Multiprogramming—Promise, performance and prospect

by THOMAS B. STEEL, JR.

System Development Corporation
Santa Monica, California

INTRODUCTION

"Multiprogramming" is the label given to the concept of a dynamic sharing of the resources of a given computer system among two or more programs. An operating multiprogramming system presents to external observers the appearance of effecting the concurrent execution of several object programs. There may or may not be truly simultaneous operation of more than one program, but it will be the case that a second program begins execution before the first program has run to completion. Simple sharing of storage among several programs in a systematic way to facilitate serial execution is insufficient to qualify an operating system as incorporating multiprogramming. There must be an oscillation of control among the several programs for multiprogramming to come into play.

In order to make a rational analysis of the relation between the claims for multiprogramming and the actual performance of multiprogramming systems, it is essential to have a clear understanding of the motivation for attempts to develop such systems. It is unexceptionable that the primary justification for multiprogramming systems is the economic advantage that accrues from their use. Many areas on the frontier of information processing science, such as artificial intelligence and mechanical translation, draw their research and development energies from the desire to augment the functional capabilities of computing systems; others, such as programming language development, are pursued in the hope of improving the ability of humans to cope with the computer. Multiprogramming is intended to improve the performance of a machine qua machine. A multiprogramming system provides no capability that could not be obtained in its absence at a price. It must be kept in mind, however, that there are certain everyday activities that, while they could be accomplished in the absence of multiprogramming, could not be afforded with contemporary hardware. A case in point is giving every graduate student his own computer.

To explicate the essentially economic nature of multiprogramming it is necessary to examine the usual justifications for its use. Three arguments are advanced: (1) improved throughput by maximizing utilization of machine components, (2) time-sharing, with all it implies, and (3) real time response.

Improved throughput is currently the most prevalent reason for employing multiprogramming and is likely to remain so for the foreseeable future, time-sharing advocates notwithstanding. In this context, multiprogramming requires some multiprocessing capability in the hardware. If it is not possible for two or more explicitly programmable functions to occur in parallel, nothing can be gained from multiprogramming in the way of improved throughput and the attendant overhead is merely expensive waste motion. For this reason, multiprogramming did not arrive on the computing scene until the advent of elementary multiprocessing.

If coupled systems and satellite computers are disregarded as irrelevant in the present context, the only pertinent parallelism in generally used computers has been, and continues to be, overlap of computing and input-output. Typically, the computing time required to service an input or output device is far less than the time required for that device to perform its function. Thus, unless another part of the program initiating the input-output operation, or another program altogether, can employ the computing circuits of the machine while the input or output request is being processed, time is lost and throughput is diminished. Most multiprogramming systems are designed explicitly to overcome this loss. Improved throughput means

*The satellites and the drivers of coupled systems should be regarded as external, asynchronous signal sources and their relationship to multiprogramming is, therefore, to be found in the real time context.
more computing per unit time which means more computing per dollar. Here the motivation for multiprogramming is palpably economic.

The relationship between multiprogramming and the technology of on line, multi-access systems, unfortunately almost universally mislabelled “time-sharing,” must be carefully delineated in order to absolve multiprogramming from unmerited responsibility for the many difficulties that have plagued on-line systems. An on-line system involves multiprogramming in a special way only in the event that it is also a multi-access system. The purpose of this multiprogramming by rapid cycling through many users is to provide an economic impedance match between slow men and a fast computer. If, as may some day happen, machines were inexpensive enough it would be reasonable to provide every user with his own computer, a tour de force eliminating the need for “time-sharing.” Admittedly, this observation ignores certain significant elements of multi-access systems such as mutually interacting users and shared files, but it does illustrate the primacy of the economic motivation for multiprogramming in this context.

Real time systems pose something of a problem for this discussion as it is not clearly appropriate to view, say, an airline reservation system in the performance of its principal duty as a multiprogrammed system. Terminal servicing of this sort under control of a master program is really more akin to the treatment of conventional programs in machines with arithmetic overflow and zero divide interrupts. There is a hint of multiprogramming in the situation but it is extremely primitive. The full employment of multiprogramming in real time systems occurs when it is found desirable to occupy machine resources that would otherwise be idle while waiting for an external signal. Here the background task filling the idle time could be performed on another computer at additional cost. Again, the motivation is primarily economic, although dynamic access to a changing data base by the background program may be a factor.

