Proceedings of the First Annual Conference of the
Rapid Prototyping of Application Specific Signal Processors (RASSP) Program

Design and Simulation of Heterogeneous Systems using Ptolemy

Brian L. Evans, Alan Kamas, Edward A. Lee

Department of Electrical Engineering and Computer Science
University of California, Berkeley CA 94720-1770

ABSTRACT

The ambitious objectives of the Ptolemy project include practically all aspects of designing signal processing systems, ranging from the design and simulation of algorithms to the generation of hardware and software, the parallelizing of algorithms, and the prototyping of real-time systems. To manage these abilities, it is essential that the design software be highly modular and extensible, so that the development of subprojects of manageable scope can proceed unencumbered and in parallel, while at the same time allowing subprojects to interact and to be combined into a complete working system description. Since subprojects must use the best available tools (which are often domain-specific), the tools must also be able to interact. The Ptolemy software architecture shows one way in which such interaction can be achieved, using object-oriented principles of polymorphism and information hiding.

1. Introduction

In signal processing systems, heterogeneity arises in two ways:

- Diverse implementation technologies are combined (such as hardware, software, or subsystem integration).
- Diverse models of computation are used to describe the system being implemented.

Diverse implementation technologies dominate in later phases of the design, in which validation is complicated by the mixture of implementation technologies. Diverse models of computation dominate during early phases of system-level design, where domain-specific, high-level tools are most effectively used. By "model of computation" we mean the operational semantics of a network of functional blocks. The objective of the Ptolemy project is to seamlessly support both forms of heterogeneity through carefully conceived software architecture, and to support seamless migration from system-level design to detailed design.

An example of the way that Ptolemy can support diverse implementation technologies is shown in figure 1. The design of each subsystem is carried out using the best available methodology and tools for that subsystem, at the highest level of abstraction possible. For example, signal processing software can be specified using a block diagram with dataflow semantics, and hardware can be specified at the architectural level as an assemblage of high-level building blocks. The subsystems are then combined for cosimulation and/or cosynthesis in a modular and transparent way.

An example of the way that Ptolemy can support diverse models of computation in a system-level design is shown in figure 2. In figure 2, the model of a broadband packet network includes the interactions between three key elements: signal processing (video and audio compression), transport (ATM, or asynchronous transfer mode, a high-speed networking protocol), and control (signaling and call processing). The signal processing is modeled again using block diagrams with dataflow semantics, while the transport and control are modeled using discrete-event semantics.

Ptolemy is an object-oriented system and achieves its goals using the principle of polymorphism. The system consists of a *kernel* and an extensible number of *domains*. The Ptolemy kernel, written in C++, defines the basic classes that allow the components of the system to function together. These classes are generic, and do not assume any particular model of computation. From these classes, application-specific objects are derived to define a domain. Information hiding and data abstraction are key to the design; the system is extensible in many dimensions without the need to modify the kernel.

Each design style is supported by one or more *domains*. A domain is an extensible library of functional blocks and a set of classes derived from the kernel classes to implement a particular model of computation. Key to Ptolemy is the ability to combine multiple domains in a single design. Table 1 summarizes some of the domains on which we are currently working.

Using heterogeneous design styles also requires using tools from different sources. We have already demonstrated varying levels of integration between Ptolemy and

