MULTIPROCESSOR SYSTEMS PROGRAMMING IN A HIGH-LEVEL DATA-FLOW LANGUAGE*

J.L. Gaudiot and L.T. Lee

Computer Research Institute
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California
(213) 743-0240

Abstract

The data-flow model of computation is an attractive methodology for multiprocessor programming for it offers the potential for unlimited parallelism detection at no programmer's expense. It is here applied to a distributed architecture based on a commercially available microprocessor (the Inmos Transputer). In this project, we have integrated the high-level data driven principles of scheduling within the Transputer architecture so as to provide high programmability of our multicomputer system. A complete programming environment which translates a complex data-flow program graph into occam has been developed and is presented in this paper. We here describe in detail the mapping from the SISAL high-level constructs into the low-level mechanisms of the Transputer (synchronization, structure representation, etc.). The partitioning issues (granularity of the graph) are presented and several solutions based upon both data-flow analysis (communication costs) and program syntax (program structure) are proposed and have been implemented in our programming environment. Finally, we present and analyze graph allocation and optimization schemes to improve the performance of the resulting occam program.

I. INTRODUCTION

The programmability of the multiprocessors of the next generation is generally recognized to be the major issue to confront designers. Several methodologies for the safe and efficient programming of parallel architectures have been proposed in order to deal with these problems. These languages include concurrent PASCAL and ADA (Andrews, 1983). Among the constructs which enable the specification of parallelism, operations may authorize the execution in parallel of two or more processes. Similarly, processes may be synchronized on data dependencies by the introduction of shared variables. In order to ensure the proper ordering of the various updates to a specific cell of memory, the programmer must specify critical sections for the program which essentially lock out the access of some specific cells of memory until they have been safely updated by a single process.

As technology progresses, it will be possible to integrate very large numbers of processors in the same machine. The explicit parallelism specification is therefore a complicated problem with large multiprocessor systems since the number of tasks that must be kept concurrently ac-

*This material is based upon work supported in part by the National Science Foundation under Grant No. CCR-8603772 and by the USC Faculty Research and Innovation Fund
tive becomes very large. For instance, the Cray X-MP-4 contains up to 4 processors around a shared memory system but the upcoming Cray-3 will include 16 processors. This demonstrates the need for a new approach for the programming of large multiprocessor systems.

Backus (1978) has introduced functional languages as a way to unshackle the programmer from the structure of the machine. Indeed, instead of considering instructions which can modify memory cells, the functional model of computation assumes functions which are applied to values, producing result values. This model of execution does not have recourse to architectural notions. Moreover, the executability of an instruction is decided by the availability of its operands. This can be implemented in a distributed fashion, thereby obviating the need for the central program counter of the von Neumann paradigm.

Data-flow systems (Dennis, 1974) obey functional principles of execution. They implement a low level description of the execution mechanisms which govern a functional multicomputer architecture. In addition, high-level languages such as VAL (McGraw, 1982), Id (Arvind, Gostelow, and Plouffe, 1978), LAU (Syre et al., 1977), and HDFL (Gaudiot et al., 1985) have been proposed as a high level interface.

While this approach brings a solution to many multiprocessing problems (Arvind and Iannucci, 1983), several data-flow projects exploit the parallelism inherent in data-flow programming by the introduction of complex custom-made Processing Elements. We describe here a multiprocessor architecture based on off-the-shelf components which can be programmed under the data-flow model of computation. The research presented here demonstrates the applicability of the functional mode of execution to such a problem. We have chosen for this work the high-level functional language SISAL (Streams and Iterations in a Single Assignment Language) developed at the Lawrence Livermore National Laboratory by McGraw and Skedzielewski (1984). Note that this language has also been chosen for high-level language programming of the University of Manchester data-flow machine (Gurd et al., 1985).

The purpose of this paper is to demonstrate and describe in detail a methodology for multiprocessor systems programming. The schemes demonstrated are to be used as a testbed for future research in MIMD systems. The underlying architecture to which this effort has been applied as well as the integrated programming environment are described in section 2. The low level model of execution of the Inmos Transputer (the language occam) upon which the architecture is based, is presented in section 3.1. High-level language programmability is afforded by the data-flow language SISAL (sections 3.2 and 3.3.) The partitioning methods of the SISAL compiler output, as well as the translation mechanisms into occam are demonstrated in section 4. In section 5, we show the partitioning and optimization mechanisms which have been developed. The mapping mechanisms from SISAL high-level constructs are demonstrated in section 6, along with an illustrative example. Finally, some concluding remarks are drawn in section 7.