Having evoked maximization of the cost effectiveness of computer resources as the dominant excuse for multiprogramming systems, it follows that instances of these systems must be judged primarily in economic terms. Care must be taken, however, to examine all aspects of the cost equation, for capturing idle machine time at the expense of programmer and operator frustration may well be a poor trade. It will be seen below that this is a nonempty caveat.

Promise

The first explicit mention of multiprogramming in the general literature was, as noted above, in the context of overlapping computing with input-output. Here, as elsewhere initially, there was no hint of a multiprogramming system, merely the suggestion of employing the technique in a single program. These early claims were modest and generally remained so in the responsible literature. As might be expected, performance claims got a little out of hand in the sales brochures of equipment manufacturers, a recurrent theme that will generally be disregarded in this account.

In the case of certain early military applications of computer systems for command and control, such as SAGE, it is at least arguable, in hindsight, that the rudiments of what would now be called a multiprogramming system were in evidence. Since for reasons of military security the details of such systems were not generally available, their development exerted little direct influence on the future course of multiprogramming. The indirect impact of these military activities was considerable, however, as a technical expertise unavailable to the less affluent civilian sector was acquired by the personnel engaged in the development of these big, complex and advanced systems. With the passage of time this expertise percolated to all corners of the industry. It is not within the scope of this paper to discuss the hopes and realities of these military systems.

Probably the first, and certainly the most ambitious, early attempt to create a multiprogramming capability in an operating system context where the object programs were expected to be totally independent was the SHARE 709 System (SOS). SOS was designed for the IBM 709, a machine with a multiple channel, asynchronous input-output capability. Among the design objectives was the ability to perform, in parallel, input for job \( N + 1 \), compute for job \( N \), and output for job \( N - 1 \). The conceptualization and design of this capability to the level of detailed flow diagrams was complete by the Fall of 1957 and the prospect of such facilities was widely accepted soon thereafter.

The next major step in the advocacy of multiprogramming systems was the explicit recognition by Strachey of the hierarchical nature of immediacy on computer time demands. The key idea in this work is not, as is usually claimed, the invention of the multi-access concept; it is the introduction of the “director,” a program element now usually referred to as a “scheduler.” In the example detailed by Strachey there was only a single on-line user envisioned, but given the concept of an on-line user with priority over all off-line users, and the director, only a modicum of imagination is required to arrive at the rest of the multi-access, on-line system concept. Subtract the on-line
user, however, and what remains is a reasonable prescription for a modern batch multiprogramming system. The real advance in this approach over the ideas inherent in the SOS structure was freeing the system from the constraint that object programs must pass through the system in a lock step order that is dictated by the initial input stream sequence.

By the end of 1961 a number, far more than cited here, of intuitive claims and theoretical analyses of multiprogramming behavior had appeared. The consensus of these studies was that effective multiprogramming was feasible, valuable and imminent. The meaning of "effective" varied, of course, from author to author. Two measures of multiprogramming effectiveness have generally been given: (1) the amount of time devoted to the overhead activity of keeping the system operating, and (2) the improvement of throughput over that encountered in a strict batch system. Both are normally quoted as percentages, and neither is really satisfactory as a measure of multiprogramming effectiveness.

The first measure indicates the minimum distance from perfection—the unrealizable state where all of the machine is busy on useful work all of the time—that the system could attain with an ideal job mix. It gives an absolute measure of the best case. The second measure indicates the relative improvement in performance but provides no indication of how this relates to the ideal. Even taken together the two measures do not close the gap. Something else is needed, perhaps a set of standard job mixes, but this paper must stand on the available data.

These early claims varied but generally suggested overhead figures of between one and fifteen percent and throughput improvements of up to several hundred percent. The low figures for overhead were quoted by manufacturers, such as Honeywell for its Model 800, who planned to employ hardware for much of the overhead activity. The extra hardware costs money, however, and this cost is normally not factored into the equation.

By 1962, and continuing to date, the scientific literature began to carry reports of actual experience with multiprogramming systems. Most of these reports were made by representatives of hardware manufacturers and, therefore, exhibited a natural tendency to minimize difficulties. Since this time, the performance claims for multiprogramming systems have remained more or less static in the responsible literature. Two revealing changes have occurred: (1) the word "multiprogramming" no longer contains a hyphen, providing at least linguistic legitimacy to the concept, and (2) the availability dates for most multiprogramming systems have slipped, often considerably, but happily at somewhat less than real time.

