BEHAVIOR OF $RANDOM ACROSS MUMPS IMPLEMENTATIONS

T. Bain Henderson

Ralph H. Johnson VA Medical Center
Charleston, SC

Abstract

The MUMPS Standard requires that the intrinsic function $RANDOM return a random or pseudo-random integer which is uniformly distributed in the closed interval [0,intexpr-1]. Several commercial implementations of the MUMPS $RANDOM function are reviewed. The results are contrasted to random number functions in two other programming languages and are considered in terms of the general concepts of software reliability, the inherent complexity of software systems, and the difficulty associated with evaluating such systems using the MUMPS $RANDOM function as a model.

1. Introduction

Humans often apply the term random to phenomena which are difficult to predict, i.e., items which are aimless, haphazard or lacking in apparent purpose. This intuitive view is in stark contrast to the use of the term random when one generates random numbers with a computer. In this latter context, a series of numbers can be random but absolutely predictable. Random numbers generated with a computer are generally termed pseudo-random or quasi-random and these numbers may be termed qualitatively random by virtue of the way they are generated. It appears that random is not so much in the **what** as in the **how**, at least where computers are concerned.

2. Computer Generation Of Random Numbers

A number of programming languages, i.e., BASIC, C, and MUMPS (to mention a few) provide tools which make the generation of random numbers relatively easy. A common method of generating random numbers with a computer is by use of variations of the linear congruential method:

$$S_{n+1} = (mS_n + i) \bmod u \qquad (1)$$

where S is the starting value;
m is the multiplier;
i is the increment;
u is the modulus.

Equation 1 is straight forward but the behavior of a generator which uses this approach is highly dependent on the values of m, i, and u; and there is something of an art associated with the selection of appropriate numbers (See Knuth[1] or Jain[2] for extended discussions of this type of generator). A random number generator which relies on Equation 1 will, theoretically, generate a uniform distribution of numbers.

3. Testing Random Number Generators

A chi-square test is commonly used to test the output of random number generators for non-randomness. Indeed, chi-square is one of the tests used in the National Bureau of Standards MUMPS Validation Suite to test $RANDOM for adherence to the MUMPS standard[3].

$$X^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \qquad (2)$$

where i = the increment, i.e., 1, 2, 3,....,k;
k = the number of categories;
O = an observed frequency;
E = an expected frequency.

It can be shown, for relatively large samples, that chi-square will follow the chi-square distribution with k-1 degrees of freedom. Keep in mind that chi-square does not tell us that our series of numbers is random; rather it detects departures from random behavior. If, for example, we executed the MUMPS line,

```
F I=0:1:29 W $R(10)," "
```

we might obtain the following sample of random numbers, 7 7 5 6 6 3 5 1 9 9 5 1 7 3 9 3 7 4 4 3 7 9 5 5 5 9 2 8 2 6 which are summarized in Figure 1 on the following page.

If we now compute a chi-square using our theoretical set of numbers we will obtain the value 11.33. We can see in Table 1 that our value (V) of 11.33 falls between the p = 50% level and the p = 75% level. According to Knuth (Page 44):

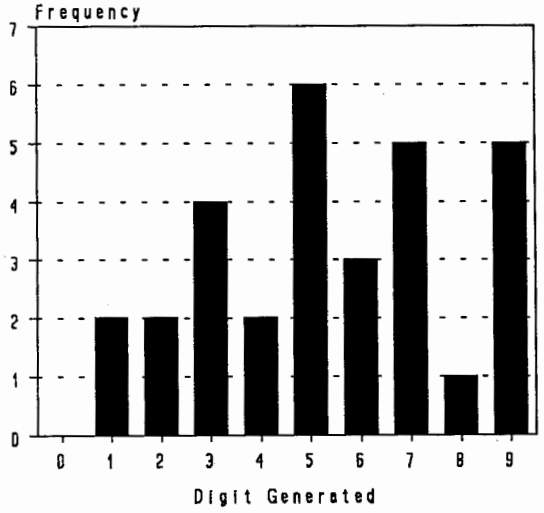

Figure 1 Trial $RANDOM Output

If V is less than the 1% entry or greater than the 99% entry, we reject the numbers as not sufficiently random. If V lies between the 1% and 5% entries or between the 95% and 99% entries, the numbers are "suspect"; if (by interpolation in the table) V lies between the 5% and 10% entries, or the 90% and 95% entries, the numbers might be "almost suspect."