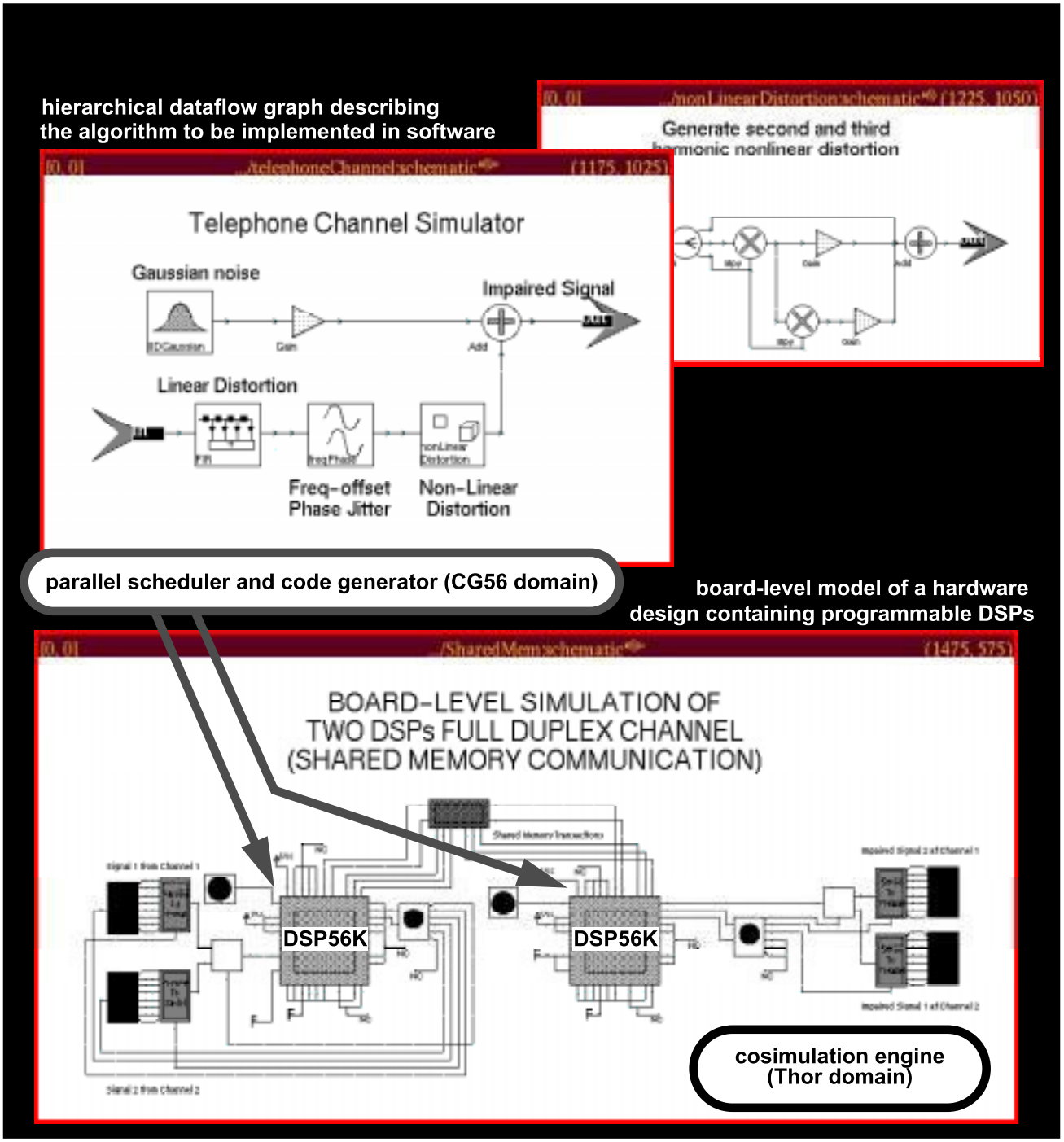

Figure 1. A board-level specification of a hardware design containing programmable DSPs (bottom window) is combined with a high-level block-diagram representation of the algorithm to be executed by the DSPs. Ptolemy handles code generation for the DSPs, including partitioning for parallel execution, and hardware/software cosimulation. (Application developed by Asawaree Kalavade).

a number of externally developed tools, including Matlab (from The MathWorks, Inc.), Hyper (a VLSI hardware synthesis tool from Berkeley), Thor (an RTL-level circuit simulator from Stanford), and sim56000 and sim96000 (instruction-set simulators from Motorola for their programmable DSPs).

2. Research problems

Key research problems being addressed by this broad project include:

- Domain-specific signal processing algorithm design.
- Hardware/software codesign.
- Dataflow semantics (for signal processing).
- Hierarchical finite-state machine semantics (for control).
- Synthesis of embedded software.

2.1. Algorithm design

Quickly and easily evaluating signal processing algorithms requires access to a variety of tools. Matlab, for

