Comparative Performance of COBOL programs on Mini vs. Large Computer Systems

Paul J. Jalics
Cleveland State University

Abstract

The comparative performance characteristics of COBOL programs in a small versus large computer systems are investigated. The vehicle consists of a set of synthetic benchmark COBOL programs, each measuring a particular aspects of COBOL programs; and in addition a large actual COBOL program. Measurement of the CPU execution time and the elapsed clock time for various COBOL computations, data manipulation, and input/output is made on both a large scale computer (IBM 370/158) and a minicomputer (Texas Instruments TI980). Results of a number of such experiments are presented and comparisons made between results gotten from the two systems.

Introduction

The performance aspects of business programs have, to a considerable extent, been ignored. The author has sought to explore this area (see [1], [2]) and gain some insights into the performance characteristics of business programs, the majority of which are written in the COBOL programming language.

The original domain of COBOL is in large scale computers but now an increasing number of minicomputers also include COBOL compilers, although typically only a subset of COBOL is supported. Thus an opportunity was seen for gaining an insight into the comparative performance of mini vs. large scale computers executing identical COBOL programs. The particular large computer available was an IBM 370/158 running with OS/VS1 and VS/COBOL (release 1.2) and the minicomputer available was a Texas Instruments TI980 with 64K words of memory running with DX980 *D operating system and COBOL/980 (level 1.0).

First, a set of existing synthetic benchmark COBOL programs (see [2]) were run on the mini and then compared with results of the same programs run on the IBM 370. In addition, a large COBOL program called ASM990 consisting of 2530 source lines, the purpose of which was to assemble TI990 programs (i.e. a cross-assembler originally executing on IBM370) was compiled and executed on both machines. Thus ASM990 was used as a test vehicle for comparing compilation and execution speeds on the two machines.

1.0 Machine characteristics

Most essential in evaluating the experiments in this paper are the physical characteristics of the two computers involved. No introduction to the IBM 370 system is necessary but a few words are required to give a general idea of the Texas Instruments TI 980 minicomputer.

The TI980 is a general purpose 16-bit word oriented minicomputer with a dedicated register architecture including 8 registers. The machine has a rich variety of addressing modes with one and two word instructions (word = 16 bits). Notably missing in the instruction set are byte load/store and other byte manipulation instructions-all such manipulation is done via load word and shift instructions. The TI980 DS31 disk is made by Diablo and is typical of disk devices connected to minicomputers. The disk is attached to the TI980 via a direct memory access port.

Some typical instruction execution times for the two machines and relevant disk device characteristics are listed below in Table I.

Characteristic	IBM 370/158	TI980
16-bit load register	.933 microseconds	1.75 microseconds
32-bit load register	.588 microseconds	2.75 microseconds
16-bit add	1.16 microseconds	1.75 microseconds
32-bit add	.933 microseconds	2.75 microseconds
16-bit multiply	1.41 microseconds	2.25 to 6.25 microseconds
disk type	IBM 3350	DS31 (one fixed, one remov.)
disk transfer rate	1.2 megabytes/sec	0.1 megabytes/sec
max disk latency	16.7 milliseconds	40 milliseconds
arm positioning	10 to 50 milliseconds	15 to 135 milliseconds
track capacity	19,069 bytes	5632 bytes
# of tracks per cyl	30	2

TABLE I. Computer Device Characteristics

1.1 The Synthetic COBOL Benchmarks

1.1.1 Data Field Types and Sizes

The first experiment was concerned with the amount of compute time used in executing typical arithmetic statements. The statement ADD A TO C. was chosen and the CPU time measured for 50,000 executions of the statement with both A and C being numeric data-items of the same size. The experiment was repeated for every numeric field size from PIC 9 to PIC 9(18) and individual sequences of 18 experiments for each of the three data types, namely USAGE DISPLAY, USAGE COMP-3, and finally USAGE COMP. Figures I and II show the results of this set of experiments on the IBM/370 VS/COBOL and the TI980 COBOL, respectively. A number of observations can be made about the data shown in Figures I and II:

