A COMMON OPERATIONAL SOFTWARE (ACOS) APPROACH TO A SIGNAL PROCESSING DEVELOPMENT SYSTEM

*Y.S. Wu

U.S. Naval Research Laboratory
Washington, D.C. 20375 USA

ABSTRACT

A Signal processing problem with moderate complexity must be segmented into "smaller" algorithms, intimately linked, to be executed by processing resources. Problems arising from such partitioning and interconnection of these constituent parts are not well understood. It is further complicated by the parallel execution of the elementary operations (primitives) which require the formation of queues and synchronization of the queue with respect to each operation. A software development system has been completed to program and to design the signal processor systems for underwater acoustics applications. This is the first operational signal processing software development system of its kind which enables the system designer to program in signal processing graph notations. The implied hierarchical architecture will lead to the interface to VLSI cell libraries for direct mask generation. A complete computer aided design approach to closely-coupled signal processing systems is described.

Introduction

A signal processing problem with moderate complexity must be segmented into "smaller" untis of work or algorithms, intimately linked, to be executed by processing resources. Problems arising from such partitioning and interconnection of such constituent parts are not well understood. It is further complicated by the parallel execution of the elementary operations (primitives) that require the formation of queues and synchronization of the execution of the queue with respect to each operation in either hardware or software. A computer-aided design approach becomes a necessity for development, study and simulation of these signal processing systems. To address this need, a common operational software (ACOS) support system has been developed by the US Naval Research Laboratory (NRL), Washington, D.C. for the production of modular signal processing software for acoustic signal processing applications. The ACOS system provides the following facilities:

- Signal Processing Graph Notation (SPGN)
- Program generator for SPGN
- Runtime support for graph executions
- A library of signal processing primitives

*The author is at present an academic visitor at the Imperial College of Science & Technology, London, UK.

An ACOS user first writes a number of procedures in graph notation. They specify control processing or signal processing functions to be performed to meet application requirements. The user specifies how procedures are related to another by defining a set of graphs. These graphs specify the topological connections among primitives such as "Filter," "FFT," "Line Integration" etc., which can be viewed as the nodes of graphs by identifying primitives and queues to be used to exchange data between pairs of primitive executions or between an execution and external sources. This user software, comprised of nodes and graphs, is processed by the program generator and the SPL/I Compiler, and loaded into the target hardware. SPL/I (1) is a standard US Navy programming language.

The program generator takes the responsibility for generating or executing, as appropriate, programs to manage storage and to initiate data transfers and to execute complex functions. The ACOS SHELL (SHELL is a term borrowed from the UNIX Operating System and UNIX is a Trademark of Bell Laboratories) controls and manages the access to hardware resources and is the runtime signal processing controller. Thus, the underlying hardware architecture is hidden from the user. The SHELL is accessed only by software generated by the ACOS program generator and the programmer is never aware of the distribution of functions between the SHELL and the hardware. At runtime the hardware executes the nodes under the supervision of the SHELL. Tasks are executed by the SHELL in response to the presence of data in the queues specified in graphs. An instance of a graph is created at runtime by starting the graph. Starting the graph causes the SHELL to begin monitoring the graph-specified queues. Upon discovering the presence of sufficient data in one or more than one of the queues, the SHELL will cause execution of graph-specified nodes.

Although ACOS is designed for the acoustic signal processing applications, it is of sufficient generality for use in other signal processing applications such as radar and image processing. In addition, because architecture details are buried in the primitive definitions and the SHELL implementation, the resulting operational software is highly machine transportable. Eventually, ACOS nodes and graphs could be mapped into VLSI layouts for direct hardware mask generation.

Signal Processing Graph Notation (SPGN)

The theory of directed graphs has been well established. Indeed, several good text books are in existence [2,3]. However, its application to signal processing and, in particular, the programming of signal processing systems have not received wide attention. The signal processing community traditionally concentrates on algorithm studies but not the topology and the decomposition of these algorithms. Although computer scientists have been attacking parallel or concurrent processing problems during the past two decades for a general class of problems, there is a total lack of interest, understanding and appreciation of signal processing applications by the computer science community. ACOS represents the first opportunity for support software existed to bridge the gaps of the appropriate interdisciplinary research areas for computer aided signal processing system design and implementation.

