MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Laboratory for Computer Science

(formerly Project MAC)

Computation Structures Group Memo 140

Computer Architecture and the Cost of Software

Jack B. Dennis

(This paper was presented at the Third Annual Computer Architecture Symposium, January 1976, and appeared in SIGARCH News, April 1976).

This research was supported by the National Science Foundation under grants DCR 74-21892 and DCR 75-04060.

July 1976

Computer Architecture and the Cost of Software

Jack B. Dennis
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
617-253-6856

The software problem is nearly always discussed in terms of software or administrative solutions, and the nature of the computer hardware is tacitly regarded as unchangeable or irrelevant. I believe such limited views of the software problem keep us from recognizing the fundamental limitations of the computer systems we must work with today. Important improvements in the methodology of developing very large programs will not happen in the absence of important changes in the hardware structure and organization of computer systems. In this paper I discuss the reasoning underlying my belief and indicate the directions of advance required in computer hardware to provide a better basis for the construction of reliable software.

Improved administrative methods for software projects and improved technical methodologies of program construction are the main avenues advocated for reducing software cost. Better management practices can have little impact on the complexity of the product of software development. Therefore, large gains in software quality will come about only through technical advances that yield simpler and more transparent program structure, provide confidence that a software product performs the intended function, and make the software design and implementation process more straightforward.

Today the most promising technical improvements in the programming process are included in a broad interpretation of that much-abused phrase "structured programming." The proponents of structured programming argue for the use of a programming methodology that allows the construction of a large program to be divided into separately specified parts such that the design, programming and verification of each part involves the fewest possible interactions with the development of other parts. This is in fact a concept of program modularity -- it should be possible to build large programs from simpler program parts (modules) that may be designed and implemented independently. However, it is now appreciated that program modularity is a far deeper concept than was understood by the early advocates of modular programming. With the development of interest in the formal verification of program correctness we now have a sophisticated criterion for a successful methodology of modular programming -- if programs are built of parts that can be developed truly independently, then proofs of program correctness will grow only linearly with the size of the program.

Achieving the goal of structured programming calls for significant changes in the characteristics of programming languages. It has been observed that our popular programming languages have features in conflict with the principles of modular programming. Moreover, the form of program module supported by a language, and its treatment of storage have a profound influence on the ability of a programmer to separate design decisions or prove that program modules satisfy their specifications. In particular, elimination of: global variables; unrestricted transfers of control; and blatant use of side effects have all been recognized as important steps toward better languages for structured programming.

Work on the development of structured programming methodology has concentrated in two areas: programs expressed in a sequential procedural language; and the cooperation of multiple sequential processes. The former work has led to new efforts at programming language design, and has established a number of principles through which the ideals of structured programming may be approached in the use of conventional high-level languages. For many applications of small to moderate scale, such disciplined use of existing programming tools can be a very effective methodology.

The work on structuring cooperating sequential processes has been most influential in contributing to improved structure of operating systems, but the contribution has been mostly limited to the structuring of control, with little attention to the structuring of data in multi-process computation.

In the construction of large programs, two phenomena arise that disasterously interfere with the practical use of structured programming methodology. Firstly hardware imposed limits are reached, the most significant being the amount of directly addressable (main) memory. The programmer is forced to use distinct means for referencing information as it resides in main or auxiliary storage, and use of a uniform convention for passing data between program modules is no longer possible -- an immediate conflict with basic requirements for modular programming.

Secondly, large programs usually require use of computer system facilities supported by the operating system: manipulation of files; cooperative multiprocessing; access control; and communication with user terminals. The programmer must go outside his programming language for these aspects of his application and deal with all the inconsistencies in data representation, access means, storage management, etc. between the operating system and his programming language.

Two steps are required to change this situation and make the goals of structured programming achievable for large software projects. We must first eliminate the distinctions among physical memory media from the programmer's view. A uniform mechanism for accessing program modules and data structures must be built into the computer system -- a generalization of the virtual memory systems now in use.

The second step is to encompass all basic facilities for programming in a language consistent with the principles of structured programming. (A major unresolved area is the incorporation of facilities for cooperative multiprocessing into a procedural language.) Once this is done, the language may be taken as the <u>specification</u> for a class of computer systems to be realized through appropriate combination of hardware, firmware and software.

There is today abundant evidence of the importance of parallel computation. Many problems of great importance require highly parallel computation to have feasible solutions -- simulation of the general circulation model of the atmosphere, for example. Also, multiple processors in dedicated configurations have been found to outperform large single-processor systems in many applications. A principal reason is the ability of such configurations to achieve better utilization of relatively expensive main memory.

A powerful but little appreciated argument for parallelism stems from the organization of computer memory systems as hierarchies of physical devices spanning a wide range of speed and capacity -- an essential characteristic for economically running very large programs or manipulating large data bases. A sequential processing unit cannot be kept busy when delays result from references to information held in slower levels of the memory system. One can design memory systems to handle large numbers of access requests concurrently and with high efficiency, but this is of little interest unless the processing components are capable of generating large numbers of concurrent requests. This is possible only if the processing components are organized and the program representation chosen so many instructions are concurrently available for processing.

Past efforts to achieve highly parallel computation (the associative, vector, and array processors) have attempted to exploit local parallelism in programs and have paid little heed to the issue of programmability. Even the concept of procedure, which is fundamental to the modular construction of programs, is rendered inapplicable if these machines are programmed to yield significantly greater performance than conventional processors.

Rather, architectural concepts are needed that support highly parallel computation and not only continue to provide support for programming constructs of confirmed merit, but are entirely consistent with the demands of structured programming.

That the architecture of present-day conventional sequential computers is ill-suited to support a good methodology of program construction should not be surprising. When the concept of the stored program computer was conceived three decades back, the overwhelming need was for the simplest general-purpose program execution mechanism that could carry out straightforward numerical computations. Such concepts as modular program structure played little if any role.

Nowadays, requirements have drastically changed: The concepts of procedures and recursion have been accepted; the importance of chosing data types to match abstract concepts of the application has been recognized. Modularity in program structure is appreciated as a goal with its implications with respect to data structuring, uniformity of reference, and storage management. And cooperative multiprocessing has become an essential part of the programming art. We should be surprised that the Von Neumann concept of stored program computer has survived this revolution in the conceptual basis of programming.

In summary, I believe two major advances in computer architecture are essential to reap the potential benefits of structured programming for very large programs. Architectures must support a uniform and device-independent mechanism for accessing all on-line information -- a generalized virtual memory; and architectures must be so conceived that many instructions are available for concurrent processing. Moreover, architectures must be developed on the basis of sound language design principles and must meet all requirements for the modular construction of large programs.