p=1%	2.09
p=5%	3.32
p=25%	5.90
p=50%	8.34
p=75%	11.39
p=95%	16.92
p=99%	21.67

Table 1 Chi-Square Table For 9 Degrees of Freedom

Following Knuth's logic, we can interpret our chi-square value of 11.33 as an indication that the numbers in our test series do not represent a substantial departure from random behavior. Note that we have not proven that our series is random, rather we have shifted the burden of proof away from the notion that our series is not random.

4. Random Number Functions In MUMPS Implementations

The MUMPS Standard requires that the intrinsic function $RANDOM return a random or pseudo-random integer which is uniformly distributed in the closed interval [0,intexpr-1]. We have reviewed the MUMPS $RANDOM function of the five MUMPS implementations which are listed in Table 2.

Vendor	Product
Datatree	DTM-PC 4.3K
Digital	VAX DSM V6.0A
MGlobal	MGM/PC 5.08
MGlobal	CCSM 5.03
Micronetics	MSM-PC 3.0.8

Table 2 MUMPS Implementations

A vendor specific batch of numbers was generated with the following MUMPS line:

```
F I=0:1:9999 W $R(10)
```

The output for each MUMPS implementation is summarized in Figures 2 through 6. A review of these summaries indicates no dramatic departures from what we would expect, i.e., a uniform distribution of numbers between 0 and 9. I have computed a chi-square value for each of our vendor specific numeric series and the results are listed in Table 3. If we use Knuth's guidelines, we will accept all of the outputs, except CCSM, as not deviating in any significant fashion from random behavior, at least as far as we are able to tell using a chi-square value as an indicator.

Further Analysis

Tables 4 through 8 (included as Attachment A) summarize the output of the following MUMPS line for each MUMPS implementation:

```
F N=0:1:21 W $R(10),!
```