a. On the IBM/370 VS/COBOL one can see that: (1) usage of COMP is by far the most efficient for fields up to and including PIC 9(9) (or PIC S9(8), in general about twice as fast as the next competitor which is COMP-3; (2) usage of COMP for fields greater than PIC 9(9) is incredibly inefficient being in general about ten times slower than the other data types, this is due to the inefficient nature of the library subroutine called for arithmetic in this size range; (3) USAGE of COMP-3 has the most uniform performance over different field sizes, it is in general about twice as fast as DISPLAY and one half as fast as COMP for smaller field sizes. COMP-3 is supported by the decimal business instructions on the 370; (4) Odd field sizes are preferred for COMP-3 since an extra instruction needs to be executed for even size fields to zero out the unused uppermost digit in the highest order byte.

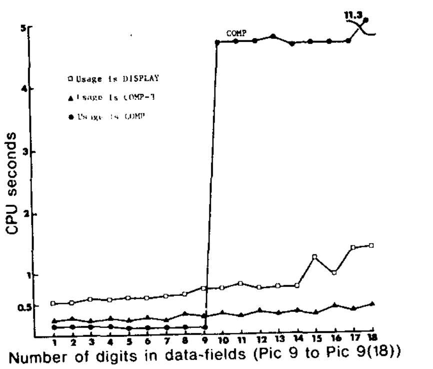


Figure 1.
IBM370 VS/COBOL CPU Execution Time vs Data Field Size for ADD A to C (50,000 Executions)

b. On the TI 980 COBOL one can see that: (1) COMP fields are most efficient since they are backed by hardware instructions: up to PIC 9(4) by 16-bit arithmetic, from PIC 9(5) to PIC 9(9) by 32-bit arithmetic instructions, and for larger fields by software library routines; (2) COMP-3 fields are not supported by the hardware, nor the COBOL compiler; (3) DISPLAY arithmetic is handled via library subroutines and are phenomenonally slower than COMP, in general 150 to 400 times slower than COMP for field sizes PIC 9(9) and smaller where DISPLAY compute time increases linearly for DISPLAY and stays essentially constant for COMP using the hardware instructions, and is 20 to 32 times slower than COMP for field sizes greater than PIC 9(9) where both COMP and DISPLAY are using library subroutines.

c. In comparing the COMP arithmetic on the two machines, one sees that computations on the minicomputer TI980 are only 1.4 to 2.5 times slower than on the IBM 370. This ratio is in line with the execution speeds of the two machines involved and is the only easy area of comparison since both machines have hardware instructions for this data type (even for fields over 32 bits there is considerable hardware support by using a number of smaller binary fields).

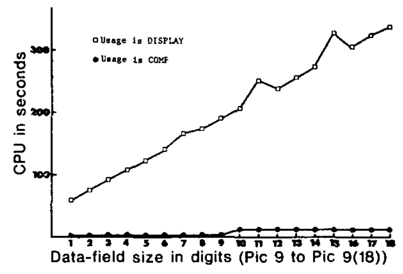


Figure 2.
TI980 COBOL CPU Execution Time vs Data Field Size for ADD A to C (50,000 Executions)

d. In comparing the DISPLAY arithmetic on the two machines, one finds that execution on the minicomputer is 100 to 360 times slower than on the 370. The reason for this factor of 100 difference between COMP and DISPLAY is due to the poor hardware support for character arithmetic and character manipulation as opposed to the considerable hardware support for both character manipulation and character arithmetic to be found on the IBM/370. Note that on the 370 DISPLAY fields are converted to packed decimal fields via the PACK hardware instructions and the result is converted back to DISPLAY character format via the UNPACK hardware instruction. The TI980, like most minicomputers, has no packed arithmetic instructions, nor PACK and UNPACK instructions, nor even good character manipulating instructions.

e. The set of experiments run are, of course, only a sampling of the performance of arithmetic statements with some deviations to be expected. For example, on the 370 a similar set of experiments were run using the statement ADD A, B GIVING C which specifies almost the same computation and the results were identical except that for the COMP-3 fields, the execution time just about doubled, this due to some strange code generation for COMP-3 fields in connection with GIVING clauses. Interestingly, the same GIVING experiments run on the TI980 gave execution times 25% less than for ADD TO for DISPLAY fields and slightly less for COMP fields.

