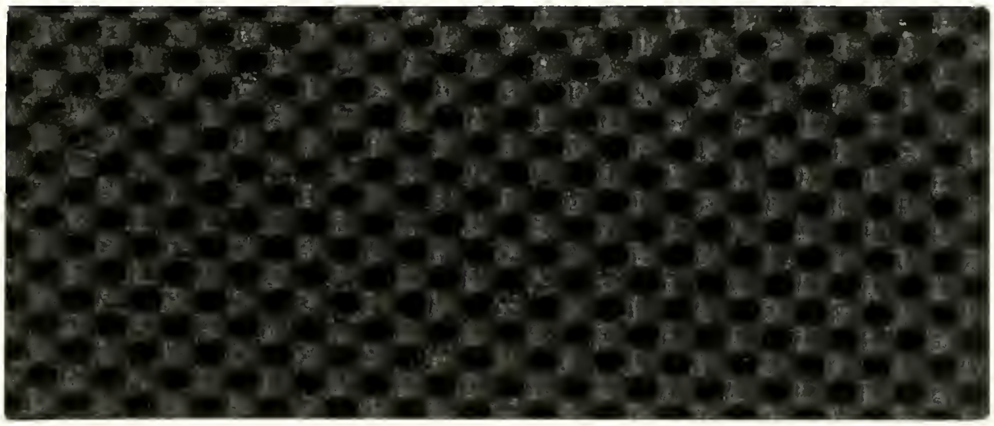
An Outsider's View of Dataflow

by

Allan Gottlieb

Ultracompute Note #164

July, 1989



An Outsider's View of Dataflow

by
Allan Gottlieb

Ultracompute Note #164
July, 1989

ABSTRACT

An outsider's view of the current state of dataflow research is presented. Two related observations are made. First, the field has matured from emphasizing proof of concept to pragmatic concerns needed to obtain high performance. Second, dataflow research is moving closer to mainstream parallel processing. Several examples of dataflow research supporting these observations are given. They include two-stage enabling rules and other eclectic scheduling disciplines, hardware and software throttling of parallelism, non-functional and (possibly) non-deterministic execution semantics, traditional memory management, and von Neumann compatibility.

Supported in part by Department of Energy contract DE-FG02-88ER025053.

1. Introduction

This report offers an outsiders view of the current state of dataflow research, emphasizing material presented at the Eilat workshop. My own research, as a part of the New York University Ultracomputer project, concerns more traditional von Neumann multiprocessors. Since this author is not an active member of the dataflow community, the present note cannot be viewed as an expert commentary on the subject but rather as the impression given by the workshop to an outside observer who has followed the field for many years. One (by no means surprising) observation is that the community is enthusiastic about their subject and confident that, in the not too distant future, dataflow machines will be among the leading supercomputers in existence. Indeed, as noted at one of the panels, I felt like a card-carrying agnostic in the land of true believers (a particularly apt analogy for a workshop held in Israel).

A more serious observation is that dataflow research has matured to the extent that, unlike the situation ten or even five years ago, the community now emphasizes the pragmatic considerations needed to get high performance. Occasionally, performance gains come at the expense of some of the simplicity and elegance found in Jack Dennis's original dataflow model. However these tradeoffs of elegance for performance have been done quite well. At the meeting, I jokingly called them contracts with the devil, but quickly added that the dataflow side negotiated well. Giving up a little elegance offered significant pragmatic gains. Often the loss of elegance appears as a more complex or eclectic design when compared with Dennis's original model.

A second observation is that dataflow research is moving closer to traditional (parallel) computing. For example, the accumulators recently added to to Id (Arvind *et al.* [78]), have enabled this dataflow language to support concurrent histogramming, an important technique used in scientific programming that had previously appeared to require a control-flow orientation. Other examples include various compiler techniques pioneered by the control flow community that are being adopted by dataflow researchers, the possible emergence of "permissible non-determinism" in dataflow languages, and traditional issues of memory management with the resulting need to throttle parallelism. Perhaps the best illustration of the convergence of the fields is the emergence of several designs for hybrid control/data-flow computers.