At the present time most large, commercially available computing systems are designed with the expectation that they will be multiprogrammed in many environments. Generally they are accompanied by a manufacturer supplied, batch multiprogramming system. Some are vehicles for on-line, multi-access systems that are now in being. Since these things exist and can be evaluated by the customer, current claims tend to be rather realistic and representative of the actual situation, at least with respect to performance. Ease of use is another matter altogether.

Performance

Having reviewed in deliberately general terms the promises for multiprogramming, it is now necessary to review in equally general terms what has actually been accomplished in providing multiprogramming capability. As noted above, the early claims were modest and made in the context of multiprogramming within the confines of a single object program. Even here the claims went a bit beyond reality. While substantial gains in program performance were obtained through overlapping computing with input-output operations, the results were less gratifying than it seemed they ought to be. Two reasons for this can be isolated.

In the first place, insufficient information was provided by the hardware for programmers to make completely effective use of the actual capabilities of the machine. The extent of the requisite interaction between hardware and program for efficient handling of asynchronous interrupts was long unrecognized, despite some careful studies on the subject. Indeed, this subject is not perfectly understood today, but at least there seems no doubt that, at long last, machine designers are aware of the problem. Attention to its resolution cannot fail to improve the performance of future systems.

The second difficulty with early multiprogramming efforts was more subtle and went unappreciated until attempts were made to instrument computing systems to determine, among other things, just how much overlap of computing and input-output actually went on. The depressing nature of the results could be traced to the fact that the average programmer simply was not good enough to make full use of the multi-channel hardware capability, even where the problem cried out for its use. It was the recognition of this situation that paved the way for the design of the first multiprogramming systems.

SOS was a failure if measured by its impact on the community. It is the author's contention—exhibiting
a bias justified in one of the designers—that the failure was in implementation, not in design. To put the matter bluntly, IBM blew it! The system that finally came out, to be distributed and maintained by the vendor, was a pale ghost of the original conception (in many aspects, not just in the multiprogramming features) that eventually evolved into IBSYS, a tolerably good batch processing system, but hardly a multiprogramming system. The fact that reasonable approximations of the original were used by the RAND Corporation, the Applied Physics Laboratory and others indicts the IBM effort. The multiprogramming aspect of this situation must be carefully weighed. It was possible to get approximately the same amount of overlap between computing and input-output in both IBSYS and SOS, but it required much more programmer effort in IBSYS. This difference is significant in view of the difficulty programmers have with this problem.

The multiprogramming aspects of large, real time systems and on-line, multi-access systems are somewhat more severe than those found in batch multiprogramming systems, due to the added pressure of vital time requirements on the scheduler not usually found in batch situations. Nevertheless, the literature reveals that the serious problems in these more complex systems really lie elsewhere in such areas as command language, paging, multi-access files and lack of reentrancy in generated code.16-17 It is a slight oversimplification, but not amiss in principle, to assert that the current performance of multiprogramming in real time and multi-access systems has about the same relationship to the earlier promises as can be shown to apply in batch systems.

The literature now abounds in reports on the performance of various multiprocessing systems,18-22 but with some notable inadequacies; e.g., for OS/360. The trouble with these reports is that they don't say very much concrete. The lack of carefully defined measures hurts the analyst and the lack of hard data hurts him even more. In view of this situation the author has forborne from constructing a table of performances of the specific existing systems. It would be invidious to attempt an explicit comparison from the available data, and no individual has had sufficient experience with several systems to permit the gathering of even intuitive judgments. What evidence there is suggests that all of the major systems share roughly the same strengths and weaknesses. Thus, it makes sense to compare a kind of generic, current multiprogramming system against the industry's broad claims for what it, collectively, expected to produce.

Viewed in these terms, the situation is rather better than one might expect. While it is only in special situations that throughput improvement has attained the several hundred percent initially claimed, the average figures of between thirty and seventy percent are but a binary order of magnitude below the claims. More or less the same factor of two appears in the overhead figures; ten to thirty percent in the real situation as against half that in the projections. The serious problem has been in delivery dates, a not uncommon element of software development. As has been observed, much of this is due to inadequacies in the hardware. Good multiprogramming systems simply were not in the cards until the current class of machines were available.

The glaring failure of current multiprogramming technology is the complications it has introduced for the programmer and operator. The current Job Control Languages (JCL) required to specify what the system is to do are, by and large, disasters. It takes far too much of a programmer's time to construct the appropriate JCL statements, and an even larger amount of time to debug them, not to mention the effect on morale of aborted runs deriving from trivial JCL errors. The implications of JCL in the machine room are even worse. The operators must be a good deal smarter (and, therefore, necessarily, paid more) than has been standard in the past. Computer center managers are faced with the realization that poor operations will destroy and overwhelm any gain in throughput with no trouble at all.

