnn 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 backspace is permitted.

(c) When structured QSIO formats are used (QREAD1,QWRIT1), 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 LRECL parameter), assuming that the device can support the record length required.[7]

(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 QWRIT1 (U,N,A,&ioerr).

(b) All calls may be written with an additional exit for non-i/o errors (e. g., no dd card, record too long)--
CALL QWRIT1(U,N,A,&ioerr,&pgmerr), CALL QREW1(U,&pgmerr).

(c) QSIO units may be opened explicitly with an optional DCB exit routine supplied by the user by CALL QOPEN1(U,M,DCBE,&pgmerr). M is the i/o mode (input or output), and DCBE 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 "QSAM 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. Ehrman's FIO999). 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 VBS, which use the spanning conventions of FORTRAN.

THE TIMING STUDY

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 QSAM 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 QSAM times were obtained using QSIO.

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+4(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:

device	prty=5 class B	prty=10 class N	prty=12 class O
disk	4	2	1
tape	5	2	0

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 QSIO task times are much less than FORTRAN task times. This is to be expected, since QSIO 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

QSAM times. In all cases QSAM 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 QSAM times. When accessed with QSAM, 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 QSAM 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 QSAM 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 realized (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 QSIO 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 I
HARDWARE ACCESS TIME ESTIMATES

I) DISK -- model 2314

transfer rate = 3.19×10^5 bytes/sec
seek time = 25ms minimum; 135ms maximum; 75ms average
rotational delay=0ms minimum; 25ms maximum; 12.5ms average

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

There are 6 blocks/track, so the dataset resides on 17 tracks of one cylinder. Transfer time is .39 sec. Total time, if one seek is required, can be computed using minimum, average, and maximum delay times (sec):

	transfer	seek	rot. delay	total
minimum:	.39	.025	0	.415
average:	.39	.075	1.25	1.715
maximum:	.39	.135	2.5	2.92

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:

	transfer	seek	rot. delay	total
minimum:	22.1	1.25	0	23.35
average:	22.1	1.28	12.5	35.88
maximum:	22.1	1.4	25.0	48.5

II) TAPE -- model 2402, 1600bpi, 9 track

transfer rate = 1.20×10^5 bytes/sec
interblock gap = 8 ms

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

Transfer time	1
Gap time	.8
Total	1.8 sec.

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

Transfer time	58.2
Gap time	8.0
Total	66.2 sec.

Reference: /7/

TABLE II.

DISK I/O 100 RECORD UNBLOCKED DATASET

METHOD-BUFNO: TIMES(100TH SEC): RFN	PRTY	F-2 REAL	TASK	Q-2 REAL	TASK	Q-3 REAL	TASK	Q-4 REAL	TASK	QC-2 REAL	TASK	QC-3 REAL	TASK	QC-4 REAL	TASK
121	12	5662	114	5561	59	5364	44	5359	56	1803	34	1892	38	2065	34
101	10	5411	106	5418	54	5404	43	5383	49	1797	31	1811	29	18C1	31
102	10	5989	113	5908	58	6306	53	6109	53	2034	31	1919	28	1929	34
51	5	5737	121	8793	43	7069	53	8C15	51	1857	34	2402	43	2213	34
52	5	7753	113	7167	49	6493	66	6275	44	2347	41	205C	24	2005	43
53	5			5305	49	5321	49	5420	51	22557	43	2957	33	3049	31
54	5	5383	118	5317	56	5333	48	5323	48	1879	31	1824	26	1799	39
AVERAGE ALL		5989	114	6209	52	5898	5C	5983	50	4896	35	2122	31	2123	35
ALL EXCL. #53		5989	114	[illegible]360	53	5994	51	6C77	50	1952	33	1983	31	1968	35
ONLY PRTY 12 & 10		5687	111	5629	57	5691	46	5617	52	1878	32	1874	31	1931	33
ONLY PRTY 5		6291	117	6645	49	6054	54	6258	48	7160	37	2308	31	2266	36
PRTY 5 EXCL. #53		6291	117	7092	49	6298	55	6537	47	2027	35	2092	31	2005	38