f. On the 370 it is always more efficient to use signed fields rather than unsigned ones for all usages with a 10 to 28% savings in CPU time when compared to unsigned fields. This happens because the code generated by the COBOL compiler will after each arithmetic operation zero out the sign of any unsigned numeric field. On the other hand, the execution time of signed and unsigned fields is identical on the TI980 COBOL.

g. A sequence of computations equivalent to COMPUTE F = A * B + C / D - E. was executed 50,000 times on both machines with each data-field being PIC 9(8). This sequence contains all the arithmetic operators and executed in about 5.5 CPU seconds on the IBM370 and in 830 CPU seconds on the TI980 making the minicomputer 150 times slower than the large computer. Thus the range of 100 to 360 difference for DISPLAY fields mentioned earlier is seen as a reasonable estimate.

1.1.2 Data MOVEs and Alignment Characteristics

Data alignment should play a significant role in execution speed of programs. For example, on the IBM 370 the main memory organization is such that half-word alignment (address even), full-word alignment (address divisible by 4), and double-word alignment (address divisible by 8) are important. Since all 01 level data-fields start at the double-word alignment, the COBOL user has complete control over the alignment of all data-fields. On the TI980 minicomputer, the only alignment of interest is word alignment (data starts on a word boundary) and in COBOL all 01 data-fields start word aligned. The following measurements of alignment performance were conducted:

a. The complete set of ADD A TO C. was originally run with A and C both completely aligned since each was declared at the 01 level. The declarations were changed so that both A and C started as the second character of an 01 record and thereby having no alignment. The set of experiments was rerun on the IBM 370 and very surprisingly the results were identical to those withthe aligned fields done earlier. After some looking at the functional characteristics of the 370/158, it was found that a cache memory of 8 K bytes exists between the CPU and main memory and thus the very first of the 50,000 executions of ADD A TO C. caused A and C to be brought to the cache and thus the succeeding 49,999 executions caused no memory access. It seems that the performance of unaligned fields used repeatedly is thus not measurably different from aligned fields. A number of other experiments were run to show the effects of unaligned fields versus aligned ones and were unsuccessful. The author still feels that alignment of data-fields plays a role in performance but the measurement techniques used were unable to show quantitative evidence of the differences.

b. The alignment measurements on the TI980 COBOL were very simple. The ADD A TO C. were repeated with A and C being not word aligned and the results are as follows: (1) for DISPLAY fields, unaligned fields are slightly slower and the difference is constant over the range of field sizes being 1.3 CPU secs where the total CPU times ranges from 57.1 CPU seconds for PIC 9 to 339.5 seconds for PIC 9(18); (2) for COMP fields up to PIC 9(9) the unaligned field arithmetic is about 20% slower, mostly because additional instructions have to be executed to pick out parts of the COMP field from different words and put them together via shift instructions; (3) for COMP fields over PIC 9(9) the unaligned fields are 4% slower, the effect of unalignment being less important than the overhead of the generalized library routine doing the arithmetic.

c. A number of experiments were run with MOVEs of various fields, aligned and unaligned of various lengths. These experiments showed no appreciable difference in performance on the IBM 370 probably for the reasons desribed above. On the TI980 minicomputer, however, the differences were dramatic. A MOVE X TO Y. was executed 50,000 times with X and Y being PIC X(4). The word aligned X and Y experiment executed in 0.5 CPU seconds and the one with unaligned X and Y executed in 30.9 CPU seconds. The difference here is that whenever the length of the sending and receiving fields is even and starts on a word boundary a loop of word load and stores is generated by COBOL, whereas if the fields are not aligned, then the generalized library subroutine is called thereby incurring tremendous overhead with character by character moves on a word machine without load byte/store byte instructions. The experiment was repeated with X and Y being declared PIC X(6) and the execution times were 1.4 seconds for aligned versus 44.2 seconds for unaligned fields. Thus the performance of unaligned fields in MOVE statements is likely to be 30 to 60 times slower than for aligned fields.