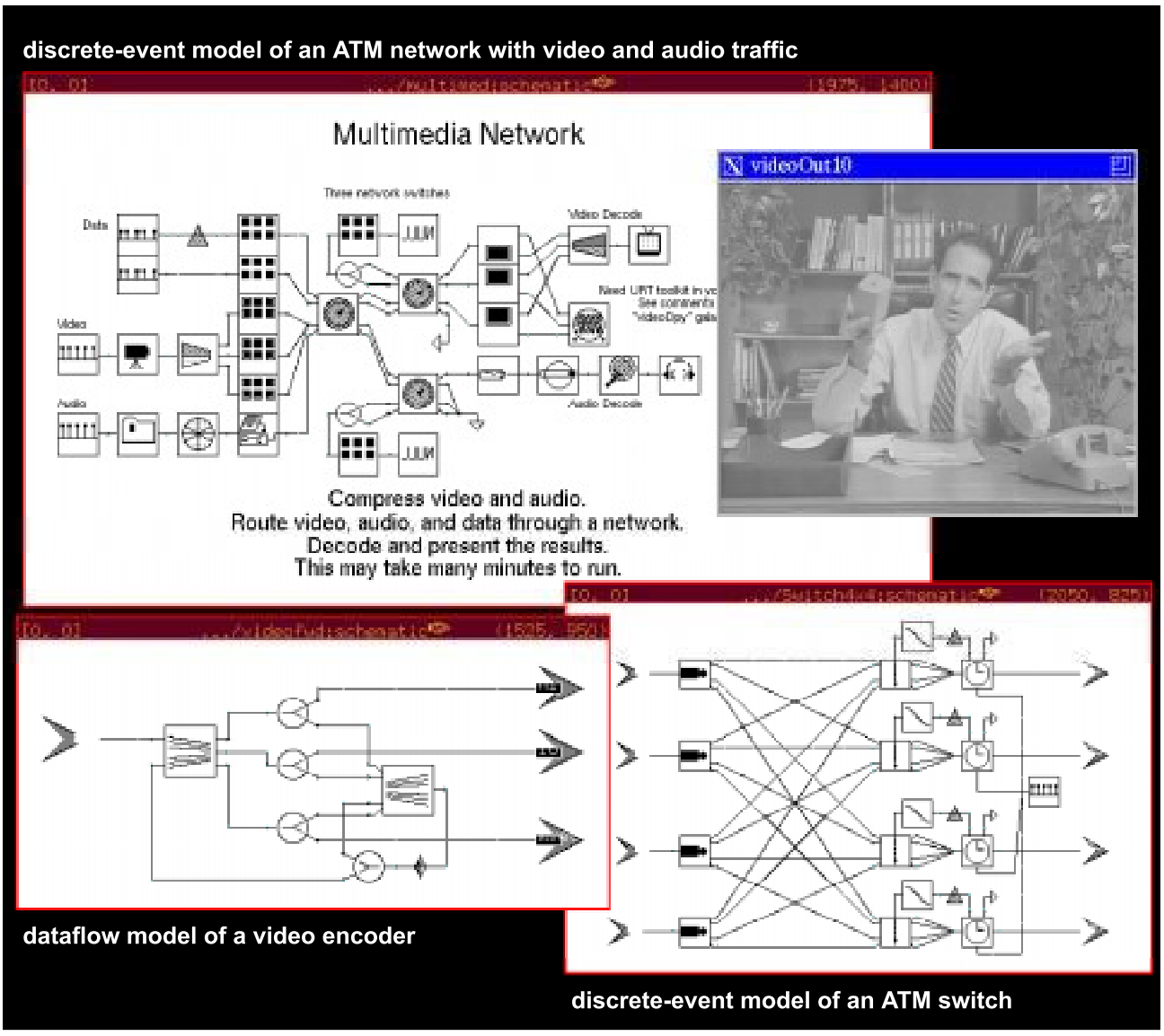

Figure 2. A multimedia networking application that combines discrete-event modeling of an ATM (asynchronous transfer mode) packet-switched network with video and audio coding and decoding (application developed by Paul Haskell).

example, can be used for rapid prototyping of complex algorithms; Mathematica can be used for symbolic manipulation of mathematical expressions; filter design programs are obviously used for filter design; and block-diagram systems can be used to assemble systems from simpler subsystems. These tools do not compete with one another. They complement one another, and in large designs involving many engineers, all could come into play.

2.1.1 Matlab integration

Recently, we have built a link to Matlab in Ptolemy that allows the functionality of a dataflow actor in a block diagram to be defined using Matlab. This permits designers to quickly evaluate algorithm alternatives, and to work together despite disagreements about the tools they use. Matlab uses one computation model based on matrix-vector calculations. The Ptolemy interface to Matlab allows Matlab functions to operate on Ptolemy matrices.