With directed graphs, as expressed in SPGN, a graph consists of a set of vertices $\{V_i\}$, a set of edges $\{e_j\}$ and a mapping that maps every edge onto an ordered pair of vertices (direction). To be consistent with the electrical network theory terminology, SPGN considers a set of "nodes" interconnected by a set of "queues," (branches for the analog network) and each node represents some form of signal processing operation as depicted in Fig. 1. Each node can also be a directed subgraph which contains a set of nodes and queues.

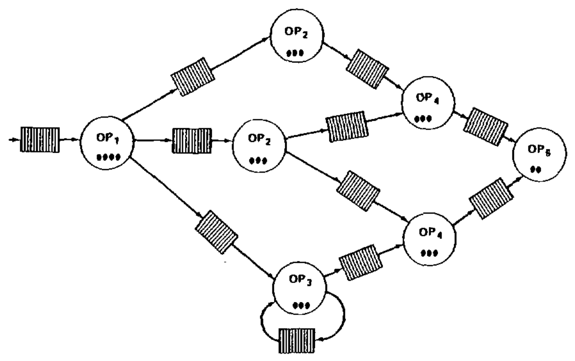

Fig. 1 Signal Processing Flow Graph

A node represents a transfer function and is completely implementation-free. There are two types of queues: the data queue and the trigger queue. Data queue is the buffered data between signal processing operations, whereas triggered queue is an "event signal," which allows synchronization of asynchronous sequence of operations. A node contains a primitive or a subgraph (such as FFT or Beamform). Node executions are controlled by its input/output data queues: input queues must reach certain "thresholds" and output queues must have "capacity" to receive data before the node is ready for execution. A queue valve can be specified and controlled to enable or to disable the associated queue.

A set of ACOS SPGN elements have been defined. A majority of SPGN elements are declaration statements. These are:

- GRAPH
- MODE
- PRIMITIVE
- SUBGRAPH
- Variable
- QUEUE
- Command Program

In addition, a NODE statement is used to map declarations onto the graph topology and an INVOKE statement is needed for the command programs which will be described later. An ENDGRAPH statement is included to complete the specification. The signal processing graph notation is specified in the ACOS User's Guide[4].

Command Programs and Control Primitives

User specifications of SPGN NODEs and QUEUEs determine the graph topology of a given signal processing application. It is relatively simple to program. However, the dynamic behavior of the graph is controlled automatically with manipulation of input/output QUEUEs at each NODE. Therefore, a command program (COMPROG) is required for the control and mode change purposes. When an instance of the GRAPH is initiated, the corresponding COMPROG activates the path of execution by "firing" interconnected nodes in a proper sequence. The ACOS graph topology is the initial static configuration. The COMPROG can alter the initial configuration dynamically by creating application specific configuration for exectuion. In other words, the execution of the COMPROG causes data to flow through the GRAPH to perform the signal processing function as specified. Theoretically, one can generate application independent GRAPHs; however, COMPROG is always application dependent. The command program starts and stops graphs. It connects and disconnects input/output queues. It also assumes the critical roles of application control and mode changes.

As originally conceived, there should be a corresponding control graph for each signal processing graph. It was hoped that command programs could also be programmed in graph notations and high level control primitives. Very little experience existed in this approach. ACOS opted to have the command programs programmed in SPL/I and to learn by experience the control primitives required. It was also believed that the control primitives should be a finite set, while the signal processing primitive library is limited but extensible.

An ACOS command program is an SPL/I program with special statements to:

- Start Graph
- Stop Graph
- Enqueue
- Dequeue
- Read queue
- Connect Trigger
- Disconnect Trigger
- Add Trigger
- Wait Trigger

etc.

The command program with the ACOS runtime support program, SHELL, will allocate storage, set-up the initial graph configuration and processes subsequent changes in configuration during graph execution.