These two observations, that the dataflow community is striving for performance even at a possible loss of simplicity and that dataflow researchers are studying many of the same issues as their control-flow counterparts, are not independent. For example, hybrid designs, which are proposed in order to increase performance, have the two side effects of increasing complexity and introducing control-flow research questions. Indeed, a major reason that the dataflow community is studying issues that arise in control flow is to increase performance.

I wish to thank the organizers, Professors Lubomir Bic and Jean-Luc Gaudiot, for their considerable efforts that resulted in such a worthwhile workshop and for permitting me to attend. I also thank the other participants for their patience with an occasionally sceptical non-expert. The Eilat experience was for me both intellectually rewarding and a great pleasure.

2. Trading Simplicity for Performance

One example of the added complexity in modern dataflow design is the two-stage enabling rule for non-local data references used in a multiprocessor version of the Argument-Fetch Dataflow Processor proposed by Dennis and Rao [88]. In this architecture, the Dataflow Instruction Scheduling Unit (DISU) identifies instructions that are enabled for execution, i.e. whose arguments have been evaluated, and notifies the Pipelined Instruction Processing Unit (PIPU) when instructions are so enabled. When the PIPU executes an instruction requiring non-local data, it initiates the fetch and suspends execution of that instruction until the data arrives, in a sense waiting for the instruction to be re-enabled.

In contrast to early dataflow research where a major goal was to maximize the number of enabled instructions (a goal that the dataflow model was well suited to meeting), several of the presentations at Eilat discussed methods of limiting this number. Unchecked parallelism can quickly consume available resources; pathological cases were mentioned where free memory could be exhausted, thereby deadlocking the hardware, almost as fast as the reset button could be pressed. Both software techniques for controlling parallelism (k-bounded loops, which limit the number of active iterations) and hardware "throttles" were reported.

Two situations were discussed in which systems software for dataflow machines may need to deal with the complexity of multiple assignment semantics, a complexity the dataflow model was designed to avoid (and does avoid for applications software). The first situation occurs in those systems that garbage collect memory, which leads to frequently studied questions of collector/mutator concurrency. The second example occurred in the presentation by Israel Gottlieb of a coordination scheme for (macro) dataflow activities based on fetch-and-add, a read-modify-write instruction that, when used efficiently, leads to multiple concurrent assignments and requires care to avoid insidious race conditions that applicative semantics so nicely avoid. (This last point should be viewed as an advantage of applicative semantics rather than a condemnation of fetch-and-add; in fact the author is a proponent of the latter).

Several presentations discussed macro or hybrid dataflow designs in which an individual dataflow actor is no longer a simple (e.g., arithmetic) operator but is instead a von Neumann execution thread. The goal of these eclectic designs is to utilize the highly efficient von Neumann program counter semantics for purely sequential straight-line execution, reserving dataflow semantics for scheduling these straight-line pieces. Some of these designs explicitly propose separate von Neumann and dataflow processing elements. But even if only one processor is proposed, the dual scheduling discipline is naturally more complex than the original pure dataflow proposal.

3. Convergence of Dataflow and Control-Flow

The dataflow and control-flow research communities now study many of the same questions. Although the term *convergence* may well be too strong in that the fields will likely remain distinct, the disciplines have cross-fertilized and, I believe, have grown considerably closer during the decade. The hybrid designs mentioned above are clear examples of this convergence. This section gives other examples as well.