2.1.2 Matrix manipulations

A second aspect of the effort to raise the level of algorithm representation in Ptolemy is the inclusion of a comprehensive matrix manipulation mechanism in C++, cleanly coupled to the block diagram representation. Figure 3 shows one application of this matrix class. Here, the multiple signal classification (MUSIC) algorithm is being implemented for identifying sinusoids in noise. This is a very high-level representation of the algorithm, much akin

Name	Expansion	Principal Use
SDF	synchronous dataflow	synchronous signal processing
DDF	dynamic dataflow	asynchronous signal processing
BDF	boolean dataflow	asynchronous signal processing
MDSDF	multidimensional dataflow	multidimensional signal processing
DE	discrete event	communication network modeling and determinate high-level system modeling
FSM	finite state machines	control
HOF	higher-order functions	graphical programming
Thor	(name given at Stanford)	RTL hardware simulation
MQ	message queue	telecommunications switching software
PN	process networks	real-time systems
CP	communicating processes	communication network modeling nondeterminate system modeling
CGC	code generation - C	software synthesis (SDF or BDF model)
CG56	code generation - DSP56000	firmware synthesis (SDF model)
CG96	code generation - DSP96000	firmware synthesis (SDF model)
Silage	a functional language	VLSI hardware synthesis (SDF model)
VHDLF	VHDL - functional	high-level modeling and design (SDF)
VHDLB	VHDL - behavioral	hardware modeling and design (DE)
Sproc	a multiprocessor DSP from Star Semiconductor	firmware synthesis (SDF)

Table 1: Some domains that have been implemented in Ptolemy. The shaded area indicates simulation domains. The rest are code-generation domains.

to the mathematical representations used in such programs as Matlab. But unlike using Matlab, the designer has much more control over the details of the implementation of the algorithm. Thus, this facility provides a beginning point for a migration from high-level algorithm exploration to implementation and deployment.

2.2. Animation and visualization

An important part of algorithm design is to visualize the dynamics of a signal processing system using interactive, animated graphics. We have linked the interpreted language Tcl [4] and its associated X window toolkit Tk [5] with the Ptolemy system. This provides a powerful, extensible environment within which users can construct customized, animated, interactive simulations. In figure 4, we show an example of how this is used.

2.3. Visual representation of complex systems

Visual representation of complex systems poses some unique challenges. Some of these can be effectively addressed through the use of *higher-order functions*, which permit compact, scalable representations. A higher-order function is simply a function that takes a function as an argument and/or returns a function. A well-known example is the *mapcar* function in Lisp, which applies a function to each element of a list.

In the visual languages commonly used for signal processing, functions are replaced by blocks in a block diagram. Each block has two syntactically distinct types of arguments, *parameters* and *signals*. Signals are streams, and their values are only known at execution time. Parameters have values known when the execution of the system is being set up. It turns out that allowing parameters to specify blocks makes a visual language much more convenient for large applications.

A simple example of such a higher-order block in Ptolemy is shown in figure 5a. The block is called "Map," and it has a parameter that specifies a *replacement block*. At setup time, the Map block creates one or more instances of the replacement block and substitutes these replacement blocks for itself in the graph. The number of replacement blocks to instantiate is sufficient to process all input streams. Thus, for example, in figure 5c, there are three input streams, so if the replacement block has one input, then three instances will be created.

A variant of the Map block is shown in figure 5b, where the terminals at the top of this rather intricate icon provide a dock for an example of the replacement block. Thus, the replacement block is specified graphically rather than textually. Variations on this can have no inputs or no outputs. Such a variation is used in figure 5c, where three instances of the TkText block are specified graphically.

Another variant of the Map block is the *IfThenElse*, which has two possible replacement blocks, and selects

