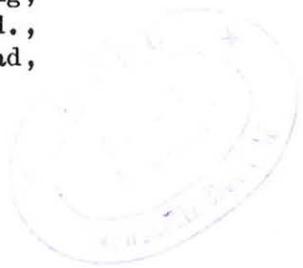


SYSTEMS EVALUATION

Mr. A. B. Cleaver

Large Systems Marketing,
IBM United Kingdom Ltd.,
389, Chiswick High Road,
London, W.4.



Abstract:

Some methods of evaluating the performance of computer systems are studied, and their respective merits and shortcomings are discussed.

Rapporteurs:

Mr. L. Waller
Mr. D. Appleton



Introduction

Many of the techniques developed and used in the past few years in order to evaluate the performance of computer system are largely irrelevant to the real problem, and it is essential that better approaches be found. It must be acknowledged that the method used will depend on the reasons for carrying out the evaluation, and also on the tools available to the investigator. For instance, simulation design studies may be useful to a manufacturer designing new systems, but since the ratio of real time to model time may be of the order of 10^4 , such methods are unacceptable in a University environment.

System Selection Criteria

When a prospective customer considers a given system, some of the points to which he gives pre-eminence are as follows.

Firstly, he must know within reasonable limits the function to which the system is to be put. The performance of a system is closely bound in with its work-load profile - the frequency of the different types of job which it must process. (It would greatly benefit future design if customers would specify accurately their current work-load profile.)

Secondly, the reliability of the system is important. Even if one system is faster in all respects than another, it does not follow that the first is better, for if it lacks reliability its increased speed may be wasted by periods spent 'down'.

Thirdly, operability must be taken into account. In the real world this aspect is important as the system performance can be materially lowered by difficulty encountered by those who have to deal directly with the machine.

It follows from the interdependence of performance, function, reliability and operability that we cannot hope to measure an 'absolute performance' for there is no such thing. In addition to the above, growth potential, supporting services and other similar aspects of a system must be considered in its evaluation, for performance is also dependent on these criteria. Table 1, which was produced by an American consultancy firm, shows an attempted breakdown into, and weighting of, a number of factors. (The figures are not necessarily a reflection of the views of the author.)

Factors affecting performance

The major factors affecting performance may be listed as follows:

CPU power, storage capacity,
channel/device power,
operating system efficiency,
compiler efficiency,
work-load profile,
system reliability,
programmer competence, and
operator competence.

The term 'CPU power' is used in preference to mere 'CPU speed' as it embraces important related factors such as the size of the instruction set.

Storage capacity is an increasingly important factor with the advent of multiprogramming, and this is one of the places where cost is liable to influence performance considerably. Many UK installations have only about 50% of the storage capacity required for optimal performance, largely because of the cost factor.

Channel/device power considerations are covered in the paper by Clark elsewhere in this document.

Very little work has been done in the evaluation of the efficiency of operating systems, and it would appear that the Universities could contribute materially to this side of performance evaluation. The factors acting have not yet been adequately investigated, or even comprehensively listed.

Some work has been done on the problem of comparing the efficiency of the code produced by various compilers for the same source program (Atlas Laboratory, Chilton), and this attacks one part of the problem, but here too there is scope for research.

System reliability has been mentioned previously, and work-load profile will be discussed more fully in the context of benchmarking. Programmer and operator efficiency is more difficult to quantify, but some beginnings have been made to this end, and it is likely that more will be done as interest develops in time-sharing and multi-access systems.

Methods of comparing systems

Traditional methods of comparing the performance of different systems include comparison of add-times or cycle-times. Tables 2 and 3 show that both of these methods are of doubtful value since large fluctuations are apparent between the various times, and results would probably be much worse for systems which, unlike those used in the examples, are not fully compatible.

An extension of these techniques is the Gibson mix. Table 4 shows the 'Gibson mix III', one of the eight mixes derived by Gibson with different commercial/scientific weighting. The method originated with Sweeney in 1955 and was employed on the IBM 650 and IBM 704 systems. The system architecture, the power of the instruction set and the ability to overlap instructions (which makes execution time depend on context in the instruction stream) are not taken into account by the Gibson mix, and this means that this device has very limited use for modern machines. Even on the basis of instruction frequencies problems arise as recent analysis of programs on System/360 has shown. (Table 5.)

Other mixes have been developed, but even if these were completely accurate they would only be measuring CPU power and would take no account of I/O or storage capacity.