SHELL—ACOS Runtime Support

The SHELL provides the hardware and operating system interface for ACOS. It instrumented a "shell" to shelter the user from the runtime environment. It initializes graphs, schedules nodes and manages the queues for the command program to direct the graph execution. Special command program statements listed above are SHELL services. SHELL monitors two types of activities: graphs to be executed and queues to be stored. As previously mentioned, firing of a node is caused by the data flow requirements and then, the node executes the associated primitive under the control of SHELL.

SHELL was implemented at first in SPL/I. Some high duty-cycle services were migrated into assembly-code programs (approximately 30%) to gain efficiency. Further migration into microcode and hardware is possible. Ultimately, SHELL service routines could become command program statements. It defines a set of control primitives. Then, the command programs can be programmed in these control primitives for interpretive executions.

Architecture Implications

The node or its underpinning primitive constitutes the minimum processing element such as filter banks, FFT devices etc. These elements are interconnected with some memory modules for buffered queues and controlled by a fixed sequence or by a control processor. In ACOS SPGN, it is allowed that a node can be a subgraph. Therefore, each node (or processing element) should have sufficient intelligence to conform with the ACOS application topology as shown in Fig. 2. Note that the interconnection "Ether" is transparent to the user.

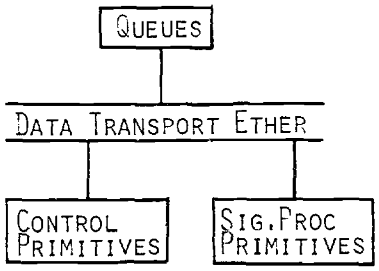

Fig. 2 A Node

Despite the claims by ACOS enthusiasts that ACOS SPGN embraces the data flow concept, the natural hardware mapping of Fig. 2 is the conventional Von-Neumann architecture. It does not fit the Newcastle definition of data flow.(5) SPGN is queue-driven, i.e. data driven but without data dependency.

The lowest level architecture, Level 0, is a low level signal processing primitive or a signal processing arithmetic unit. ' The Level 1 architecture is a node capable of subgraph executions. A control processor must be included to execute the command programs and control primitives. The hardware realization of Level 1 architecture has been examined by this author and others.(6,7,8,9) Several Level 1 processors are commercially available. These are in the general class of "array processors" including a (or several) pipe-lined signal processing arithmetic unit as shown in Fig. 3.

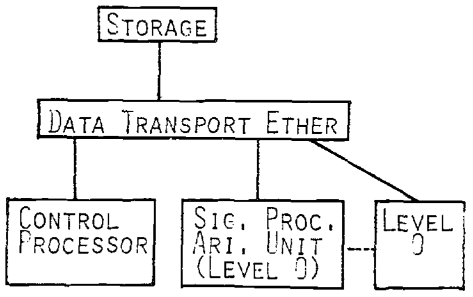

Fig. 3 Level 1 Architecture

Analysis and experiences indicated that the through-put of the Level 1 architecture is limited by the capability of the control processor.(6) In today's available semi-conductor technology, storage capacity and arithmetic speed impose no constraint on Level 1 design. In ACOS methodology, this implies that the largest subgraph (or graph) that can be realized in a Level 1 processor, is determined by the number of nodefirings per unit time in its control processor. Higher level architures can be obtained by partitioning the application into Level 1 subgraphs (nodes), and these Level 1 nodes are controlled and sequenced by an additional control processor executing a Level 2 command program. To be completely general, a hierarchical relationship is shown in Fig. 4 for the Level n architecture.

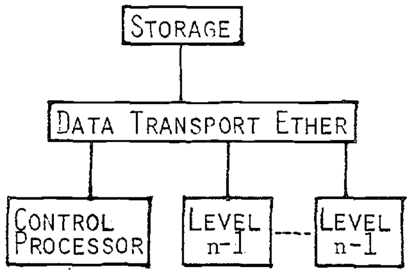

Fig. 4 Level n Architecture

The ACOS methodology makes no assumption on hardware implementation. In fact, there is no distinction between software and hardware architectures. As to physical realizations at various levels, primitive library functions are allowed to migrate from high level language programs to special purpose hardware (10,11) when performance requirements justify the cost of a particular design decision, and the distribution of storage at various levels will influence the design of the data transport ether.

