Predictive Block Dataflow Model for Parallel Computation

EE382C: Embedded Software Systems
Literature Survey

Vishal Mishra and Kursad Oney
Dept. of Electrical and Computer Engineering
The University of Texas at Austin

Abstract

Dataflow architecture as a concept has been around for a long time for parallel computation. In dataflow machines, the hardware is optimized for fine-grain data-driven parallel computation. The architecture can tolerate long unpredictable communication delays and supports generation and coordination of parallelism directly in the hardware. We present a coarse-grained dataflow model of computation for parallel architectures named *Predictive Block Dataflow (PBD)* due to its predictive capacity and processing of instructions in blocks instead of the fine grain model of one machine instruction. Its based on the dynamic dataflow model but differs in some ways that will be explained in section 4.

1 Introduction

Dataflow is an intuitively appealing, simple yet powerful model of parallel computation. In dataflow programming and architectures there is no notion of a single point or locus of control. There is no concept of program counter corresponding to the conventional sequential computational models. The dataflow model describes computations in terms of locally controlled events, each event corresponds to the "firing" of an actor. An actor can be a single instruction or a sequence of instructions. An actor fires when all the inputs it requires are available. In a dataflow execution, many actors maybe ready to fire simultaneously and thus represent many asynchronous concurrent computation events.

Dennis [1] developed the model of dataflow schemas, building on work by Karp and Miller [2]. These dataflow graphs, as they were later called, evolved from a method for designing and verifying operating systems to a base language for a new architecture. The first designs for such machines were made at MIT in the early 1970s by Dennis and Misunas [3] and Rambaugh[4]. These designs use dataflow graphs [5] to represent and exploit the parallelism in programs. We would discuss the basics of dataflow graphs and a few of the original machines in Section 2.

The failure of the original dataflow machines led to a new wave of hybrid multithreaded machines. It is argued that von Neumann and dataflow machines are not, in fact, orthogonal but rather sit on opposite ends of a spectrum of architectures. These hybrid architectures combine features of von Neumann and Dataflow. In section 3 we introduce the basic concepts, characteristics and evolution of multithreaded computer architectures with dataflow origin and briefly review a few of them.

Algorithms in signal processing tend to be modeled as compositions of components that conceptually run concurrently and communicate via signals. Typically, the signals are continuous functions of time continuum. However, modern signal processing applications involve more discrete-time signals than continuous-time signals. Instead of time continuum, a discrete clock is used to regulate the communication and the phase of the computation. The synchronous models globally order the tokens according to a global clock. In dataflow models, by contrast, signals are streams of tokens, where the relative order of the tokens within a stream usually matters, but the ordering of the tokens across streams is irrelevant. Dataflow presents a different abstraction to ordering by eliminating the global ordering of synchronous models and leaves it to a designated scheduler. Thus, the concurrency is even higher than the circuit-theory roots of the signals processing, since the tight coordination implied by a global ordering based on time is no longer required. It is arguable that this reduces stream-based processing to its computational essence.

We believe that the recent advances in chip fabrication have once again made the dataflow concept important for massively parallel architectures (MPA). We propose a model comprising of a 2-D multiprocessor array configuration on a single chip. In section 4 we describe our model of computation, which is a dynamic dataflow model with a few variations. Finally in section 5 we present a few comments and our project proposal.

2 Dataflow Architecture

Dataflow research made great strides since the seminal paper on dataflow graphs by Dennis [5]. In a Dennis dataflow graph, operations are specified by actors (or nodes) that are enabled only when all the actors that produce required data have completed their execution. The dependence relationships between pair of actors are defined by the arcs of a graph, which represent results forwarding by an actor to successor actors, and by the firing rules, which specify exactly what data is required for an actor to fire. Decision and control actors may be included to represent conditional expressions and iterations. They alter the routing of tokens to affect data-driven control. Data structures may be constructed and accessed by appropriate dataflow actors.
Two forms of dataflow architecture have become known:

- Static: The arc connecting one instruction to another can contain only a single result value (a token) from the source instruction. There can be only one instance of a dataflow actor in execution at any time. Thus concurrent reactivation of nodes (or actors) is not permitted and hence static systems are restricted to implementing loops and cannot accommodate recursion.

- Dynamic: Code-copying or dynamic tagging are used to permit recursive reactivation. In dynamic tagging, tags are conceptually or actually associated with tokens so that tokens associated with different activations of an actor may be distinguished. This enables arcs to simultaneously carry multiple tokens, thereby exposing more data parallelism.