1.1.3 Subroutine Calling Overhead

The CPU time spent in entering a subprogram and exiting a subprogram are important factors in the design of programs. COBOL has internal subroutines (PERFORM verb) with no parameter passing and external subroutines with parameter passing. A number of measurements were made to determine the overhead of the various subroutine calling and looping mechanisms in COBOL:

a. Internal subroutines via PERFORM statements are relatively fast and take about 4 microseconds on the 370/158 in a PERFORM ... TIMES statement. To accomplish the same via PERFORM ... UNTIL ... appears to be significantly less overhead where the experiment consisted simply of performing an arithmetic statement 10,000 times. Slightly faster still is a simple PERFORM with testing and a conditional GO TO to the beginning of the performed paragraph. This strange form had to be resorted to in COBOL code to be run on both the IBM370 and the TI980 since the TI980 COBOL does not implement the PERFORM ... UNTIL (this being the only major shortcoming seen in TI980 COBOL features).

b. Internal subroutines via PERFORM ... TIMES take about 30 microseconds on the TI980 COBOL this being 7.5 times as long as that on the IBM370. The simple PERFORM with the test and looping within the performed paragraph is exactly as fast on the TI980 as the PERFORM ...TIMES.

c. External subroutines via CALL statements take about 85 microseconds on the 370/158 and are thus 21 times slower than internal subroutines. Note also that the external subroutine calling does not include the looping included in the timing of the internal subroutines. The slowness of the external subroutine mechanism is due mainly to the lengthy prologue and epilogue routines that are executed upon entry and exit of a subprogram. Accessing passed parameters from the main program appears to have little or no overhead associated with it as all that needs to be done is a base register load.

d. External subroutines via CALL statements on the TI980 take approximately 40 microseconds which is only 33% more than the PERFORM ... TIMES mechanism. Thus in contrast to the 370 subroutine calling overhead which is 21 times as large for external routines, the overhead here is only slightly greater for external subroutines (a factor of 1 to 2). This fact is very important in making the TI980 COBOL a viable product because, as will be discussed later, the compile speeds for large COBOL programs on the TI980 are prohibitive, and structuring a program into small subroutines is essential. Note also that the external subroutine call overhead for the TI980 is less than half that for the 370/158, one of the few experiments where the TI980 COBOL was actually faster than the 370. This is due to the lack of extensive prologue and epilogue routines in the TI980 COBOL.

1.1.4 Input/Output Speeds

In order to gain some insight into the workings of the COBOL I/O systems, a program was written which writes 1,000 records to a temporary disk file, then closes the file, reopens it for input, and reads the file in. There are some difficulties in one-to-one comparisons of timings on the IBM 370 versus the TI980 because: (1) the difference in disk devices used by each; (2) the 370 was not, in general, available on a standalone basis-but rather only via multiprogramming with the system load an unknown variable whereas the TI980 whereas it is capable of multiprogramming was standalone at the times of the measurements; (3) finally, the CPU execution time is not really available to the program on the TI980 (although the clock time is available and is an accurate measure of CPU execution time for segments of code containing no I/O on an otherwise idle system such as all of the experiments above). The I/O experiments are detailed below:

a. The first experiments run on the IBM 370 measured the CPU time and elapsed clock time for the execution of 1,000 WRITE statements. This experiment was performed in normal multiprogramming so the elapsed clock time is merely an indication as to how long the experiment required under a "normal" system load. One big variable in I/O processing is the blocking factor so the experiment was repeated for various block sizes starting with BLOCK CONTAINS 1 RECORDS to BLOCK CONTAINS 140 RECORDS. The CPU time used was of interest in order to find out the CPU resources spent in the I/O access methods and also to see what effect block size has on the CPU resources required. Figure III shows the results.

