SOME PRIMITIVES FOR
THE PORTABLE PROGRAMMING OF ARRAY
AND VECTOR PROCESSORS

PETER J. L. WALLIS

Abstract.
A high-level language for array and vector processors is analyzed by the methods of
dernotational semantics. The analysis leads to the identification of a set of primitives
suitable for the portable programming of array and vector processors. Discussion of the
primitives includes consideration of the efficiency with which they may be implemented on
different machines, and their possible application to portable programming, the design of
intermediate languages, and the design of future array and vector processors.

Key Words and Phrases: Actus, array processors, denotational semantics, parallelism,
portability, vector processors.


1. Introduction.
Computers for really fast, large-scale numerical computation must necessarily
have an architecture that allows some kind of parallelism within the basic
hardware. There are many ways of designing such computers; attention in the
present work is restricted to the “single instruction stream multiple data stream”
(SIMD) machines. Here the same operation is applied repeatedly to different data
with extensive overlapping of instruction execution (vector processors), or the
same instruction is broadcast to different data sets (array processors). Such
machines are particularly suitable for large-scale numerical computation and a
number of different SIMD machines have been designed. These include both array
processors (e.g. Illiac IV, Phoenix, ICL DAP) featuring a number of identical
arithmetic units and vector processors (e.g. Cray-1, CDC STAR) with pipelined
functional units.

As the term “SIMD machine” implies, the programmer of such a machine
necessarily thinks of a single instruction stream; this stream can be executed in a
lockstep parallel fashion over a number of different data sets. Programming such
a machine is much like programming a conventional machine with two additional
complicating factors: the programmer must control both the “extent of
parallelism” (i.e. the number of data streams participating at each point in the
computation) and the routing of data between the data streams. The present work

Received August 12, 1981.
is mainly concerned with the handling of the extent of parallelism; this has far more effect on the overall structure of programs than the programmer's treatment of data routing, considered here in less detail.

The programming of SIMD machines has traditionally been highly machine-dependent even when a high-level language is used, since the high-level languages first developed for them included many machine-dependent features [1]. Clearly this machine-dependence must be controlled in some way if portable programming for SIMD machines is to become feasible. The way to control the machine dependencies is to encourage the programmer of an SIMD machine to work in terms of abstractions or primitives that can be efficiently matched to the different underlying architectures. Such matching might be accomplished by a compiler for a common high-level language implemented for several SIMD machines; alternatively the primitives themselves might be encoded for different machines as library subroutines with machine-dependent bodies but with machine-independent specifications. Whatever method is used for the portable programming of SIMD machines, its ultimate success will depend on the identification of a suitable set of primitives. This paper is concerned with the identification and evaluation of such primitives, emphasising those connected with the extent of parallelism.

A useful set of machine-independent primitives for the portable programming of SIMD machines would be both efficient to implement and convenient for programmers to use. The present search for such primitives takes Actus [2, 3] as its starting-point. Actus is a high-level language that can be efficiently implemented on both array and vector processors. Moreover its design was preceded by a formal survey [4] of the current and previous users of the Illiac IV array processor that appears representative of its user population. Thus the constructs of Actus appear to meet the criteria of efficient implementation and of acceptability to programmers. This suggests that a suitable set of primitives for the portable programming of SIMD machines might be identified by a semantic analysis of Actus.

Actus is a fairly large programming language, so a full treatment of its semantics would involve large amounts of detail of no interest in the present context. After a brief informal introduction to the relevant features of Actus, a design and denotational semantics are developed for SIMDL, a severely curtailed programming language that nonetheless requires that the extent of parallelism be handled as in Actus. The auxiliary valuations of the denotational semantics for SIMDL are then used to identify a corresponding set of runtime parallel primitives for the implementation of Actus. Finally these primitives are discussed in some detail.

2. Brief introduction to Actus.

This section features a brief introduction to Actus sufficient to motivate the