Due to the inherently parallel nature of dataflow execution, dataflow computers provide an elegant solution to the two fundamental problems of von Neumann computers, namely memory latency and synchronization overhead, as described by Arvind and Iannucci in [6]. Latency is tolerated by dynamic switching between ready computation threads, whereas synchronization is supported in hardware leading to low overhead.
We now briefly discuss some basics behind three well researched early dataflow machines.

- Tagged Token Dataflow Architecture *(TTDA)*: [7] gives the details of the MIT TTDA dynamic dataflow architecture. It has a number of processing elements (PEs) or dataflow processors and storage units interconnected by an n-cube packet network. The storage units are addressed uniformly in a *global address space*. Each storage unit is an *I-structure* unit, which basically resides in global memory and introduces the notion of state into dataflow graphs. The main characteristic is that each datum or token

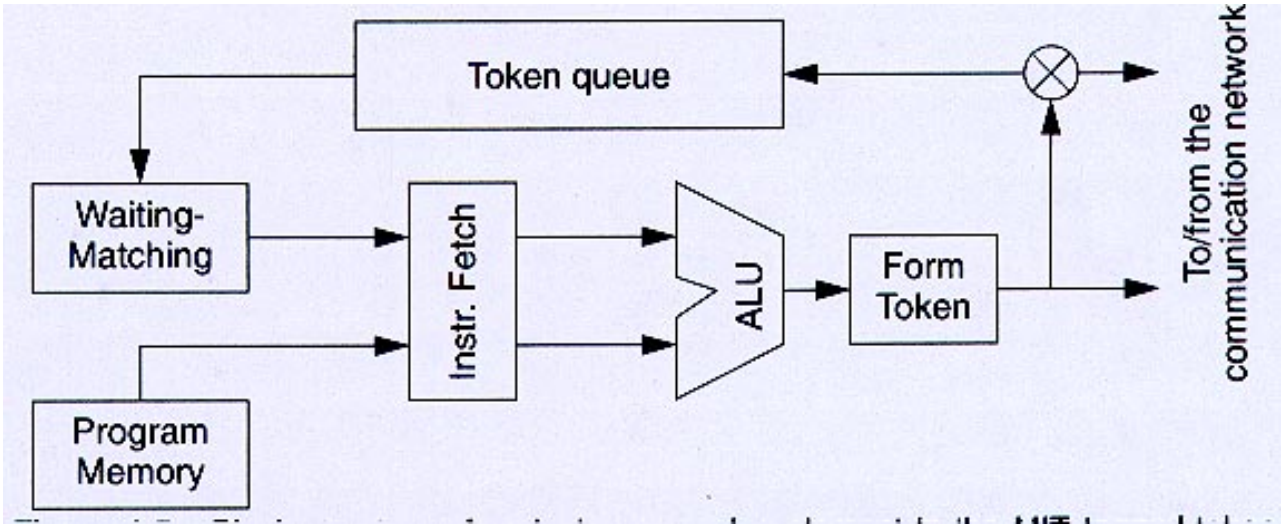

Figure 1: Block Structure of a Single PE in MIT's TTDA

is tagged with *context* identifier (format $< context, destination\_instruction, datum >$) that specifies the activation to which the token belongs. These tokens are matched at the destination node by *wait-match* units, which act as rendezvous points for pairs of arguments for dyadic operators.

- Manchester Prototype Dataflow Computer: It is a dynamic dataflow machine [8] very similar to TTDA. It also employs dynamic tagging like TTDA for identifying *iteration level* and to implement recursion by matching tokens by an associative lookup.

- Monsoon Dataflow Architecture: In 1987-88, the associative waiting-matching store of the above machines was replaced by an explicitly managed directly addressed token store, by Papadopoulous and Culler in the Monsoon machine [9]. They proposed an *Explicit Token Store (ETS)* model with the central idea that storage for tokens is dynamically allocated in sizable blocks. When a function is invoked, an *activation frame* is allocated explicitly to provide storage for all tokens generated by the invocation. A token now comprises of a value, a pointer to the instruction to execute (IP) and a pointer to an activation frame(FP).

The paper by Veen [10] gives a good background and comparison of some early dataflow research.

