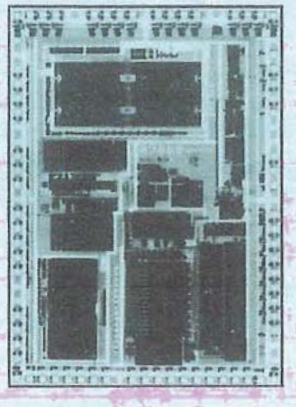
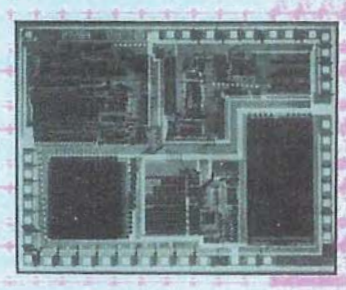
proceedings OF THE IEEE

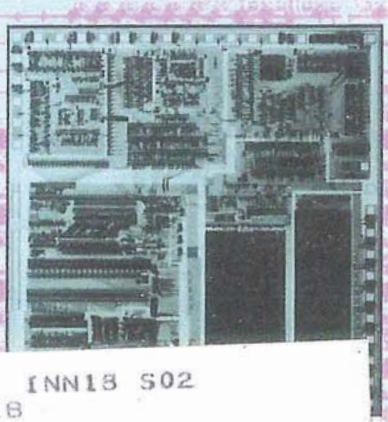
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS
(ISSN 0018-9219)

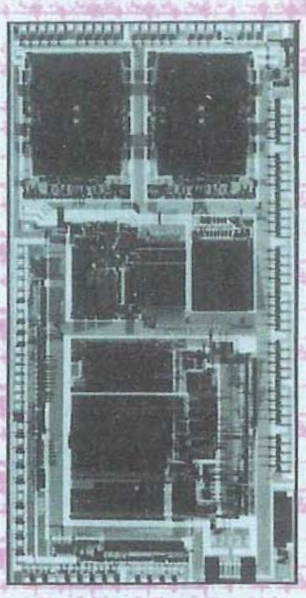
september 1987

SPECIAL ISSUE ON
ardware and software for DSP









proceedings OF THE IEEE

published monthly by the Institute of Electrical and Electronics Engineers, Inc.

september 1987

SPECIAL ISSUE ON
HARDWARE AND SOFTWARE FOR DIGITAL SIGNAL PROCESSING
Edited by Sanjit K. Mitra and Kalyan Mondal

COVER A family of successive digital signal processors (overlaying a programmable bit-level pipelined MAC chip) representing an important development within the topic area of this special issue.

PROCEEDINGS OF THE IEEE is published monthly by The Institute of Electrical and Electronics Engineers, Inc. **IEEE Headquarters:** 345 East 47th Street, New York, NY 10017-2394. **IEEE Service Center** (for orders, subscriptions, and address changes): 445 Hoes Lane, Piscataway, NJ 08854-4150. **Telephones:** Technical Editor 212-705-7906; Publishing Services 212-705-7560; IEEE Service Center 201-981-0060; Advertising 212-705-7579. **Postmaster:** Send address changes to PROCEEDINGS OF THE IEEE, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4150.

Annual Subscription: Member and nonmember subscription prices available on request. **Single Copies:** IEEE members $10.00 (first copy only), nonmembers $20.00 per copy. (Note: Add $4.00 for postage and handling charge to any order from $1.00 to $50.00, including prepaid orders.) **Other:** Available in microfiche and microfilm. Change of address must be received by the 1st of a month to be effective for the following month's issue. Send new address, plus mailing label showing old address, to the IEEE Service Center.

Advertising correspondence should be addressed to the Advertising Department at IEEE Headquarters.

Manuscripts should be submitted in triplicate to the Editor at IEEE Headquarters. A summary of instructions for preparation is found in the most recent January issue of this journal. Detailed instructions are contained in "Information for IEEE Authors," available on request. See note at beginning of "Proceedings Letters" for special instructions for this section. After a manuscript has been accepted for publication, the author's organization will be requested to pay a voluntary charge of $110 per printed page to cover part of the publication cost. Responsibility for contents of papers rests upon the authors and not the IEEE or its members.

CONTRIBUTED PAPERS

The PROCEEDINGS OF THE IEEE welcomes for consideration technical papers on topics of broad significance and long-range interest in all areas of electrical, electronics, and computer engineering. In-depth reviews and tutorials are particularly appropriate, although papers describing individual research will be considered if they meet the general criteria above. Papers that do not meet these criteria should be submitted to the appropriate IEEE Transactions and Journals.