Prospect

In considering where multiprogramming may be expected to go from here, it is worth noting the reason why this technology fares considerably better than most when measured against the early claims of its proponents. In multiprogramming there is a well defined, finite upper limit for improvement in capability. The most one could ever claim was to use all of the machine all of the time. This is hardly true in other cases; witness artificial intelligence where the more flamboyant claims defy the imagination. As multiprogramming technology approaches its natural limit, it clearly ceases to make sense to expend much effort trying for the last fraction. Satisfaction is obtained by near optimum performance at acceptable cost.

Some performance improvement can be obtained through judicious use of the various empirical44 and theoretical45-57 studies of multiprogramming, and through attention to the problem of hardware-software interaction. It is likely that use of a firmware approach to multiprogramming supervisory by placing them in read-fast, write-slow storage units will provide the last big reduction in overhead cost. Improvements beyond a factor of two or three will probably cost more than they are worth. Of course, the introduc-
tion of multiple, interacting arithmetic, logic and control units may start a whole new ball game.

The real improvements in multiprogramming systems must come in the area of making them easier to program and to operate. As indicated above, much is left to be done in this area, and it shows little sign of happening so far. Programmers sweat, operators err and managers complain, but JCL marches on.

REFERENCES

1 E F Codd
   Multiprogramming

2 N Rochester
   The computer and its peripheral equipment
   Proc EJCC p 64 1955

3 R R Everett C A Zraket H D Bennington
   SAGE—a data processing system for air defense
   Proc EJCC p 146 1957

4 O Mock C J Swift
   The SHARE 700 System: Programmed input-output buffering
   JACM 6 2 p 145 1959

5 C Strachey
   Time sharing in large fast computers
   Proc ICIP p 336 1959

6 E F Codd
   Multiprogram scheduling
   CACM 3 6 p 347 1960

7 B L Ryle
   Multiple program data processing
   CACM 4 2 p 99 1961

8 T Kilburn D J Howarth R B Payne
   F H Sumner
   The Manchester University Atlas operating system Part I: Internal Organization
   Computer J 4 p 222 1961

9 R D Smith
   Multiprogramming the RCA 601
   Preprints of 16th ACM National Meeting p 12C-1 1961

10 G Felton
    Multiprogramming on the Ferranti Orion computer system
    Proc IFIP p 570 1962

11 R Clippinger
    Multi-programming on the Honeywell 800/1800
    Proc IFIP p 571 1962

12 J L Francois

Programmation simultanee dans le cas du Gamma 60 Bull
Proc IFIP p 573 1962

13 E F Codd
   Multi-programming stretch: A report on trials
   Proc IFIP p 574 1962

14 A J Critchlow
   Generalized multiprocession and multiprogramming systems
   Proc EJCC p 107 1963

15 F P Brooks JR
   A program-controlled program interruption system
   Proc EJCC p 128 1957

16 J Schwartz
   A general purpose time-sharing system
   Proc SJCC p 207 1964

17 F J Corbató V A Vyssotsky
   Introduction and overview of the multics system
   Proc IFIP p 185 1965

18 B B Clayton E K Dorff R E Fagen
   An operating system and programming system for the 6600
   Proc SJCC v 2 p 41 1964

19 H S Bright
   A Philco multiprocessing system
   Proc SJCC v 2 p 36 1964

20 D M Daum
   Using the B5500 disk file master control program
   Proc IFIP v 2 p 362 1965

21 J W Lewis
   Multiprogramming on Leo III
   Proc IFIP v 2 p 363 1965

22 J Bouvard
   Experience with multiprogramming on the Honeywell 800/1800
   Proc IFIP v 2 p 364 1965

23 R E Fagen
   Time-sharing the Control Data 6600
   Proc IFIP v 2 p 368 1965

24 H N Cantrell A L Ellison
   Multiprogramming system performance measurement and analysis
   Proc SJCC p 213 1968

25 W Chang D J Wong
   Analysis of real time multiprogramming
   JACM 12 4 p 581 1965

26 B Peters
   Security considerations in a multi-programmed computer system
   Proc SJCC p 283 1967

27 D P Gaver JR
   Probability models for multiprogramming computer systems
   JACM 14 3 p 423 1967