The first example involves the accumulators recently added to Id. I remember a dataflow presentation about five years ago at which a question concerning concurrent histogramming was raised by an computational scientist experienced with von Neumann multiprocessors. The situation described was a parallel program in which largely independent tasks would each repeatedly extract a datum from a global pile and then classify the datum into one of K types. The result required is a histogram giving the number of data of each type. A natural solution for a von Neumann multiprocessor. is to define a shared array of counters with one entry for each type and have each task increment the entry corresponding to the type of the datum that it has extracted. However, this solution violates single-assignment semantics and thus is not directly expressible in a dataflow language. Although dataflow solutions were suggested, each involved a space or time penalty and none was found completely satisfactory by the computational scientist.[1] With accumulators, a deterministic dataflow solution is possible that closely mimics the multiprocessor solution but avoids the space and time penalties mentioned above. An (addition-based) accumulator is established for each of the K types to serve as a counter of the number of data items of that type. Accumulators support an atomic operation that increments their value by a second value given as a parameter. Thus, like "von Neumann variables", accumulators take on many different values during their lifetime and readily yield an algorithm for concurrent histogramming. The key additional property is that only the *final* value of the accumulator can be read. It is this restriction that guarantees deterministic algorithms.

The above discussion of accumulators is adapted from the box entitled "The Rings of Id" presented in Almasi and Gottlieb [89], which in turn is strongly influenced by conversations with Arvind. That same box mentions another area where dataflow edges toward control-flow semantics: the limited use of non-functional and possibly even non-deterministic operations. The much studied I-structures do not satisfy functional semantics (very roughly speaking, they are single-assignment rather than zero-assignment structures) and hence dataflow languages like Id that include I-structures are not functional languages. As explained in "The Rings of Id" the intended use of I-structures is to define libraries of purely functional routines, which are then used in application programs. With this methodology only the library developer, and not the applications programmer, is exposed to non-functional semantics. Moreover, even if a methodology is adopted that promotes unrestricted use of I-structures, the resulting execution semantics contain many of the favorable characteristic of functional languages. In particular, determinism is maintained. Nonetheless, when compared to dataflow languages like Val (Ackerman and Dennis [79]) that do not support I-structures, Id has inched toward von Neumann semantics.

Notwithstanding the arguments presented for deterministic computations, there are occasions where support for non-determinism is important. Operating systems, programs for real-time control, and other software systems that interact with the (non-deterministic) external environment are natural examples. For these reasons, some members of the dataflow community are considering a weakening of the requirement

[1]It should be added that for interesting variations of this problem in which tasks insert as well as remove items from the data pile, the multiprocessor solution described is non-deterministic and thus would not be found completely satisfactory by dataflow researchers. Indeed, deterministic computations are a (largely achieved) goal of dataflow programming and no one challenges the assertion that stamping out non-deterministic bugs can be very difficult.

that all computations be deterministic. One perhaps fanciful possibility is to permit accumulators based on non-associative operators.

Some recent proposals for memory management on dataflow machines have advocated a rather traditional frame-based approach to the subject, which has been much studied in the context of von Neumann computers. As described in Papadopoulos [88], the Monsoon dataflow processor uses directly-addressed frames of memory instead of an associative waiting-matching store and [instruction pointer, frame pointer] pairs replace the tags found in previous tagged-token designs. Frames are also used in the dataflow/von Neumann hybrid architecture presented in Iannucci [88].

The subject of program compilation, in particular code generation and optimization, has long been a major area of study in computer science. It has also led to an amusing history of debates over the quality of compiled code for applicative versus imperative languages. Previously, the languages involved were usually an applicative subset of Lisp and the imperative language FORTRAN. More recently, newer purely functional languages have entered the debate (and on occasion C has replaced FORTRAN as the imperative representative). Initially the dataflow community did not emphasize compiler technology as strongly as did the control-flow community. Recall that a *primary* goal of Backus's original FORTRAN group was to generate high quality machine code (see Backus [78] for a history of the group). With hindsight it is probably fair to say that compiler technology should have received higher priority. Only recently have high performance compilers for dataflow languages been produced. Arvind, Culler, and Ekanadham [88] have carefully analyzed the Livermore "Simple" benchmark and found that an Id implementation comes within a factor of two of FORTRAN.[2]

Compiler technology is of considerable interest to the dataflow community at present. One illustration of this interest was the excitement generated when the workshop received a new University of Colorado technical report claiming that various programs written the the functional language Sisal (McGraw *et al.* [85]) executed with efficiency comparable to that obtained by equivalent programs written in FORTRAN and C. It was agreed that, if substantiated on a large application class, this compiler development would be extremely significant. (See also Bohm and Sargent [89] for recent results on numerical programming in Sisal.)

