DEVELOPMENTS IN SUPERCOMPUTER LANGUAGES

R.H. Perrott
Department of Computer Science, The Queen's University, Belfast BT7 1NN, Northern Ireland

ABSTRACT

Concurrent languages, such as Ada and Pascal Plus, have been designed and implemented for configurations which consist of a number of independent and concurrently operating processors. The development of languages for programming array and vector processors has proceeded independently and produced variants of Fortran for representing this type of parallel processing. However, the latest hardware configurations contain both types of parallelism, for example, the Cray X-MP contains several vector processors which are capable of acting independently and in parallel. It is appropriate with the introduction of these new configurations to consider the design of a language capable of handling both types of parallelism. Such a proposal is considered in this article.

1. INTRODUCTION

The developments in hardware technology which have enabled the production of supercomputers are well documented and well utilized in existing systems. However, in the case of the software for these supercomputers the developments have not been so radical or widely accepted among the user community. One of the reasons for this reluctance to accept new software techniques is that these machines have been and are widely used by the engineering and scientific communities. Over the last thirty years, these communities have been well served by a useful and enduring tool in the Fortran language.

This restriction to a Fortran base has, in turn, affected the languages which are proposed or implemented for these new architectures. Originally these machines consisted of an array or vector processor, but now configurations are being produced which consist of array or vector processors as part of a multiprocessor system. As a result languages are required which capitalize on the array or vector processing facilities as well as the other components of a multiprocessor system.
At The Queen's University we have been carrying out research into the design and implementation of parallel languages which are suitable for both array and vector processor configurations and multiprocessor configurations. The array and vector processor work originated in a project carried out at the NASA Ames Research Center in California, to design a parallel language for the successor to the Illiac IV. This language is known as Actus (Perrott, 1976).

Since that time several language experiments have been carried out based on the philosophy expounded in the Actus language. This has led to a gradual evolution and better understanding of language features suitable for array and vector processing. Other colleagues have been investigating multiprocessor programming languages and have produced the language Pascal Plus (Welsh and Bustard, 1979) based on the monitor and conditions for process synchronization. As a result of these experiences a recent project has considered the synthesis of these two languages to produce a language for programming a configuration which can consist of array or vector processors and other independent processors all executing in parallel (Orr, 1986).

2. MULTIPROCESSING/DISTRIBUTED PROGRAMMING

The term process or more recently task is used to describe a sequence of program instructions that can be performed in parallel with other program instructions. A program can therefore be represented as a number of processes which can be executing concurrently. The point at which a processor is withdrawn from one process and given to another is dependent on the progress of the processes and the algorithm used to assign the available processor(s). The net effect is that processes are capable of interacting in a time-dependent manner.

Thus in a concurrent programming environment a programmer requires not only program and data structures similar to those required in a sequential programming environment but also tools to control the interaction of the processes - processors which are proceeding at fixed but unknown rates.

The situations in which processes interact can be divided into two categories. The first situation occurs whenever processes wish to update a shared variable or a resource at the same time. For example, when several processes wish to use the same resource, only one process must succeed in gaining access to the resource at any time. Once a process has obtained the resource it must be able to use the resource without interference from the other competing processes. This is described as mutual exclusion.

The second situation occurs when processes are co-operating, they must be correctly synchronized with respect to each other's activities. For example, when one process requires a result not yet produced by another process. The first process must be able to wait on the second process and the second process must take the responsibility of-resuming the first process when it arrives with the result. The processes are communicating or scheduling one another and are aware of each other's ex-