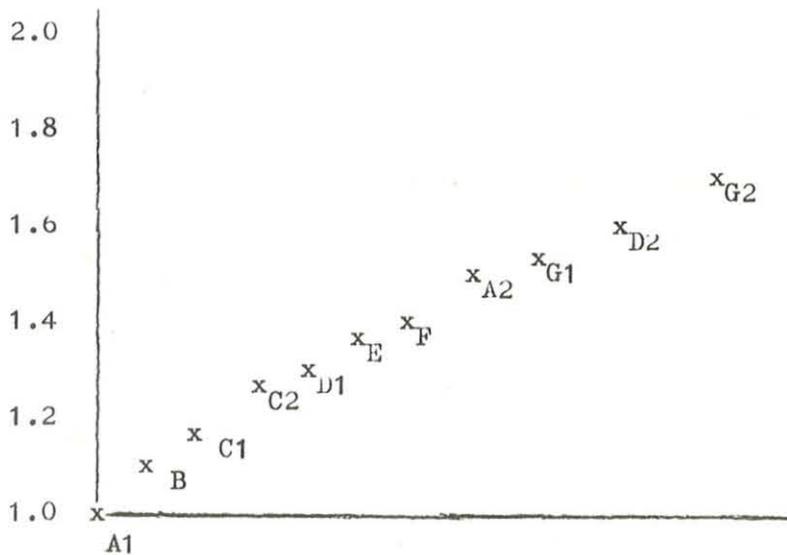
Software comparisons

The foregoing techniques are very largely measures of hardware performance, but it is likely that it is software which today plays the leading role; consequently attempts are frequently made to compare compilation times. Table 6 shows compilation times for two Fortran source programs using different compilers on identical System/360 configurations. This, however, is less than half the story, as these compilers differ dramatically both in language features supported and efficiency of the code produced. Similarly, figure 1 indicates the variation in object code efficiency produced by different compilers on the same system, and the variation across different configurations.

Another possible analysis is of the number of instructions executed in running a program. The ratio between the least efficient compiler and the most efficient, on this basis, for the program of figure 1 is a factor of 5.

Fortran Object-Code Analysis

Instruction Storage Space (Bits)



A letter refers to a particular configuration, and a number to a particular compiler on that machine.

Figure 1

Such a software approach may be valuable if the amount and type of work being done on available compilers is known, and since it is economic on machine time, though not in total time, it may appeal to British systems analysts.

Benchmarking

Benchmarking is defined as 'actually running representative user jobs on the proposed configuration, using the software suggested in the normal environment'. To obtain accuracy of results there must be one official co-ordinator who sets a completion deadline and gives all the necessary rules in writing. Nearly all large systems recently installed in the United Kingdom have used some sort of benchmarking, and advantages of the method include its relatively economic use of machine time and the

opportunity it offers of becoming acquainted with a new system before deciding to have it installed. Unlike the methods discussed previously, benchmarking is capable of taking into account most of the factors contributing to performance. This is shown schematically in Table 7.

What constitutes a representative stream of jobs can be determined from the operating system or accounting routine employed. This imposes little overhead on the system, and the following statistics on the work-load profile should be easily obtained:

elapsed time and CPU time for all system jobs and programs,
storage and device requirements for jobs,
total time and frequency of use of programs.

Benchmarking must be regarded as the best tool available in system performance evaluation at the present time.

Other methods

Some manufacturers and software houses now provide packages which may be useful in calculating performance, but although the use of such packages is to be encouraged, care must be taken not to collect more data than can be analysed. It should also be noted that overheads on the system are high, and the running of such additional programs may substantially alter the system load.

Hardware devices, which do not affect the system under investigation are also available. These are more accurate than software packages, giving a resolution of 1 μ sec., and combinations of different system states can be considered. It has been possible to use such devices to obtain interesting results on I/O efficiency.

So far it has been assumed that the system to be investigated actually exists, at least in prototype, but it is important to be able to estimate a system's likely performance before it is operable. A trace timer approach is advocated. Table 8 shows some of the statistics used in the design of the IBM 360/85. The analysis of data is unfortunately very time-consuming, the figures in Table 8 being derived from a mere quarter of a million instructions.

Other types of simulation to compare systems are possible, but with the problem of gathering relevant data on the hardware, and more especially the software, of each system, combined with the complexity of the overall problem, and the amount of work involved, the method is, on the whole, economically non-viable.