Naturally, these achievements did not come easily. In fact many of the compilation techniques developed for control-flow computers have been required in the dataflow/applicative arena as well. One might expect that the absence of side effects would make compilation of applicative language much less difficult than for imperative languages. However, obtaining competitive code quality has required significant development efforts.

In addition to the von Neumann influence in modern dataflow research, one can actually detect a measure of von Neumann compatibility. At Eilat, Kai Hiraki described the most powerful dataflow machine built to date, the ETL Sigma 1

[2]The metric used was the dynamic instruction count. This comparison was a harsh test for the Id compiler since the competition was the highly optimizing IBM compiler for the 370. The number of floating point operations executed was identical for the Id and FORTRAN versions.

supercomputer, containing 128 processing elements and achieving a performance in excess of 100 MFLOPS. Although Dr. Hiraki is a strong proponent of dataflow and dataflow programming languages, the production programming language for Sigma I is the very traditional imperative language C. Dataflow languages will surely be implemented but Hiraki felt that the first language should be one for which there was a large collection of programmers and software already present. Hiraki noted that, although it was fairly simple to write C programs that left 127 processors idle, good performance was obtained when the programmers followed certain guidelines that his group produced.

A second example of von Neumann compatibility in a dataflow design is the P-RISC proposed by Nikhil and Arvind [89]. This machine is a conventional RISC microprocessor with a small set of extensions to enable dataflow execution. The stated purpose of extending a standard microprocessor was to obtain (upwards) compatibility and hence the ability to run conventional software without modification.

The narrowing of the differences between dataflow and control-flow research is not simply the result of dataflow adopting control-flow ideas; each field has learned from the other. In particular, the idea of macro-dataflow is studied by researchers interested in von Neumann parallel processors. For example, early versions of the Ceder design from the University of Illinois (Gajski, *et al.* [83]) specified a "global control unit" that was a (hardware) scheduler for "compound functions", which are the moral equivalent of macro dataflow actors. Indeed, this version of Ceder was often referred to as a high-level dataflow machine. A software analog of the Ceder global control unit was suggested by Gottlieb and Schwartz for the NYU Ultracomputer [82]. These last two papers are early examples of a currently popular style of parallel processing where one establishes a workpile of tasks to be accomplished and schedules for execution entries from the subpile consisting of tasks with no remaining dependencies. A good recent example is the Schedule system of Dongarra *et al.* [88]. Although the dependencies processed by Schedule are not limited to dataflow dependencies (e.g. load balancing conditions are modeled), the system is in the spirit of large-grain dataflow.

4. Summary

When viewed from the outside, dataflow has progressed during the 1980s. A powerful multiprocessor has been constructed that achieves supercomputer performance exceeding 100 MFLOPS, the level of compiler development has increased, and in general the field has matured from emphasizing proof of concept to pragmatic concerns needed to obtain high performance.

In many areas of study, when a field is new the first questions are exciting and tend to be of the form "How can we do this?". When the field matures the questions turn to "How can we do this well?" and then to "How well can we do this?". For example, almost all recent papers in conventional computer architecture include significant quantitative comparisons, often based on simulation studies.