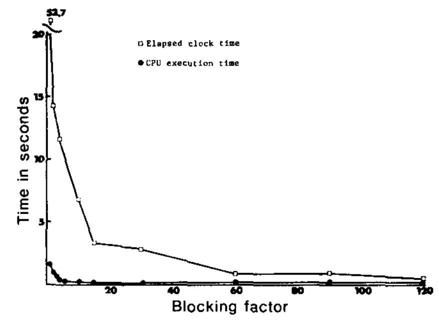


Figure 3. IBM370 VS/COBOL Time to Write 1,000 Records vs Blocking Factor

Among the observations one can draw from Figure III are: (1) the CPU time required decreases dramatically from 1.65 seconds for a blocking factor of 1 to about .15 CPU seconds for blocking factors over 60. Thus the CPU time spent in the access methods can be reduced by 90% if the file is properly blocked; (2) The elapsed clock time also shown in Figure III shows the same kind of pattern with 50 seconds for Blocking factor of 1 then being reduced to .9 seconds for a blocking factor of 60.

b. The 1,000 WRITEs experiment was also run on the TI980 COBOL where BLOCK CONTAINS is ignored by COBOL but where blocking can be specified by job control. The results are shown in Figure IV and as mentioned above, these experiments were run on an otherwise idle system and the CPU time is not shown because it is not available to TI980 programs. The elapsed clock time starts with blocking factor of 1 at 65.5 seconds and goes down to 18 seconds for a blocking factor of 35 (large blocks are not allowed). Thus a reasonable comparison for a sufficiently large blocking factor might be the elapsed time of 2 seconds for a blocking factor of 30 on the IBM 370 versus the elapsed time of 18 seconds on the TI980 thus the minicomputer is 9 times as slow as the 370 (this in the face of multiprogramming for the 370 and standalone on the TI980). The device characteristics of the two disks involved are themselves sufficient to explain the factor of 9 difference (see Table I for device characteristics). Other contributors to the difference might be I/O and scheduling overhead differences in the two operating systems involved.

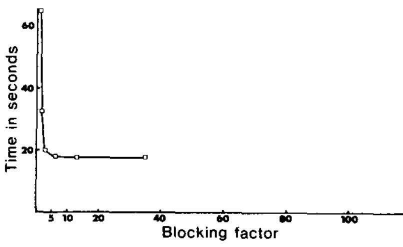


Figure 4. TI980 COBOL Elapsed Time vs Blocking Factor to Write 1,000 Records (80 Characters each)

1.2 The ASM990 Benchmark

ASM990 is a cross-assembler for a TI990 microcomputer and as such is not untypical of business data processing programs. It inputs a TI990 Assembly language source file, creates a temporary file containing partially assembled code, then reads the temporary file, finishing the assembly by generating an assembled listing and an object file ready for the TI990 loader. ASM990 was written to be as efficient as possible and still be 100% in COBOL with no assembler subroutines. Efficiency was designed into the data types (COMP wherever possible), data-field sizes (as short as possible), data-field alignments (all alignments observed for both IBM370 and TI980), table searching techniques (binary search for op-code table, hashing plus binary trees for symbol table), file blocking (blocking factor 15 for object output file and 20 for temporary file).

Thus ASM990 was used to make measurements that seek to: (1) compare COBOL compilation speeds on the IBM370 versus that on the TI980; (2) relate performance of ASM990 to other similar products on both machines; (3) measure COMP variables to DISPLAY.