Time-sharing systems

Whereas in most batch-oriented systems the idea of a 'good performance' is fairly intuitive, this is not the case in time-sharing systems. One possible measurement is response-time, but it is difficult to know how to weight the response-times to trivial commands, non-trivial commands and run commands to give a useful mean. Another approach is the measurement of user satisfaction, or 'the smile factor', but this cannot easily be quantified. Perhaps it will be possible to devise a good method of evaluating multi-access systems by building on the procedure adopted by IBM in developing TSS. This involved the definition of a standard task and the determination of the number of such tasks which could operate simultaneously under the given system without saturating it. This is also an expensive approach and needs a great deal of development if it is to become worthwhile.

Summary

It was demonstrated that system performance is closely related to the function, reliability and operability of the system as well as depending on such aspects as CPU power, storage capacity and computer efficiency. Work-load profile was ascribed major importance, and the possibility of calculating an absolute value for performance was rejected.

Traditional methods of performance evaluation, such as the Gibson mix, were discussed and shown to be unsatisfactory. The inherent difficulties in measuring compiler efficiency were stressed.

Benchmarking was recommended as the best available method for performance evaluation, and attention was drawn to the possibility of using a software package of hardware device.

The basis of a method of evaluating time-sharing systems was suggested.

VENDOR EVALUATION AND SELECTION CRITERION

SUMMARY POINTS

I	(20%)	Hardware	
		A. Vendor Recommended System	(50%)
		B. Total Vendor Capability and Expandability	(50%)
II	(20%)	Programming Systems	
		A. Vendor Recommended System	(50%)
		B. Range of Operating System Environments	(30%)
		C. Total Vendor Programming Systems Capability	(20%)
III	(10%)	System and Programming Interrelation and Design	
		A. System Balance	(50%)
		B. Software Compatibility to Other Systems	(25%)
		C. System Complexity and Reliability	(25%)
IV	(15%)	Benchmark Performance	
V	(15%)	Vendor Capability and Support	
		A. Conversion Assistance	(25%)
		B. Maintenance	(25%)
		C. System Design Assistance	(15%)
		D. Education	(10%)
		E. General Vendor Capability	(25%)
VI	(20%)	Direct Total Cost	
		A. Recommended System	(30%)
		B. Additional Equipment	(10%)
		C. Personnel	(10%)
		D. Conversion	(20%)
		E. Maintenance and Extra Shift	(10%)
		F. Other Cost Considerations	(20%)

Table 1

ADD-TIME COMPARISON

INSTRUCTION	SYSTEM/360 MODEL		
	65	75	85
AU	2.57	6.88	7.96
A	2.85	5.71	10.25
AP	3.48	3.65	16.91
ADR	4.55	9.22	26.13
AR	5.00	8.12	40.62

ALL FACTORS ARE RATIOS TO MODEL 50

Table 2

CYCLE-TIME COMPARISON

	SYSTEM/360 MODEL				
	<u>50</u>	<u>65</u>	<u>75</u>	<u>85</u>	<u>195</u>
MAIN STORAGE CYCLE	2us	750ns	750ns	1040ns	756ns
DATA PATH WIDTH (BYTES)	4	8	8	16	8
DEGREE OF INTERLEAVING	0	2	2-4	2-4	8-16
BASIC MACHINE CYCLE	500ns	200ns	200ns	80ns	54ns

Table 3

GIBSON MIX III

COMPARE	3.8
CONDITIONAL BRANCH	16.6
FIXED-POINT ADD/SUB	6.1
FIXED-POINT DIVIDE	0.2
FIXED-POINT MULTIPLY	0.6
FLOATING-POINT ADD/SUB	6.9
FLOATING-POINT DIVIDE	1.5
FLOATING-POINT MULTIPLY	3.8
INDEXING	18.0
LOAD/STORE REGISTER	31.2
LOGICAL OPERATIONS	1.6
UNCONDITIONAL BRANCH	5.3
SHIFT	4.4
	<hr/>
	100.0

Table 4

INSTRUCTIONS FREQUENCIES

CLASS OF INSTRUCTIONS

<u>Sample Job Segment</u>	<u>BR</u>	<u>C-L-S</u>	<u>L-S</u>	<u>M-C-E</u>	<u>C-IO</u>	<u>FxPt</u>	<u>F1Pt- Long</u>	<u>F1Pt- Short</u>
6. Matrix Inversion	11.9	3.4	47.0	0.7	0.1	22.5	13.6	0.8
10. Integral Evaluation	11.8	3.5	57.8	1.7	0.0	2.2	17.6	5.4
11. Curve Fitting	11.9	16.2	45.7	1.5	0.0	15.4	0.5	8.9
12. Program Compiling	28.6%	16.0%	44.0%	4.4%	0.6%	6.4%	0.0%	0.0%

BR = branches

L-S = loads/stores

M-C-E = move, compare, edit (characters)

C-L-S = compares, logical instructions, shifts.