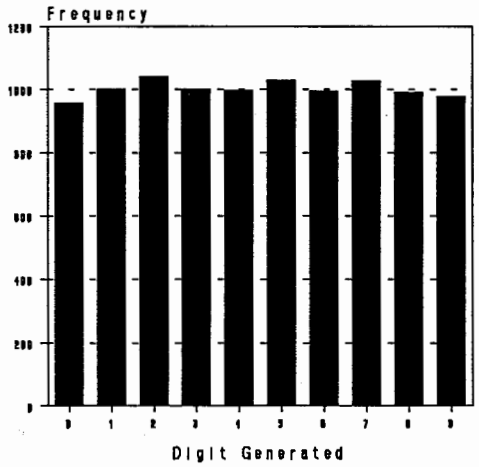

Figure 2 DTM $RANDOM Output

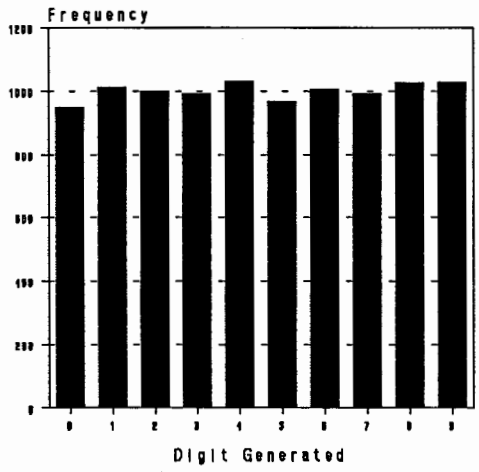

Figure 3 DSM $RANDOM Output

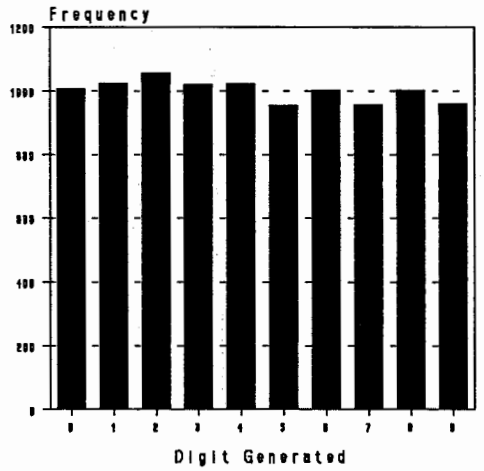

Figure 4 MGM/PC $RANDOM Output

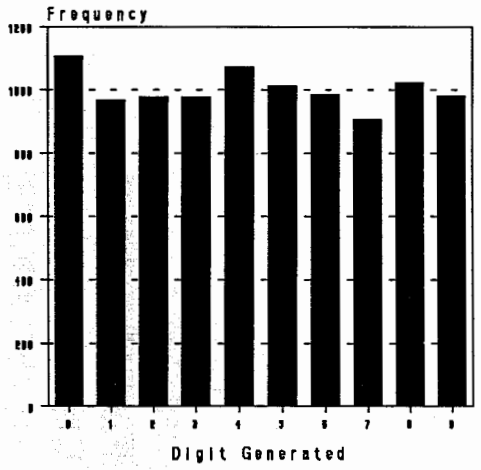

Figure 5 MGM/CCSM $RANDOM Output

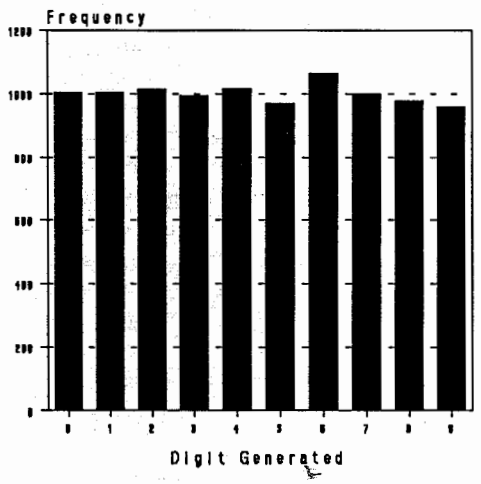

Figure 6 MSM $RANDOM Output

Product	Value
DTM-PC 4.3K	5.65
VAX DSM 6.0A	6.32
MGM/PC 5.08	10.55
CCSM 5.03	29.06
MSM-PC 3.0.8	7.55

Table 3 Chi-Square Values

An identical MUMPS program containing this line of code was created for each of the MUMPS implementations listed in Table 2. The program was then executed ten consecutive times from the operating system level for each of the MUMPS systems used. The following sequencing was used: 1. The MUMPS program was started using the appropriate operating system command line reference; 2. the MUMPS program containing the above line of code was executed and the output from the program was saved; 3. MUMPS was halted; 4. the entire sequence was repeated until ten cycles were completed.

The results of this activity point out inconsistencies in the different implementations. The Datatree DTM-PC test (See Table 4 in Attachment A) produces an "apparent" random series on each of the ten runs. The Digital Vax DSM test (See Table 5 in Attachment A) produces a series of numbers which begins with zero on each execution, though the numbers after zero appear to be random. The MGlobal MGM/PC test (See Table 6 in Attachment A) also produces an apparently random series of numbers. The MGlobal CCSM test (See Table 7 in Attachment A) produces several series that are the same or almost the same on each run. The Micronetics MSM-PC test (See Table 8 in Attachment A) generates exactly the same series of numbers for each execution.