3 Multithreaded Hybrid Architectures

The dataflow model and von Neumann control-flow model are generally viewed as two extremes of execution models on which a spectrum of architecture models can be based. But it has been argued that the two models are in fact not orthogonal. Starting with the operational model of a pure dataflow graph, one can easily extend the model to support von Neumann style program execution. A region of actors within a dataflow graph can be grouped together as a thread to be executed sequentially under its own private program counter control, while the activation and synchronization of threads are data-driven. It has been speculated that there are machines along this spectrum which trade instruction scheduling simplicity for better low level synchronization and that there exists some optimum point between the two extremes *i.e.* a new hybrid model which synergistically combines features of both von Neumann and Dataflow, as well as exposes parallelism at a desired level. Such hybrid multithreaded architecture models have been proposed by a number of research groups with their origins in either static dataflow or dynamic dataflow. [11] is a good source for a discussion on the principal projects and representative work before 1995 in multithreaded computer architecture. Below we briefly discuss the basics of a few research projects.

- McGill Dataflow Architecture: It is inspired by the static dataflow model. The McGill Dataflow Architecture Model [12] has been proposed based on the *argument-fetching* principle [13]. The architecture departs from a direct implementation of dataflow graphs by having instructions fetch data from memory or registers instead of having instructions deposit operands (tokens) in *operand receivers* of successor instructions. The completion of an instruction will post an event (called a signal) to inform instructions that depend on the results of the instruction. This implements a modified model of dataflow computation called dataflow signal graphs. The architecture includes features to support efficient loop execution through *dataflow software pipelining*, and the support of threaded function activations.

- Iannucci's Model: Based on his research on the MIT Dynamic TTDA and the experience gained by [14], Iannucci combined dataflow ideas with sequential thread execution to define a hybrid computation model described in his Ph.D. thesis [15]. The ideas later evolved into a multithreaded architecture project at IBM Yorktown Research Center as described in [11]. The architecture includes features such as a cache memory with synchronization controls, prioritized processor ready queues and features for efficient process migration to facilitate load balancing.

- P-RISC: This hybrid model [16] is interesting in the sense that it explores the possibility of constructing a multithreaded architecture around a RISC processor. Nikhil and Arvind's P-RISC model split the *complex* dataflow instructions into separate synchronization, arithmetic and fork/control instructions, eliminating the necessity of presence bits on the token store (or frame memory) as proposed in the Monsoon machine [9]. P-RISC also permitted the compiler to assemble instructions into longer threads, replacing some of the dataflow synchronization with conventional program counter based synchronization .

- *T (StarT): This project [17] is a successor of the Monsoon project at MIT. It defined a multiprocessor architecture using an extension of an off-the-shelf processor architecture to support fine-grain communication and scheduling of user microthreads. The architecture is intended to retain the latency-hiding feature of the Monsoon split-phase global memory operations while being compatible with conventional MPA's based on von Neumann model. Inter-node traffic consists of a *tag* (called *continuation*, a pair comprising a context and instruction pointer). All inter-node communications are performed using the *split-phase* transactions (request and response messages); processors never block when issuing a remote request, and the network interface for message handling is well integrated into the processor pipeline. A seperate co-processor handles all the responses to remote requests.

4 PDB: Model of Computation

Our proposed model is basically a dynamic dataflow architecture with a few enhancements and variations. It is a 2-D mesh of processors with local memory connected by a network architecture that has not been decided as yet. This processor array executes a dataflow graph like other DDF systems i.e. a node is enabled when all the input data arrives on the input arcs. This schedules firing of the node and a token is sent on the output arc. There are many differences with respect to the early dataflow systems covered in section 2. It is not a fine-grain system but much more coarse-grained as a block of instructions is broadcast for execution to every node instead of a single instruction. There is a decoupled co-processor that pre-fetches the "block" of instructions based on a separate program produced by the compiler and then broadcasts or multicasts to the processor array. This program is based on the instruction dependence graph of the program. The size of a block has not been fixed but a block can end at either a function call or start/end of a loop. It is a trivial issue for which solutions are available. PBD is a message passing multiprocessor system, where *tokens* are

passed. The fields of any token include:

$$< destination\_node\_id, instruction\_pointer, value, context\_id >$$

This resembles the approaches followed in TTDA and Manchester with respect to dynamic tagging for context identity. All the processors are data-driven like a dataflow architecture with another subtle variation called the *owner computes rule*, which means only those nodes will be activated that have the data in their local memory. Our plan involves making all data accesses local and to exclude all remote data requests unless required. The operations are totally asynchronous and the need for synchronization is minimal. Synchronization can be performed on a need basis and the synchronization overhead will be tolerated. It is not expected to be an issue as dataflow models have low synchronization overhead as explained in section 2. The memory is addressed uniformly in a global address space. It is simulated using the processor ID as the MSB and the local memory in the node as LSB. There is a block level locality in this model unlike the fine-grained models which only had instruction level locality. This can be effectively employed for pipelining within each node. Portions of the graph are active at one time so some kind of spatial locality.