It is suggested that a prospective author, before preparing a full-length manuscript, submit a proposal containing a description of the topic and how it meets the above criteria, an outline of the proposed paper, and a brief biography showing the author's qualifications to write the paper (including reference to previously published material as well as the author's relation to the topic). If the proposal receives a favorable review, the author will be encouraged to prepare the paper, which after submittal will go through the normal review process.

Please send papers and proposals to the Technical Editor, PROCEEDINGS OF THE IEEE, 345 East 47th Street, New York, NY 10017-2394, USA (Telephone: 212-705-7906).

Data-Driven Multicomputers in Digital Signal Processing

JEAN-LUC GAUDIOT, MEMBER, IEEE

New technologies of integration allow the design of powerful systems which may include several thousands of elementary processors. These multiprocessors may be used for a range of applications in signal and data processing. However, assuring the proper interaction of a large number of processors and the ultimate safe execution of the user programs presents a crucial scheduling problem. The scheduling of operations upon the availability of their operands has been termed the data-driven mode of execution and offers an elegant solution to the issue. This approach is described in this paper and several architectures which have been proposed or implemented (systolic arrays, data-flow machines, etc.) are examined in detail. The problems associated with data-driven execution are also studied. A multi-level approach to high-speed digital signal processing is then evaluated.

I. Introduction

If we are to approach the computational throughputs equivalent to billions of instructions per second which will be required from the processing systems of the future, improvements on all levels of computer design must be made. Faster technology and better packaging methods can be applied to raise clock rates. However, a one billion instructions per second machine would require a clock period as low as a nanosecond. This approach is inevitably bounded by physical limits such as the speed of light. Therefore, instead of considering the *technological approach* to performance improvement, we emphasize here the *architectural method*. Indeed, instead of merely increasing the clock frequency for a corresponding increase in overall throughput, performance can also be improved by allowing multiple processing elements to collaborate on the same program. This inevitably introduces synchronization problems, and issues of resource allocation and sharing must be solved. Programmability is indeed the central problem. In one solution, a conventional language such as Fortran is used to program the application. A sophisticated compiler is relied upon to partition a sequential program for execution on a multiprocessor. This approach has the advantage of imposing no "software retooling." However, complex numerical applications will not be easily partitioned and much potential parallelism may remain undetected by the compiler.

Ada, CSP [26], extended Fortran (e.g., HEP, Sequent), on the other hand, allow the programmer to deal with parallel processes by the use of primitives for parallel task spawning, synchronization, and message passing. However, while the programmer can express some of the parallelism characteristic of the application, much potential concurrency may never be uncovered because of the inherent sequential concepts of the language which must be countered through the use of special "parallelism spawning" instructions. Also, development time becomes important since the programmer must "juggle" with many parallel tasks to synchronize. In addition, debugging becomes correspondingly more difficult due to the sometimes undeterministic appearance of errors.

For these reasons, an *implicit* approach must be devised. In the above two methods, instruction scheduling is based upon a central program counter. We propose to demonstrate here the data-driven approach to programming multiprocessors: instructions can be scheduled by the *availability of their operands*. This model of execution is a subset of the *functional* model of execution [9]. It provides a significant improvement to the programmability of multiprocessors by excluding the notion of global state and introducing the notion of values *applied* to functions instead of instructions *fetching* the contents of memory cells as they are in the conventional "control-flow" model.

The overall objective of this paper is to demonstrate the applicability of data-driven principles of execution to the design of high-performance signal and data processing architectures. Several approaches will be demonstrated and their particular domain of application will be contrasted. The description of low-level processing systems is beyond the scope of this paper and the interested reader is referred to an excellent survey by Allen [3]. Instead, we will concentrate here on the issues related to building high-performance multiprocessors for signal processing applications. In Section II, we show the type of problems considered in signal processing. The data-flow principles of execution as they relate to digital signal processing problems are described in detail in Section III while several exist-

Manuscript received September 4, 1986; revised January 23, 1987. This work was supported in part by the Department of Energy under Grant DE-FG03-87ER 25043. The views expressed in this paper are not necessarily endorsed by the U.S. Department of Energy.

The author is with the Computer Research Institute, Department of Electrical Engineering—Systems, University of Southern California, Los Angeles, CA 90089, USA.

IEEE Log Number 8716208.