Same Seed Result I first observed what I have come to call the same seed result, i.e., the MSM type output, with an early version of MUMPS for UNIX which was developed by Dave Bridger and implemented on an Onyx C8000 computer. Subsequently, I experienced the same result with a UNIX-MUMPS system implemented by PFCS Corporation (I believe that the latter MUMPS was derived from Bridger's MUMPS) and running on a BBN C/70. I am under the impression that the current PFCS implementation of UNIX-MUMPS no longer produces the same seed result[4]. An early version of MSM-UNIX also produced the same seed result[5].

In some situations the same seed result might be desirable, i.e., simulation activities, but in other situations, i.e., drawing random samples of telephone numbers or patients, same seed behavior is not a desirable characteristic. Recently, I discussed the same seed behavior with an individual at Micronetics Design Corporations[6] and was provided with the following work-around:

```
random  n a,i,x
        ;First get time of day from $H
        s a=$p($h,",",2)
        ;Next seed $R with FOR loop
        ;f i=1:1:a#100 s x=$r(1000)
        Here add your own code...
```

It is apparent from our small review of the $RANDOM function that we get somewhat different results depending on what implementation we use and how we use it.

5. Random Number Functions In Other Languages

A number of computer languages, other than MUMPS, implement random number functions which are relatively easy to use.

BASIC Language

Various implementations of the BASIC computer programming language have a function called RND which is similar to the MUMPS $RANDOM function. The BASIC code which follows will generate a series of 22 pseudo-random numbers in the range 0 through 9:

```
10 FOR R=1 TO 22
20 PRINT INT(RND*10);
30 NEXT
```

Output from
Program: 7 5 5 2 3 7 0 7 8 7 0 4 8 7 3 9 8 0 9 3 5 7

In fact, each time this code is run it will generate the same set of random numbers. If you want a different series you must seed the generator using BASIC's RANDOMIZE statement. You can initialize the RANDOMIZE statement with the system clock or with a number. The BASIC code which follows will reseed the RND function each time it is run and will theoretically produce (within system limits) a different series each time the program is run.

```
10 RANDOMIZE TIMER
20 FOR R=1 TO 5
30 PRINT INT(RND*10);
40 NEXT
```

I have modified line 10 of the first BASIC program above, i.e,

```
10 FOR R=0 TO 9999
```

and run the program to produce the series of 10,000 random numbers which is displayed in Figure 7. The value of chi-

square computed from these numbers is 6.59 which (according to Knuth's yard stick) indicates that our numbers do not represent a substantial variation from random behavior.

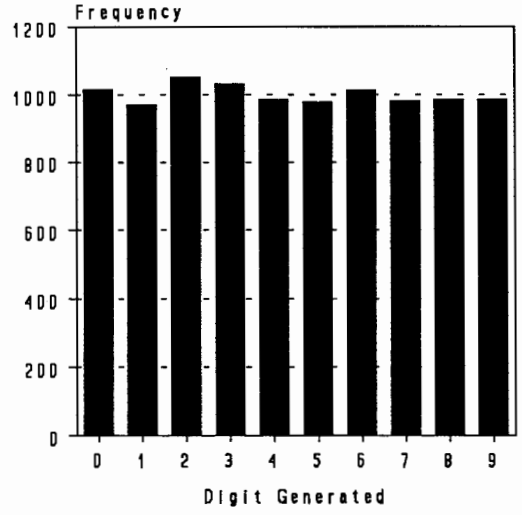

Figure 7 BASIC Random Output

C Language

C has capabilities similar to BASIC for dealing with random numbers. The ANSI C standard library has two related functions for dealing with the generation of random numbers, srand and rand. Srand is the C complement of BASIC's RANDOMIZE statement and rand is similar to BASIC's RND function. Rand returns a pseudo-random integer in the range 0 to the constant RAND_MAX , which is at least 32767[7]. The following C program, when compiled and executed, will generate a series of 22 pseudo-random numbers:

```
main()
{
int n;
for (n=0; n < 22; ++n)
        printf("%d ", rand());
}
```

Output From
Program: 0 4310 24759 15029 17457 7174 1541 22245
22259 30628 1 2566 17020 27229 1132 1751
17357 21992 252 12563 2190 13680 22504

This C program is approximately equivalent to the MUMPS line:

```
F N=0:1:22 W $R(32768)," "
```

This C program will generate the same series of random numbers each time it is run.

