ABSTRACT

Of the many proposals for parallel computer architectures, dataflow systems offer perhaps the greatest potential parallelism at the hardware level. However, it is unreasonable to expect users to exploit this parallelism at such a low level, and it is necessary to provide high level programming facilities which allow the user the abstract freedom of expression he requires, whilst permitting efficient use of the hardware. It has been noticed that single-assignment languages are particularly well-suited to dataflow machines, yet exhibit most of the familiar, useful features of conventional high level languages. This paper considers some aspects of code generation from the single-assignment language Lapse for the Manchester prototype dataflow computer. The syntax of Lapse, which resembles Pascal, is introduced, and code generation templates are presented. Some possible optimisations of flowgraph code are discussed, particularly in the implementation of arrays.

1 INTRODUCTION

A large volume of interest has been shown recently in parallel computing systems, and one type of parallel architecture, the dataflow machine, shows promise of providing highly parallel hardware available for the programmer's general use. Several dataflow systems have been designed [2,8,9,12,16,19], mostly with a view to providing high computing speed via easily accessible parallel hardware at the machine-code level. However, there is a general awareness that the average user does not require parallelism at quite this low a level. He would rather find that parallel execution is automatically invoked after he has presented his programs in a suitable high level language. Consequently, a current goal of many dataflow projects is to develop appropriate high level language facilities. As with other types of computer, the achievement of satisfactory results involves a compromise between the expressive power of the language, and the efficiency of translation and execution of the resulting object code. An obvious route to high level dataflow programming is to de-serialise a conventional language such as Pascal or Fortran. Several optimising compilers for vector processors already use this technique (e.g. Crayl and DAP Fortran), and there are known algorithms for converting sequential code to graphical form [14]. However, even the direct conversion process is lengthy, and the more sophisticated unravelling optimisations are extremely expensive. In any case, it is hard to avoid the feeling that the act of writing the program in a serial language in
the first place has already stifled many opportunities for parallel execution. For
the user, it would thus seem better to use a language with no (or at least few)
serial connotations. Abandoning the traditions of conventional languages gives us the
chance to adopt radical alternatives. For example, we might consider using a
denotational language in which a program is a list of equations or relations defining
the problem to be solved, and containing no operational hints as to how to obtain the
solution. Many such languages have been proposed, including Lucid [4] and Prolog
[18], and some preliminary research into dataflow implementations has been
undertaken. However, problems concerning efficient execution abound, as they do for
conventional machines, and this is an open area for the moment.

The above analysis leads us to a middle-of-the-road solution to the needs of the
general purpose user of dataflow machines. We must provide a language whose semantics
are operational, but based on parallel execution via dataflow graphical code. This
will lead to easy translation from high to low level (cf. Fortran for von Neumann
machines), and will make this paper readily comprehensible to readers with an
operational background. The class of languages which is suitable for study in this
context is that of the single assignment languages (SALs), originally proposed before
dataflow was widely known [7,17], and since revamped in many forms, including in the
context of dataflow [1,3,16]. The language used to illustrate this paper is known as
Lapse, and is taken from the work of Glauert [11]. The important features of these
languages are the way in which high level structures in the language reduce simply to
equivalent machine level code, and the resultant performance of the programs. It has
been found that the reduction of high level statements can be achieved via
application of graphical templates, supported by macro expansion into object code.
Templates are important abstractions that will be used extensively in this paper.
They can show how code is produced from Lapse statements, how it performs when
executed, and how it may be optimised.

The paper commences with a brief description of the labelled flowgraph notation used
to describe the graphical machine code. The syntax of Lapse is then introduced using
short example program segments whose translation is also illustrated. Discussion of
the more complex of these examples leads to the consideration of code optimisation.

2 DATAFLOW GRAPHS AND LABELLED TOKENS

Computational dataflow graphs are directed graphs in which nodes represent functions.
Each node has a fixed number of input and output points to which incoming and
outgoing arcs, respectively, are attached. These arcs act as directed data paths
which transmit computed data values, carried by tokens, between functions. Nodes may
be primitive, in which case they represent machine level functions (i.e.
instructions), or compound, representing a higher level operation that cannot be
achieved directly at the machine level. Machine level nodes are constrained to have a
maximum of two input and two output points. Compound nodes can always be translated
into an appropriate combination of primitive nodes, for example by a process of macro