Future Research

All top-down analysis systems fall short on synthesis. ACOS represents a systematic and structured methodology for signal processing system development. ACOS is a tool for system synthesis. At present, ACOS has demonstrated that a practical Naval acoustic system can be designed and programmed in SPGN. It forms the basis for the development approach for all future US Naval acoustic signal processing systems. However, it still requires ad hoc partitioning of algorithms to obtain the initial SPGN topology. Graph theoretic simulation techniques can be applied to investigate the queuing and synchronization strategies for decomposing relatively complex problems such as adaptive (or optimum) processing and multi-dimensional compression algorithms. Hopefully, a general simulation model can be formulated as part of the SPGN design and analysis aids. Recent work in formalism for describing a system as a set of sequential processes and distinct resources (12,13) probably can support this formulation. Because the path of graph execution determines the dynamic configuration of ACOS applications, partition for reliability to allow alternate or redundant paths should be an interesting study. Through design automation, it should be feasible to map ACOS Level 0 architecture to VLSI cell libraries. The ACOS interface to VLSI/VHSIC design automation systems is an area of investigaion that will lead to high pay-offs in direct mask generation for future DOD technology developments.

Acknowledgement

For the past ten years, it has been a honor to be associated with the evolution of the US Navy signal processor development and to witness its impact on the Navy R&D community. ACOS may symbolize only a way station in this "long march." The author wishes to express his gratitute to all his comrades who have supported this effort with dedication and technical excellence, many of whom are no longer with the program. In particular, he would like to thank W.R. Smith, E. Freeman, G.R. Lloyd, R.J. Harrington, P.A. Rigsbee, S.X. Weinstein, and S.L. Zuckerman for the privilege of leading an outstanding technical team that could never have been assembled before in the Navy computing community; E.E. Wald and D. Kaplan whose determination and hard labor nursed ACOS to maturity; Capt. C.M. Rigsbee, USN (ret) and Capt. Christopher Robbins, USN for their unflinching management support. Efforts and participations from colleagues of other Navy laboratories and contractor personnel are also greatfully acknowledged.

Reference

1. SPL/I Language Reference Manual, Naval Research Laboratory, Washington, D.C. February, 1977
2. Christofides, N. "Graph Theory—An Algorithmic Approach" Academic Press, New York, London (1975)
3. Harary F., Norman, R.Z. & Cartwright, D. "Structural Models: An Introduction to the Theory of Directed Graphs" Wiley, New York (1965)
4. Anker, D.S. et al "ACOS User's Guide" (Draft) Naval Research Laboratory" Washington, D.C. 17 March 1982
5. Treleaven, P.C. et al "Data Driven and Demand Driven Computer Architecture" Univ. Newcastle upon Tyne Computing Laboratory Report #168, Newcastle upon Tyne, June 1981
6. Wu, Y.S. "Architectural Considerations of A Signal Processor Under Microprogram Control" AFIPS Vol. 40 pp 675-682 April 1972
7. Kratz, G.L. et al "A Microprogrammed Approach to Signal Processing" IEEE Trans. Comput. C-3 (No. 8) pp 808-817 Aug. 1974
8. Wu, Y.S. "Microprogramming Applications to Signal Processing Architecture"—Micro-Architecture of Computer Systems, R.W. Hartenstein and R. Zuks (eds) North Holland, Amsterdam (1975)
9. Liu, B. and Peled, A "Digital Signal Processing" Wiley, New York (1979)
10. Wu, Y.S. and Lloyd, G.R. "System Functional Migration" AIAA Computer Conference (invited paper) Los Angeles, Ca. Dec 1977
11. Van Dam, A, Stabler, G and Harrington, R. "Intelligent Satellites For Interactive Graphics" Proc. IEEE Vol 62, no. 4, pp 483-492, April 1974
12. Lauer, P.E. et al "COSY—A System Specification Language Based on Paths and Processes" Acta Informatica, Vol 12 pp 109-158 1979
13. Hoare, C.A.R. "Communicating Sequential Processes" CACM Vol 21, No. 8, Aug 1978