The following C program[8] makes use of the C srand function for seeding rand and is driven from the system clock. This program will generate a different random number series (within limits) each time it is run and is similar to our second BASIC program above:

```
main()
{
long int time();
int n;
srand ((int) time ((long int *) 0));
for (n = 0; n < 22; ++r)
        printf("%d ", rand());
}
```

I have modified our first C program, compiled and executed it, to generate the 10,000 random numbers which are displayed in Figure 8.

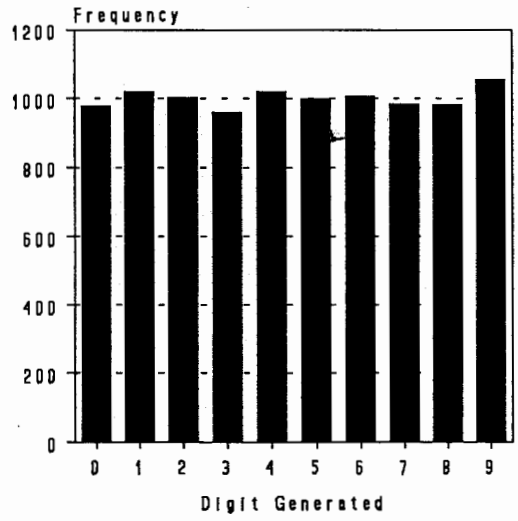

Figure 8 C Random Output

Note that we have used the tenths digit of each random number generated by our C program in our summary and

analysis. The value of chi-square computed from these numbers is 6.45 and referring, once again, to Table 1 we see that these numbers would not appear to represent a substantial departure from random behavior.

6. Summary

A comprehensive evaluation of the characteristics of the random number generators that we have examined would require a much more rigorous approach to the problem than we have taken in this paper. Ideally, we would run several chi-squares for each implementation. In addition, there are a number of other statistical tests (which we do not have the space to discuss here) that should be used to evaluate various characteristics of random numbers. However, even if all of our random number generators could pass the world's supply of statistical tests, we would still have to deal with the inconsistencies that we have found between the various MUMPS random number generators.

The point here has been to demonstrate the inconsistencies between the various generators. Some of the generators exhibit the same seed characteristic, i.e., MSM, while others exhibit behavior which is more difficult to explain but which nonetheless is undesirable in a random number generator. The implementation of a random number generator would appear to be a straight forward task, yet the complexity of software systems is such that even software implementations which have rigorous standards exhibit inconsistencies when examined closely.

What have we learned? Basically, we've learned that evaluating random number generators is a complex task and occasionally, there is more than a little magic associated with the effort. Using just the chi-square test we have had only one failure, i.e., the $RANDOM function in the MGM/CCSM implementation. It is something of a surprise that any of the MUMPS $RANDOM functions that we looked at exhibit non-random behavior given that each implementation purports to be ANSI MUMPS.

BASIC and C, unlike MUMPS, both provide systematic and reasonably well documented random number functions. In addition, BASIC and C, don't appear to have the surprises that the MUMPS implementations exhibit. I believe that MUMPS vendors should offer a similar functionality in $RANDOM.

7. References

[1]Knuth, D. E. *The Art Of Computer Programming Volume 2, Seminumerical Algorithms, 2nd edition.* Reading: Addison Wesley Publishing, 1981.

