THE SEMANTICS AND COMPLEXITY OF PARALLEL PROGRAMS FOR VECTOR COMPUTATIONS.
PART I: A CASE STUDY USING ADA*

EDWARD K. BLUM

Mathematics Department, University of Southern California, Los Angeles, CA 90089, USA

Dedicated to Peter Naur on the occasion of his 60th birthday

Abstract.

Recent research in parallel numerical computation has tended to focus on the algorithmic level. Less attention has been given to the programming level where algorithm is matched, at some extent, to computer architecture. This two-part paper presents a three-level approach to parallel programming which distinguishes between mathematical algorithm, program and computer architecture. In part I, we motivate our approach by a case study using the Ada language. In part II, a mathematical concept of parallel algorithm is introduced in terms of partial orders. This serves as the basis of a theory of parallel computation which makes possible a precise semantics and a precise criterion of complexity of parallel programs. It also suggests some notation for specifying parallel numerical algorithms. To illustrate the ideas presented in part II, we concentrate here on parallel numerical computations which have vector spaces as their central data type and which are intended to be executed on a multi-processor system. The Ada language, with its task constructs, allows one to program computer algorithms to be executed on multi-processor systems, rather than on “vector (pipelined) architectures”. To provide a concrete example of the general problem of programming parallel numerical algorithms for multi-processor computers, we do a case study of how Ada can be used to program the solution of a system of linear equations on such computers. The case study includes an analysis of complexity which addresses the cost of data movement and process control/synchronization as well as the usual arithmetic complexity.

CR categories: F.1.2, F.1.3, F.3.2.

1. Introduction.

Many computational systems involve operations on vector quantities which can be executed in a concurrent mode. We consider the specification/design of such systems. In our view, specification takes place on three levels: (1) the mathematical algorithm level; (2) the program level; and (3) the computer architecture level. This is an idealization of the practical situation where it is not always possible to separate the three levels of specification, either logically or in order of performance.

* This research was partially supported by NSF Grants DCR-8406290 & CCR-8712192.

We assume that specification starts on level 1 with some mathematical equations expressing the system input-output function. For example, the solution of a system of linear equations is specified by the equations $Ax = b$ and $x = A^{-1}b$, where $A$ is a nonsingular $n \times n$ matrix and $b$ is an $n$-dimensional vector. The equations involve basic operations on basic sets in one or more data types; e.g. the operations in numerical computations are the arithmetic operations on integers and reals and the boolean operations on \{0, 1\}. However, as in our example, they may include operations on vectors and matrices. Such a purely mathematical specification may be regarded as taking place on the highest level, which we shall call level 1a. On level 1b, the mathematical equations are transformed into equations which specify a mathematical algorithm. In our example, these are the equations defining the familiar Gauss elimination algorithm. (See section 3.) The concept of parallel mathematical algorithm needs to be made precise and we do this in part II in a fairly general context. Here we use an intuitive notion of parallel algorithm. In numerical algorithms involving vectors, a natural parallelism is often evident in the operations on components. However, there are usually several different modes of parallel execution possible. To achieve the most efficient usage of the architecture available on level 3, it may be necessary to specify explicitly which parallel mode is to be selected. As yet, there is no standard notation for this kind of specification on level 1 [2]. We shall discuss this issue in the next two sections.

Assuming that a specification of a parallel mathematical algorithm is at hand, the next stage is the programming of the algorithm for a particular class of architectures. We view a program as the specification of a computer algorithm. A computer algorithm is the result of a transformation of a mathematical algorithm. For most existing programming languages, this transformation introduces two kinds of "non-mathematical" operations: those which perform data movement and those which control the computational flow. Concurrent execution of various parts of a program on a multi-processor architecture may require synchronization control as well. We shall consider this form of parallelism and illustrate it by a case study employing the Ada language with its tasking construct. There is no standard parallel programming language at present [10], [11], [12], [13], and many divergent opinions on how to construct one [2]. The Ada study allows us to address important aspects of parallel programming in a specific way.

The Ada task construct is reviewed briefly in section 4. It suggests a multi-processor shared-memory architecture on level 3. However, other architectures can be admitted. A convenient summary of architectures and current computers as compiled by Hwang [1] is reproduced, with his permission, in Figure 1, with some additions.

The main objective in parallel numerical computation is to achieve speedup over sequential execution within reasonable efficiency constraints. In data-processing computations, such as the reader-writer problem, concurrency is