A natural consequence of the progression to more quantitative studies is that the issues involved become more detailed and often more complex. Compare the elegance of the concept of demand paging as presented in Fotheringham's 1961 paper on Atlas, with the detail of more modern work on the comparative performance of various heuristics for approximating LRU page replacement. This remark is not meant to minimize the importance of the later work. Good ideas do not guarantee good systems. Careful design and high quality engineering are required as well. Although the initial work on a subject is often the most elegant, the subsequent more pragmatic and detailed development of the idea can be as important for its ultimate success.

So it is with dataflow. The elegant dataflow model proposed by Dennis in the mid 70s has lead to an important line of research and development, which has included not only other elegant suggestions but solid engineering achievements as well, such as quality compilers and supercomputer performance. These achievements have required attention to detail and occasional contracts with the devil; but I know of no other path to a successful system. I end this report by asking those who joined me at Eilat to consider these last few paragraph as the long version of the (deliberately provocative) quip I made during a panel session: "Back when dataflow didn't work so well, it seemed a lot more elegant!".

References

William B. Ackerman and Jack B. Dennis, "VAL—A Value-Oriented Algorithmic Language: Preliminary Reference Manual", MIT LCS Tech. Rept. TR-218, June, 1979.

George S. Almasi and Allan Gottlieb, *Highly Parallel Processing*, Benjamin Cummings Publishing Co., 1989.

Arvind, Kim P. Gostelow, and W. Plouffe, "An Asynchronous Language and Computing Machine", Univ. Calif. Irvine Tech. Rept. TR114a, Dec. 1978.

Arvind, David E. Culler, and K. Ekanadham, "The Price of Asynchronous Parallelism: An Analysis of Dataflow Architectures" *Proc. COMPAR88*, Manchester, England, Nov. 1988, pp. 168-182.

John Backus, "The History of FORTRAN I, II, and III", *ACM SIGPLAN Notices*, **13** No. 8, Aug. 1978, pp. 165-180. This issue contained the preprints of the papers prepresented at the ACM SIGPLAN History of Programming Languages Conference, June 1978.

A.P. Wim Bohm and John Sargeant, "Code Optimization for Tagged-Token Dataflow Machines", *IEEE Trans. Comp.* **38** No. 1, Jan. 89, pp. 4-14.

Jack B. Dennis and Guang R. Gao, "An Efficient Pipelined Dataflow Processor Architecture", *Proc. Supercomputing Conf.*, Orlando FL, Nov. 1988.

Jack J. Dongarra, Danny C. Sorensen, Kathryn Connolly, and Jim Patterson, "Programming methodology and Performance Issues for Advanced Computer Architectures", *Parallel Comp.*, Oct. 1988, pp. 41-58.

J. Fotheringham, "Dynamic Storage Allocation in the Atlas Computer including an Automatic Use of a Backing Store", *CACM* 4 Oct. 1961, pp. 435-436.

Daniel Gajski, David Kuck, Duncan Lawrie, and Ahmed Sameh, "Cedar—A Large Scale Multiprocessor", *ICPP*, Aug. 1983, pp. 524-529.

Allan Gottlieb and Jacob T. Schwartz, "Networks and Algorithms for Very Large Scale Parallel Computation", *Computer* **15** Jan. 1982,, pp. 27-36.

Robert A. Iannucci, "Toward a Dataflow / von Neumann Hybrid Architecture" *Proc. 15th Ann. ISCA*, IEEE Comp. Soc., Honolulu, June 1988, pp. 131-139.

James R. McGraw *et al.*, "SISAL: Streams and Iteration in a Single Assignment Language—Language Reference Manual, version 1.2", Lawrence Livermore Nat. Lab. Tech. Rept., Mar. 1985

Rishiyur S. Nikhil and Arvind, "Can Dataflow Subsume von Neumann Computing?", *Proc. 15th Ann. ISCA*, IEEE Comp. Soc., Honolulu, June 1988, pp. 262-272.

Gregory M. Papadopoulos, "Implementation of a General-Purpose Dataflow Multiprocessor", Technical Report TR-432, MIT Lab. for Comp. Sci., Aug., 1988.