[2]Jain, R. *The Art Of Computer Systems Performance Analysis, Techniques For Experimental Design, Measurement, Simulation, And Modeling.* New York: John Wiley & Sons, 1991.

[3]National Bureau Of Standards, MUMPS System Laboratory. *MUMPS Validation Suite, Version 7.4, 1989 (for microcomputers)* NTIS Accession No. 2B90-500125.

[4]Stenn, Harlan, Personal Communication, December, 1992.

[5]I tested a MSM-UNIX implementation at the 1983 MUMPS User's Group Annual Meeting held in San Francisco and shared the result with David Marcus who was present at the time.

[6]Bruni, Vince, Micronetics Design Corporation, Personal Communication, December 8, 1992.

[7]Kernighan, B. W. & Ritchie, D. M., *The C Programming Language, 2nd edition*, Englewood Cliffs, NJ: Prentice Hall, 1988.

[8]Kochan, S. G., & Wood, P. H., *Topics In C Programming*, Indianapolis: Hayden Books, 1987.

Attachment A

N	Execution									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	3	7	4	8	3	7	4	8	5	9
1	2	2	2	9	6	4	3	1	2	9
2	8	4	3	7	2	7	6	1	2	7
3	3	1	4	6	8	1	4	6	5	7
4	8	6	1	9	7	5	0	8	5	2
5	8	3	7	4	1	8	2	9	6	3
6	6	3	8	0	2	4	9	1	7	9
7	1	8	1	1	2	3	6	7	7	8
8	8	8	3	8	2	7	2	6	6	1
9	0	6	3	2	1	9	7	5	4	3
10	3	0	7	4	2	0	7	4	7	4
11	0	9	7	2	7	2	0	5	4	9
12	8	8	4	7	9	1	7	9	8	0
13	7	4	0	7	5	2	8	5	4	1
14	0	9	4	0	5	1	6	1	5	0
15	1	4	5	4	3	2	3	2	5	4
16	8	2	5	9	2	6	9	2	0	3
17	5	4	4	9	5	0	0	6	9	4
18	3	5	1	4	6	9	6	8	4	7
19	9	8	5	8	2	6	2	6	3	6
20	6	6	4	4	3	3	1	0	7	7
21	4	0	9	5	2	8	7	3	2	9

Table 4 DTM - Ten OS Command Line Executions
Of MUMPS Code F N=0:1:21 W $R(10),!

Attachment A (continued)

N	Execution									
	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	6	8	4	9	4	2	1	2	9	0
2	2	6	8	1	8	1	5	1	6	1
3	2	3	3	7	6	2	8	2	2	4
4	4	4	8	0	4	7	1	8	9	2
5	8	0	2	3	4	5	6	6	6	5
6	3	6	4	2	5	8	1	4	7	5
7	6	4	7	7	7	0	3	8	8	1
8	8	2	6	2	5	3	1	8	9	2
9	7	4	7	3	6	2	8	9	9	8
10	5	0	2	0	3	5	7	7	5	7
11	3	3	1	0	2	0	7	7	5	2
12	5	1	7	2	7	1	5	8	1	2
13	6	5	8	3	9	1	2	2	4	4
14	1	2	0	2	3	6	0	1	4	9
15	7	8	9	8	1	8	5	4	9	2
16	6	1	1	4	9	1	2	3	9	3
17	4	0	8	1	8	7	7	8	3	4
18	6	6	9	0	7	7	7	5	2	0
19	3	7	3	0	0	1	2	8	2	7
20	9	6	6	0	8	8	8	0	4	4
21	5	2	4	6	2	8	4	0	5	5

Table 5 DSM - Ten OS Command Line Executions
Of MUMPS Code F N=0:1:21 W $R(10),!

