Chapter 13

DATA FLOW PROGRAMMING LANGUAGES

13.1. The Data Flow Model

The advancement of the speed of computers has been phenomenal over the past two decades. There have been several orders of magnitude improvement, so that now processors exist with cycle times of a few nanoseconds that can execute several million instructions in a second. Nevertheless the demand for computing cycles has outstripped even these gains. Compute bound jobs such as weather prediction, simulation, or monte-carlo analysis can require execution speeds in the billions of instructions per second. Therefore the search for faster computers goes on.

But on one level this search is beginning to run into important problems. As hardware designers continue to try to make elementary components smaller, they are running up against physical limitations. For example as the size of connectors is made smaller, the resistance per unit length of a conductor scales up quadratically. This requires additional circuitry, thus eliminating much of the gain. Therefore one cannot expect to improve processor speed much more by placing components closer together. Another factor is the amount of power versus the cooling required as circuits become densely packed. This ratio has been growing very fast to the extent that the newest super-
computers will have to be super-cooled in order to get them to achieve super speeds. For example circuits built using Josephson junction technology must be cooled to 4 degrees above absolute zero by immersion in liquid helium. In short, the approach of focusing on the central processing unit and trying to speed up its circuitry does not appear capable of sustaining the orders of improvement that have been achieved in the past.

But this is not the only means that has been investigated for achieving faster computation speeds. Other solutions have included the creation of special purpose processors such as array processors, vector machines, and multiprocessors. An array processor is a machine which uses pipelining to speed up the application of a single instruction on an array of data. This paradigm of computing is essential in many important algorithms such as the fast Fourier transform. Adding an array processor to a medium scale computer can have the effect of turning it into a "number-cruncher" for certain kinds of problems. This is a very cost-effective approach to improved computing. But one problem with this approach is that one must program in a baroque assembly language in order to get the array processor to do new tasks. For essential tasks such as the FFT, the programming effort is worth it, but the complexity of programming does preclude using this approach in a more general way. Another approach to improving the speed of computers has been to "vectorize" the source code. This consists of having a post processor which takes, e.g. Fortran code, and tries to determine when an operation is being applied to a vector of values. This approach has not been especially successful chiefly because existing programming languages force programmers to strictly order their computations and this ordering is difficult to unravel. Another approach to faster computation is the construction of a multiprocessor, that is a set of processing elements connected together and perhaps sharing a common memory. One of the serious problems here is how to schedule the computational tasks so that a near maximum of parallel operation is achieved. This approach has suffered from the appearance of bottlenecks which all but reduce the multiprocessor to a uniprocessor and in some cases to a slow one.

What we intend to examine in this chapter is another approach to fast computing, the data flow model of computation. This model offers an entirely new way of viewing the execution of programs. Its virtue appears to be a simple and elegant way to take advantage of any parallelism which is inherent in a task without requiring the programmer to explicitly communicate it to the language translator. This is the goal of the data flow model of computation. Up to this time only a few