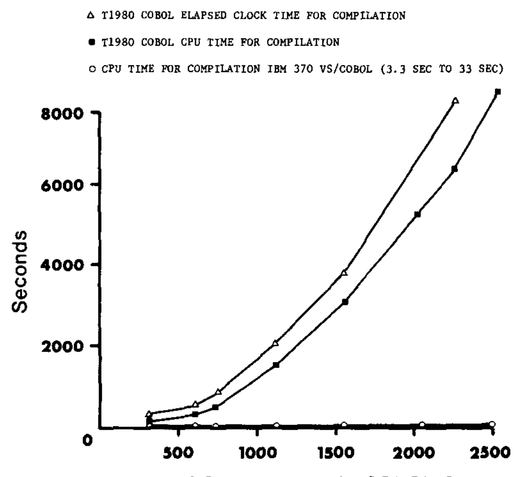
1.2.1 COBOL Compilation Speeds

In order to get an idea of compiling rates, various subsets of ASM990 were compiled to yield compilation times for programs of 295 lines, 600 lines, 730 lines, 1100 lines, 1500 lines, 2200 lines, and finally 2530 lines. Compilations were carried out on both the IBM370 and the TI980 and the results are shown in Figure V. The relationship between the timing of the compilations on the two machines is truly remarkable. The full 2530 line program takes 10,327 seconds elapsed clock time on the TI980 as compared to about 60 seconds on the IBM370. CPU execution time on the TI980 is 8442 seconds versus 33.2 seconds on the 370 making the 370 254 times faster than the TI980 although this factor gets smaller with program size so that for a 295 line program the factor is only 32. One can easily see from looking at Figure V why it has become so customary to develop software for minis via cross-compilers running on large computers. One can also see that in order to program effectively on a mini such as the TI 980, one must develop software in modules of not more than a few hundred lines each.

1.2.2 ASM990 Execution Speeds

ASM990 was then compiled in its entirety and a load module created on both computers. The experiments on each computer are described separately:

a. Input source decks were artificially generated to provide sample input data for the execution of ASM990. The input decks varying in length from 500 to 2500 lines each contained TI990 Assembly Language statements. Execution times of ASM990 on the IBM370 with various length input decks are shown in Figure VI. In order to be able to compare ASM990 performance

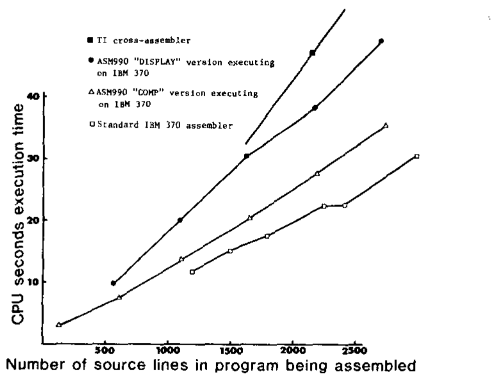


Number of Source Lines in COBOL Program

Figure 5.
Compilation Speed vs COBOL Program Size

with other similar products, another set of input source decks was created for the IBM370 Assembler and used as input data for the standard IBM370 Assembler. Results are depicted in Figure VI and show that the 370 Assembler executes with 20 to 25% less CPU time than ASM990. This result is encouraging since the IBM370 Assembler is written in machine language and probably written with some emphasis on performance in order to be able to assemble large pieces of software like the operating system reasonably. Also important is that the execution time increases at about the same rate for ASM990 and the IBM Assembler. Finally, a third set of input decks was created for a TI provided cross-assembler for the TI980 which was also written in COBOL. Figure VI also shows the performance curve for this assembler which is significantly less efficient than either of the other two and in particular the slope of the curve indicates a much faster degradation in performance.

Thus, it is possible to write a large software product in COBOL and with appropriate attention to efficiency considerations get a product with performance near to that of a well written machine language written.



Number of source lines in program being assembled

Figure 6.
Assembler Execution Speeds vs Source Program Size on IBM370

b. The ASM990 load module created earlier on the TI980 was then executed with the same input decks used on the 370. The resulting curve showing execution times is shown in Figure VII.

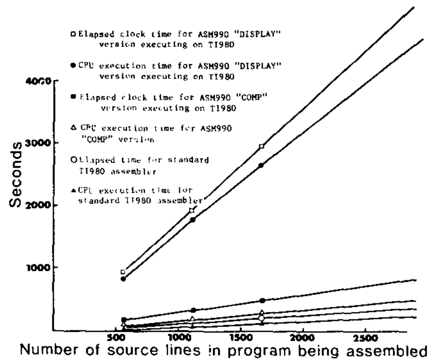


Figure 7.
Assembler Execution Speeds vs Source Program Size on TI980

The CPU execution time is 13 to 16 times what is was for the 370 ASM990 - not an unreasonable figure. The ratio of elapsed clock time for a program of 1500 lines is 12.5 longer on the TI980- also a reasonable figure.