Table 5

FORTRAN COMPILE-TIME COMPARISON

	PROGRAM 1	PROGRAM 2
FORTRAN H OPT=2	155	74
FORTRAN H OPT=0	84	58
FORTRAN G	73	32
FORTRAN E	72	40
WATFOR	9	4

COMPILATION TIMES IN SECONDS ON IDENTICAL CONFIGURATION.

Table 6

COMPARISON OF METHODS

	<u>CPU POWER</u>	<u>STORAGE CAPACITY</u>	<u>I/O POWER</u>	<u>OPERATING SYSTEM</u>	<u>COMPILER EFFICIENCY</u>	<u>WORKLOAD PROFILE</u>
ADD TIME	?					
CYCLE TIME	?					
GIBSON MIX	?					
COMPILE TIME				?	?	
CODE ANALYSIS					X	
BENCHMARKING	X	X	X	X	X	X

Table 7

SYSTEM/360 MODEL 85 DESIGN STUDY

TIME RESULTS FROM ONE TAPE

Number of Instructions	236,288
Number of Start I/Os	188
Number of Fetches	361,000
Number of Stores	40,957

		<u># CYCLES</u>	<u># BLOCKS BROUGHT IN</u>	<u># REMAPS</u>
Configuration with Main Store only.	4 way Interleave	1,091,309		
	Single 80ns	93,387		
Sector Oriented without Fetch Anticipation.	8K, 8 Sectors, 64b/bl, 4 way	1,413,654	27,836	7110
	16K, 16 " " " "	1,025,972	4,556	952
	32K, 32 " " " "	985,353	2,127	257
Sector Oriented with Fetch Anticipation.	8K, 8 Sectors 16b/bl, 4 way	1,391,617	82,250	7110
	16K, 16 " " " "	1,026,010	14,689	952

Table 8

PERFORMANCE EVALUATION

- A SHORT BIBLIOGRAPHY -

1. Gosden, J. A. and Sisson, R. L.
Standardized Comparisons of Computer Performance.
Information Processing (IFIP 62) pp. 57 - 61.
2. Raichelson, E. and Collins, G.
A Method for Comparing the Internal Operating Speeds of Computers.
Communications of the ACM. Vol. 7/No. 5, May 1964 pp. 309 - 310.
3. White, P.
Relative Effects of Central Processor and Input-Output Speeds upon
Throughput of the Large Computer.
Communications of the ACM. Vol. 7/No. 12, December 1964 pp. 711 - 714.
4. Arbuckle, R. A.
Computer Analysis and Thruput Evaluation.
Computers and Automation, January 1966 pp. 12 - 19.
5. Solomon, M. B.
Economics of Scale and the IBM System/360.
Communications of the ACM. Vol. 9/No. 6, June 1966 pp. 435 - 440.
6. Calingaert, P.
System Performance Evaluation: Survey and Appraisal.
Communications of the ACM. Vol. 10/No. 1, January 1967.
7. Kerry, D. W.
Choosing Computers for the Post Office.
Computer Bulletin, March 1967 pp. 12 - 16.
8. Katz, J. H.
An Experimental Model of System/360.
Communications of the ACM. Vol. 10/No. 11, November 1967 pp. 694 - 702.
9. Ihrer, F. C.
SCERT - Systems and Computers Evaluation and Review Technique.
Presentation at GUIDE 26, June 13th 1968, 17p.
10. Sackman, H., et al.
Exploratory Experimental Studies Comparing Online and Offline Programming
Performance.
Communications of the ACM. Vol. 11/No. 1, January 1968 pp. 3 - 11.
11. Smith, J. M.
A Review and Comparison of Certain Methods of Computer Performance
Evaluation.
Computer Bulletin. Vol. 12/No. 1, May 1968 pp. 13 - 18.

Performance Evaluation - Bibliography (continued)

12. Conti, C. J., et al.
Structural Aspects of the System/360 Model 85.
IBM Systems Journal. Vol. 7/No. 1, 1968 pp. 9 - 13.
13. Ashley, D. W.
A Methodology for Large Systems Performance Prediction.
IBM Systems Development Division report - 1968, 45p.
14. Gold, M. M.
Time-Sharing and Batch-Processing: An Experimental Comparison of their
Values in a Problem-Solving Situation.
Communications of the ACM. Vol. 12/No. 5, May 1969 pp. 249 - 259.