The main advantage that we are expecting to show is a huge speed up due to massively parallel operations. The broadcast mechanism enables execution of multiple instances of the same function and multiple loop executions concurrently. This model provides flexibility in terms of the data structure used. It is capable of handling all irregular data structures like trees and graphs unlike the vector processors which only performed effectively for vector data structures.

The work done till now has only been on at a higher level and we still have some issues remaining like flow control, termination detection, replacement policy for the blocks and exception handling. In this project we would be concentrating more on the potential speed-ups that can be achieved for various algorithms.

5 Comments & Proposed Work

The more recent Superscalar microprocessors emulate dataflow machines. The control flow program is converted to a dataflow graph in the instruction window, which is nothing but a dataflow graph. And then a reorder buffer retires the instructions in order. In a dataflow model like ours, the whole program is a dataflow model to exploit as much parallelism as possible. We believe it is more effective to expose the dataflow as a programming model than converting sequential instructions to and from dataflow during execution.

We plan to devise algorithms to implement quicksort and graph coloring on our model and analyze their performance in terms of time complexity.

References

[1] J. B. Dennis. Programming generality, parallelism and computer architecture. In *IFIP Congress*, volume 1, pages 484–492, 1968.

[2] Richard M. Karp and Raymond E. Miller. Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing. *SIAM Journal of Applied Mathematics*, 14(6):1390–1411, 1966.

[3] J. B. Dennis and D. P. Misunas. A Preliminary Architecture for a Basic Data-Flow Processor. In *Proc. Second Annual Symp. Computer Architecture*, pages 126–132, Houston, Texas, January 1975.

[4] J. E. Rumbaugh. A Data Flow Multiprocessor. *IEEE Computer*, 26(2):138–146, 1977.

[5] J. B. Dennis. First version of a data-flow procedure language. In *Proceedings of the Colloque sur la Programmation*, volume 19, pages 362–376. Springler-Verlag, 1975.

[6] Arvind and R. A. Ianucci. Two Fundamental Issues in Multiprocessing. In *Proceedings of DFVLR Conference on Parallel Processing in Science and Engineering*, pages 61–88, 1987.

[7] Arvind, D.E. Culler, R.A. Iannucci, V. Kathail, K. Pingali, and R. E. Thomas. The Tagged Token Dataflow Architecture. *Technical Report, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139*, 1984.

[8] J.R. Gurd, et al. The Manchester prototype dataflow computer. *Communications of the ACM*, 28(1), January 1985.

[9] Gregory M. Papadopoulos and David E. Culler. Monsoon: An Explicit Token-Store Architecture. In *25 Years ISCA: Retrospectives and Reprints*, pages 398–407, 1998.

[10] Arthur H. Veen. Dataflow Machine Architecture. *ACM Computing Surveys (CSUR)*, 18(4):365–396, 1986.

[11] R.A. Iannucci, editor. *Multithreaded Computer Architecture - A summary of the state of the art*. Kluwer Academic Press, 1994.

[12] G.R. Gao. An efficient hybrid dataflow architecture model. *Parallel and Distributed Computing*, 19(4):293–307, 1993.

[13] J. B. Dennis and G. R. Gao. An efficient pipelined dataflow processor architecture. In *Proceedings of Supercomputing '88*, pages 368–373, 1988.

[14] Arvind and R. S. Nikhil. Executing a Program on the MIT Tagged-Token Dataflow Architecture. *IEEE Computer*, 39(3):300–314, 1990.

[15] R.A. Ianucci. Toward a Dataflow/von Neumann Hybrid Architecture. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 131–140, 1988.

[16] R. S. Nikhil and Arvind. Can dataflow subsume von Neumann computing? In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 262–272, 1989.

[17] R. Nikhil and G. Papadopoulos. *T: A Multithreaded Massively Parallel Architecture. In *Proceedings of 19th International Symposium on Computer Architecture*, pages 156–167, 1988.