Attachment A (continued)

N	Execution									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	4	0	6	5	3	9	6	6	0	4
1	6	7	4	1	9	8	8	6	1	0
2	7	2	7	9	5	3	3	3	4	4
3	4	5	5	8	2	6	4	1	8	0
4	5	6	8	8	2	8	8	9	2	8
5	0	2	5	9	7	0	8	9	9	3
6	9	5	6	1	4	8	5	1	5	9
7	3	0	3	8	9	0	0	7	4	5
8	1	0	8	7	9	2	1	7	6	9
9	3	3	0	9	6	5	4	3	1	1
10	2	1	8	3	1	8	8	6	5	2
11	8	0	9	9	6	5	2	0	2	1
12	6	3	2	9	4	2	4	5	7	3
13	8	5	6	5	0	5	3	0	6	1
14	7	9	1	6	0	1	8	6	0	1
15	3	4	0	8	8	1	9	1	1	4
16	4	8	9	8	8	5	6	0	6	4
17	5	5	3	2	5	8	7	2	0	3
18	9	8	6	9	9	6	6	1	3	0
19	7	0	9	5	2	7	0	5	1	6
20	3	6	4	9	7	5	6	8	4	2
21	4	4	9	3	8	8	0	4	9	1

Table 6 MGM/PC - Ten OS Command Line Executions
Of MUMPS Code F N=0:1:21 W $R(10),!

Attachment A (continued)

N	Execution									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	3	3	7	2	2	1	1	1	3	3
1	3	3	7	2	2	1	1	1	3	3
2	2	2	3	9	9	9	9	9	3	3
3	2	2	9	7	7	9	9	9	8	8
4	8	8	6	5	5	4	4	4	6	6
5	7	7	7	2	2	1	1	1	3	3
6	7	7	7	2	2	1	1	1	3	3
7	7	7	7	5	5	4	4	4	6	6
8	1	1	7	4	4	3	3	3	5	5
9	1	1	2	5	5	4	4	4	6	6
10	7	7	3	6	6	5	5	5	6	6
11	0	0	8	1	2	0	0	0	2	2
12	8	8	8	3	4	2	2	2	4	4
13	7	7	9	8	9	7	8	7	9	9
14	2	2	4	7	6	6	7	6	6	6
15	0	0	9	6	7	2	6	2	7	7
16	2	2	7	5	6	2	5	2	6	6
17	0	0	4	8	8	9	4	9	8	8
18	3	3	1	3	3	4	2	4	3	3
19	5	5	6	3	3	9	2	9	4	3
20	4	4	8	6	6	4	5	4	7	6
21	5	5	2	7	7	5	6	5	3	7

Table 7 MGM/CCSM - Ten OS Command Line Executions
Of MUMPS Code F N=0:1:21 W $R(10),!

Attachment A (continued)

N	Execution									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	7	7	7	7	7	7	7	7	7	7
1	7	7	7	7	7	7	7	7	7	7
2	5	5	5	5	5	5	5	5	5	5
3	6	6	6	6	6	6	6	6	6	6
4	6	6	6	6	6	6	6	6	6	6
5	3	3	3	3	3	3	3	3	3	3
6	5	5	5	5	5	5	5	5	5	5
7	5	5	5	5	5	5	5	5	5	5
8	1	1	1	1	1	1	1	1	1	1
9	9	9	9	9	9	9	9	9	9	9
10	9	9	9	9	9	9	9	9	9	9
11	5	5	5	5	5	5	5	5	5	5
12	1	1	1	1	1	1	1	1	1	1
13	7	7	7	7	7	7	7	7	7	7
14	3	3	3	3	3	3	3	3	3	3
15	9	9	9	9	9	9	9	9	9	9
16	3	3	3	3	3	3	3	3	3	3
17	7	7	7	7	7	7	7	7	7	7
18	4	4	4	4	4	4	4	4	4	4
19	4	4	4	4	4	4	4	4	4	4
20	3	3	3	3	3	3	3	3	3	3
21	7	7	7	7	7	7	7	7	7	7

Table 8 MSM - Ten OS Command Line Executions
Of MUMPS Code F N=0:1:21 W $R(10),!