In order to compare the TI980 ASM990 performance with other similar products on the TI980, the same set of input decks used to drive the 370 TI Cross-assembler were now run on the standard TI980 Assembler to produce the curve shown in Figure VII. The results show that the standard TI980 Assembler executes in 42% of the CPU execution time required for the TI980 ASM990 (and similarly 43% of clock time). Thus there is additional overhead in the COBOL written Assembler (ASM990) but not disastrously more than in the standard assembler. Note also that the difference here is much greater than the difference between IBM370 Assembler and IBM370 ASM990.

1.2.3 ASM990 "DISPLAY" Version Execution Speeds

As a final experiment, all the variables, tables, counters, etc. in ASM990 were changed from COMP to the default DISPLAY. The new slower ASM990 was then run again on both machines and execution speeds are recorded in Figure VI for the IBM 370 version and in Figure VII for the TI980 version.

a. On the IBM370 "DISPLAY" Version of ASM990, execution times increased considerably by 33% and the rate of increase was faster as program size increased. In order to be able to compare the elapsed time of execution of the original ASM990 vs. the "DISPLAY" version, both were run on an otherwise idle 370 and interestingly, the elapsed clock time was just about identical for the two versions (for 1500 lines to be assembled 42 for the original ASM990 vs. 43 seconds for the "DISPLAY" version). This led to the conclusion that ASM990 on the 370 was very much an I/O bound job since an increase of 33% in CPU time had no difference on the elapsed time. Note, however, that almost all programs are run in multiprogramming on the 370 so the 33% savings in the original ASM990 will still be reflected in the computer accounitng charges and a lessening of total system load.

b. On the TI980 "DISPLAY" Version of ASM990 execution times increased more dramatically: a factor of 8.5 to 9.7 for CPU execution time, and a factor of 4.9 to 6 in elapsed clock time. Thus whereas ASM990 was reasonably efficient to use, now one could notice a two to three second pause between successive input cards being read (with the CPU indicator being solidly active) and likewise a similar pause between every line printed. Thus whereas degradation of performance on the 370 was restricted to 33%, the degradation on the minicomputer was more like 600%, i.e. data types are much more critical on the mini than on a large computer. Also clear is that whereas ASM990 was an I/O bound program on the 370, it is more CPU bound on the TI980.

Also noteworthy is that the memory requirements for the ASM990 load module increased by 55% when the "DISPLAY" version was created. This is a very significant point because most minicomputers do not have memory space to spare.

Finally, if the simple change of COMP to DISPLAY caused such dramatic performance degradation, other aspects such as data-field sizes, alignments, and blocking factors are also expected to have much more dramatic effects on performance than may be expected on the large computer.

Conclusion

A few of the trends and general conclusions that can be drawn from the experiments performed are summarized below:

1. Data types and sizes play an important role in program performance. In particular, an awareness of hardware supported data types and sizes is very necessary. Performance effects of data types and sizes are much more dramatic on the minicomputer than on the large system.

2. Alignment of data-fields is definitely significant on the minicomputer and much less important on a large system, especially one with a cache memory.

3. Subroutine calling overhead is relatively small on the minicomputer and this contributes greatly to the viability of the mini COBOL since there is little performance penalty to pay for modularization.

4. I/O is significantly slower on the minicomputer and blocking is equally important on both systems.

5. Large software products written in COBOL are equally viable in both systems but an awareness of efficiency characteristics is almost mandatory on the mini COBOL whereas that is less so on the large system.

6. COBOL compilation speeds are quite unreasonable on the mini for any programs over a few hundred lines. This can be a problem if there are extensive record descriptions and tables which are required in several modules and may require several different record descriptions each one detailing only those data-fields accessed by that module with the rest FILLER so as to reduce the size of the DATA DIVISION entries to as few lines as possible.

7. The mini COBOL programs tend to be much more CPU intensive and therefore attention to arithmetic, data manipulation, and testing efficiencies is much more important on the mini.

8. The efficiency of programs is seen as a much more immediate problem on the minicomputer involving minutes or hours of actual waiting whereas such immediate feedback is often missing in the large computer OS batch systems.

[1] P. J. Jalics "Improving Performance the Easy Way," DATAMATION, April 1977.

[2] P. J. Jalics "Benchmarks for Measuring Performance of COBOL Implementations," ACM Computer Science Conference, 1978, Detroit, Michigan.