0018-9219/87/0900-1220$01.00 © 1987 IEEE

ing data-driven architectures are described in Section IV. In Section V, we analyze a multi-level data-driven architecture and examine its programming environment. Conclusions are drawn in Section VI.

II. The Requirements of Signal Processing

Digital signal processing techniques are applied to many different technical problems. These include radar and sonar systems, image processing, speech recognition, etc. The elementary building blocks of these were originally concentrated on such tasks as convolution, correlation, and Fourier transform. More complex algorithms (matrix operations, linear systems solvers, etc.) are now considered. Higher order operations include not only simple problems such as elementary filtering (IIR, FIR, etc.), but also more complex functions such as adaptive and Kalman filtering [45]. Also, such complex problems as Computer-Aided Tomography or Synthetic Aperture Radar can be considered [39], [16]. Signal processing algorithms are very appropriate for description by functional languages. Indeed, a signal processing algorithm is often represented in a graph form [36] and can be decomposed in two levels:

- a regular level which can be implemented by a *vector operation* (i.e., a loop in which all iterations present no dependencies among themselves);
- a level which contains conditional operations and heuristic decision making.

This description shows that the lower operational levels can easily deliver parallelism (by compiler analysis or programmer inspection). This layer usually consists of simple constructs (arithmetic instructions, FFT butterfly networks, simple filters, etc.). However, the higher levels will require more complex problem insight and even runtime dependency detection in order to allow maximum parallelism. We will now describe principles of execution which will allow us to deliver this concurrency.

III. Data-Flow Principles

The data-flow solution to the programmability problems of large-scale multiprocessors [5] has been pioneered by Adams [2], Chamberlin [11], and Rodriguez [43]. It is now described in detail in this section.

A. Basic Principles of Execution

In the conventional von Neumann model of execution, an instruction is declared executable when a Program Counter of the machine points to it. This event is usually under direct programmer control. While a control-flow program is a sequential listing of instructions, a data-flow program can be represented as a graph where the nodes are the instructions (*actors*) which communicate with other nodes over *arcs* (Fig. 1). An instruction is declared executable when it has all its operands. In the graph representation chosen above, this means that all the input arcs to an actor must carry data values (referred to as *tokens*) before this actor can be executed. Execution proceeds by first absorbing the input tokens, processing the input values according to the op. code of the actor, and accordingly producing result tokens on the output arcs. In summary, it can

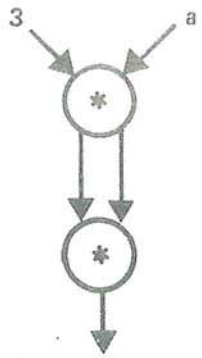


Fig. 1. A simple data-flow graph.

be said that the data-flow model of execution obeys two fundamental principles:

- *Asynchrony of operations:* The executability of an instruction is decided by a *local criterion* only. The presence of the operands can be sensed "locally" by each instruction. This is an attractive property for an implementation in a distributed environment where no central controller should be used for global scheduling.
- *Functionality of the operations:* The effect of each operation is limited to the production of results to be consumed by a specific number of other actors. This precludes the existence of "side-effects." These side-effects may be long-ranging in that the execution of an instruction may effect the state of a cell of memory which will be used only much later by another unrelated operation.

B. Data-Flow Interpreters

When iterations are executed, the underlying principle of data-flow (*single assignment of variables*) must invariably be violated. Indeed, for an actor to be repeatedly evaluated as in an iteration, its input arcs must carry several tokens (from different iterations). Several solutions have been proposed to allow the *controlled violation* of these rules without compromising the safe execution of the program. Among these, the Acknowledgment scheme and the U-interpreter have been given the most consideration.

1) Acknowledgment Scheme [14]: Proper matching of the tokens can be observed by ordering the token production. This would be done by a careful design of the program graph so as to insure that tokens of two different iterations can never overtake each other. In addition, it must be guaranteed that *no* token pileup is encountered on *any* one arc. This condition can be verified by allowing the firing of an actor when tokens are on all input arcs *and* there are no tokens on any output arcs. In order to enforce this last condition, an *acknowledgment* must be sent by the successor(s) to the predecessor when the token has been consumed (Fig. 2). Note that an actor is executable when it has received its input arguments as well as all acknowledgments. The parallelism which can be exploited from this scheme is mostly pipelining between the actors of different iterations. Thus when the number of instructions in the body of an iteration is the same as the number of available processors, the speedup observed by this mechanism of execution is maximal. However, for small iterations (compared to the size of the machine), the exploited parallelism

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.