DISK I/O 6000 RECORD BLOCKED DATASET

METHOD-BUFNO: TIMES(100TH SEC): RFN	PRTY	F-2 REAL	TASK	Q-2 REAL	TASK	Q-3 REAL	TASK	Q-4 REAL	TASK	QC-2 REAL	TASK	QC-3 REAL	TASK	QC-4 REAL	TASK
121	12	5734	1386	5812	78	5686	96	5473	91	3132	83	3164	69	3133	86
101	10	5620	1457	5688	146	5686	131	5577	128	3090	124	3089	129	3100	133
102	10	8154	1381	7277	106	6892	153	6587	149	3232	101	3225	108	3157	114
51	5	6589	1432	6763	103	5669	99	6868	76	4682	96	4388	89	3453	76
52	5	9627	1404	8315	108	9635	94	6671	79	4373	91	3376	83	3715	83
53	5			7311	84	13247	1C3	9621	83	5543	71	5283	56	3987	103
54	5	5758	1424	5461	153	5443	129	5485	71	3070	118	4125	113	3359	114
AVERAGE ALL		6913	1414	6661	111	7465	115	6611	96	3874	97	3807	93	3414	101
ALL EXCL. #53		6913	1414	6552	115	6501	117	611C	99	3596	102	3561	99	3319	101
ONLY PRTY 12 & 10		6502	1408	6259	110	6088	126	5879	122	3151	102	3159	102	3130	111
ONLY PRTY 5		7324	1420	6962	112	8498	1C6	7161	77	4417	94	4293	86	3628	94
PRTY 5 EXCL. #53		7324	1420	6846	121	6915	1C7	6341	75	4041	101	3963	96	3509	91

TABLE II CONTINUED

TAPE I/O 100 RECORD UNBLOCKED DATASET

METHOD-BUFNO: TIMES(100TH SEC): RFN	PRTY	F-2 REAL	TASK	Q-2 REAL	TASK	Q-3 REAL	TASK	Q-4 REAL	TASK	QC-2 REAL	TASK	QC-3 REAL	TASK	QC-4 REAL	TASK
101	10	1108	31	275	7	258	3	253	8	214	3	213	1	223	1
102	10	2551	29	222	5	223	4	221	1	213	4	211	1	216	9
51	5	2295	28	1100	9	684	8	534	8	254	8	261	4	254	6
52	5	542	31	228	7	233	6	251	3	217	4	213	4	240	3
53	5	1181	31	355	10	330	8	292	6	233	1	233	6	216	1
54	5	255	28	222	4	223	4	222	4	213	3	211	1	213	4
55	5			425	1	223	6	224	1	215	3	213	6	211	6
AVERAGE ALL		1322	29	403	6	310	5	292	4	222	3	222	3	224	5
PRTY 10 ONLY		1829	30	248	6	240	3	237	4	213	3	212	1	219	5
PRTY 5 ONLY		1068	29	466	6	338	6	314	4	226	3	226	4	226	5

TAPE I/O 6000 RECORD BLOCKED DATASET

METHOD-BUFNO: TIMES(100TH SEC): RFN	PRTY	F-2 REAL	TASK	Q-2 REAL	TASK	Q-3 REAL	TASK	Q-4 REAL	TASK	QC-2 REAL	TASK	QC-3 REAL	TASK	QC-4 REAL	TASK
101	10	10212	1396	9286	113	7288	134	7127	136	6804	114	6807	123	6801	104
102	10	8312	1424	6816	136	6838	128	6819	133	6807	118	6809	124	6806	119
51	5	13791	1421	14871	134	35207	133	7796	134	9919	119	9031	143	8492	108
52	5	7753	1424	6909	129	6886	126	6862	139	7315	128	6875	109	6867	121
53	5	8895	1434	7332	129	7170	134	7044	104	6806	122	6806	119	6804	116
54	5	6872	1376	6804	93	6800	93	6806	88	6796	86	6796	76	6794	71
55	5			11875	222	11806	236	12058	247	11788	166	11814	179	11806	186
AVERAGE ALL		9305	1412	9127	136	11713	140	7787	140	8033	121	7848	124	7767	117
ALL EXCL. #51 & 55		8408	1410	7429	120	6996	123	6931	120	6905	113	6818	110	6814	106
PRTY 10 ONLY		9262	1410	8051	124	7063	131	6973	134	6805	116	6808	123	6803	111
PRTY 5 ONLY		9327	1413	9558	141	13573	144	8113	142	8524	124	8264	125	8152	120
P=5 EXCL #51 & 55		7840	1411	7015	117	6952	117	6904	110	6972	112	6825	101	6821	102

Key: Method codes: F=FORTRAN, Q=QSAM with normal scheduling, QC=QSAM with chained scheduling.
RFN: Arbitrary reference number to refer to results of each run.

References

1. J. Ahern, QSIO -- Queued Sequential Access Method (QSAM) Input/Output for FORTRAN. (Program description). In preparation.

2. W.C. McGee, H.R. Penafiel, and S.K. Howry. Analysis and Display of Physics Data. IBM Systems Journal, v. 7, no 3-4 (1968). pp. 342-354.

3. G.T. Scharf, Improving FORTRAN I/O Timing. User note #26, SLAC Facility, Stanford Computation Center. READ1 is documented as SLAC Library Program #I1.43.

4. IBM/360 Operating System FORTRAN IV (H) Compiler Program Logic Manual. Form Y28-6642-2.

5. IBM System/360 Operating System MVT Control Program Logic Summary. Form Y28-6658-1.

6. IBM System/360 Operating System Sequential Access Methods Program Logic Manual. Form Y28-6604-1

7. User's Manual (SLAC Facility Edition). February 1968.