DESIGN METHODOLOGY FOR DSP

Edward A. Lee, Principal Investigator
Department of Electrical Engineering and Computer Science
University of California, Berkeley CA 94720

Final Report 1994-95, Micro Project #94-075

Industrial Sponsors: Bell Northern Research, Dolby Laboratories, Mentor Graphics, Philips Research, and Rockwell International.

ABSTRACT

The focus of this project is on design methodology for complex real-time systems where a variety of design methodologies and implementation technologies must be combined. Design methodologies are encapsulated in one or more models of computation, while implementation technologies are implemented as synthesis tools. Applications that use more than one model of computation and/or more than one synthesis tool are said to be heterogeneous. Hardware/software codesign is one example of heterogeneous design. Project results have been disseminated via the Ptolemy software system, in addition to papers. The overall Ptolemy project is fairly large, with additional support from ARPA, SRC, and a number of other companies, and is strongly collaborative. The MICRO portion of the project has focused on real-time signal processing, although the larger project is broader.

1. THE CONTEXT

The objectives of the Ptolemy Project include most aspects of designing signal processing and communications systems, ranging from designing and simulating algorithms to synthesizing hardware and software, parallelizing algorithms, and prototyping real-time systems. Research ideas developed in the project are implemented and tested in the Ptolemy software environment. The Ptolemy software environment, which serves as our laboratory, is a system-level design framework that allows mixing models of computation and implementation languages.

In designing digital signal processing and communications systems, often the best available design tools are domain specific. The tools must be able to interact. Ptolemy allows the interaction of diverse models of computation by using the object-oriented principles of polymorphism and information hiding. For example, using Ptolemy, a high-level dataflow model of a signal processing system can be connected to a hardware simulator that in turn may be connected to a discrete-event model of a communication network.

A part of the Ptolemy project concerns programming methodologies commonly called “graphical dataflow programming” that are used in industry for signal processing and experimentally for other applications. By “graphical” we mean simply that the program is explicitly specified by a directed graph whose nodes represent computations and the arcs represent streams of data. The graphs are typically hierarchical, in that a node in a graph may represent another directed graph. In Ptolemy the nodes in the graph are subprograms specified in C++.

It is common in the signal processing community to use a visual syntax to specify such graphs, in which case the model is often called “visual dataflow programming.” But it is by no means essential to use a visual syntax.

Hierarchy in graphical program structure can be viewed as an alternative to the more usual abstraction of subprograms via procedures, functions, or objects. It is better suited than any of these to a visual syntax, and also better suited to signal processing.

Some other examples of graphical dataflow programming environments intended for signal processing (including image processing) are: Khoros, from the University of New Mexico (now distributed by Khoral Research, Inc.), the signal processing worksystem (SPW), from the Alta Group at Cadence (formerly Comdisco Systems), COSSAP, from Synopsys (formerly Cadis), and the DSP Station, from Mentor Graphics. MATLAB from The Mathworks, which is popular for signal processing and other applications, has a visual interface called SIMULINK. These software environments all claim variants of dataflow semantics.

Most graphical signal processing environments do not define a language in a strict sense. In fact, some designers of such environments advocate minimal semantics, arguing that the graphical organization by itself is sufficient to be useful. The semantics of a program in such environments is determined by the contents of the graph nodes, either subgraphs or subprograms. Subprograms are usually specified in a conventional programming language such as C. Most such environments, however, including Khoros, SPW, and COSSAP, take a middle ground, permitting the nodes in a graph or subgraph to contain arbitrary subprograms, but defining precise semantics for the interaction between nodes. We call the language used to define the subprograms in nodes the host language. We call the language defining the interaction between nodes the coordination language.

Many possibilities have been explored for precise semantics of coordination languages. Many of these limit expressiveness in exchange for considerable advantages such as compile-time predictability. In Ptolemy, a domain defines the semantics of a coordination language, but domains are modular objects that can be mixed and matched at will. Thus we gain flexibility without the sloppiness of unspecified semantics in the coordination language.

Graphical programs can be either interpreted or compiled. It is common in signal processing environments to provide both options. The output of compilation can be a standard procedural language, such as C, assembly code for programmable DSP processors, or even specifications of silicon implementations. We have put considerable effort into optimized compilation.
2. RESULTS OF MICRO SUPPORT

2.1. Methodology

We have made progress on several different models of computation and their interaction. In dataflow modeling, we have formalized relationships among actors using partial orders and characterized the various types of dataflow in terms of their mathematical properties. We have also developed a new efficient robust scheduler for dynamic dataflow and investigated efficient dataflow representations for multidimensional multirate systems. The Process Networks model is a superset of the dataflow models and schedules blocks as processes (threads) under the control of a multitasking kernel. In other areas of modeling, we have researched formal models for characterizing hierarchical finite state machine (FSM) controllers. For interaction between models of computation, we have studied the mixing of FSM controllers with dataflow models, FSM controllers with discrete-event models, and dataflow models with discrete-event models.

In implementation, we have made fundamental contributions in optimized synthesis of embedded software on both uniprocessor and multiprocessor architectures. For uniprocessor code generation, we can trade off program size, data size, and throughput. For parallel code generation, we have made progress in optimizing synchronized communication between processors. We have also made contributions in hierarchical scheduling and incremental compilation and developed methodologies for converting (untimed) models of computation into (timed) clocked circuits implemented in VHDL. And we have made progress in integrated multi-lingual design, where for example a subsystem specified in VHDL can interact with one specified in C, which in turn can interact with another specified in assembly code for a DSP.

In hardware/software codesign, we have derived an efficient algorithm to partition dataflow graphs into a combined hardware/software implementation. We have also made progress in design methodology management by defining an abstraction for capturing the dependencies between the data, design representations, and tools used in a complex design process. Having a formal design model allows heterogeneous design styles to cooperate in a productive and predictable manner.

2.2. The Ptolemy software

We have successfully tested many theoretical ideas in modeling, implementation, system design, and user interfaces in the Ptolemy software environment. We have distributed two new releases of the software, which runs on ten different Unix architectures. These releases contain many improvements to the Ptolemy interactive graphical interface, including highly interactive, animated simulation capabilities. The GUI now supports a constructive and predictable manner.

2.3. Plans

The top priority topics that we will be addressing over the next year are:

- hierarchical syntax management,
- hardware synthesis from dataflow graphs,
- mixed control and dataflow,
- interaction of synchronous languages and dataflow,
- "cyclo-static" dataflow model for scheduling,
- redesigned image processing infrastructure,
- integrated symbolic computation,
- graphical design, and
- automated regression tests for the software.


November 1, 1995.  (http://ptolemy.eecs.berkeley.edu/ ~pino/papers/asilomar-95)