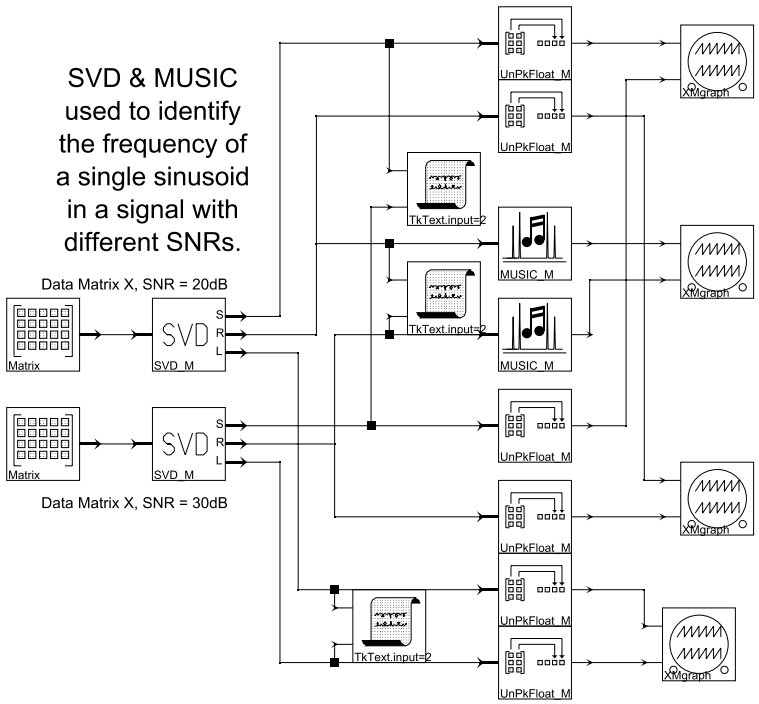

Figure 3. An application of the Matrix class in Ptolemy, used for high-level representation of algorithms.

Figure 4. Ptolemy has been linked to Tcl/Tk, an interpreted language and X window toolkit. This permits users to design customized, interactive, animated simulations of algorithms. The example shown here shows the taps of an adaptive filter at the upper right as the filter adapts. The user controls the noise and adaptation step size.

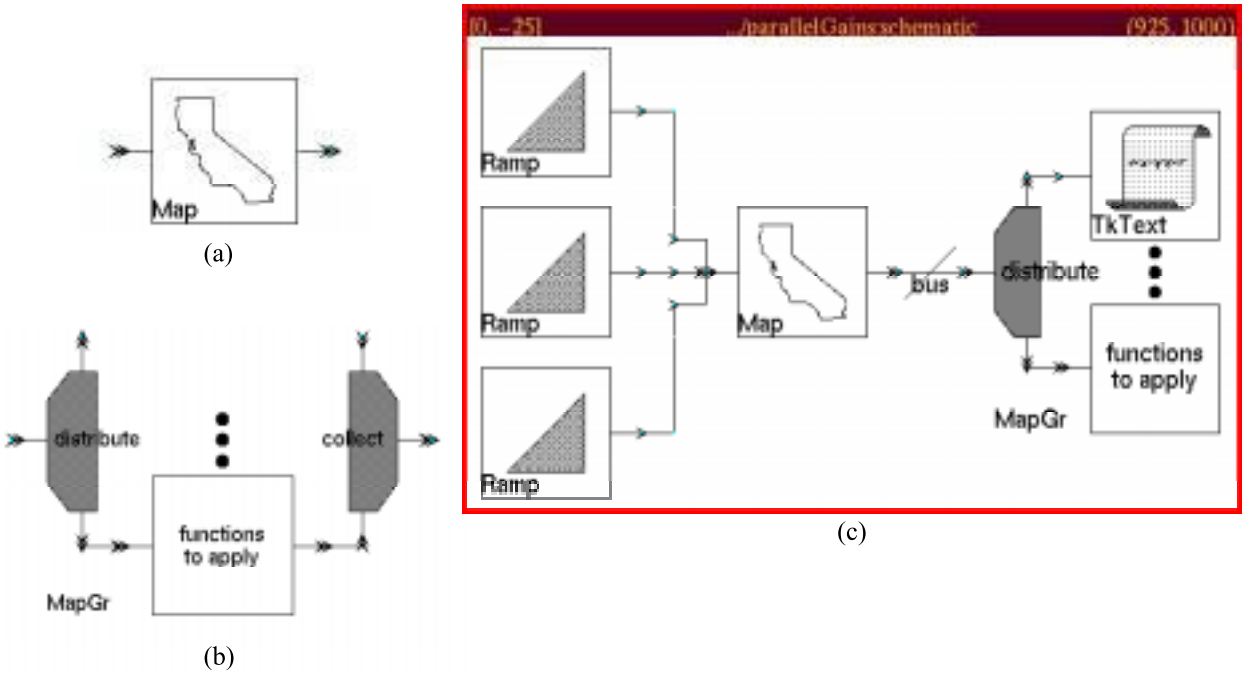

Figure 5. (a) An icon for a simple higher-order function in Ptolemy, Map, that applies a named actor to its input streams. (b) A variant of Map where the actor to apply to the input streams is specified graphically instead of textually. (c) A simple use of the two types of Map actors where three ramps (increasing sequences) have three instances of a named actor applied to them, and the three resulting streams are displayed using TkText.

between them using a predicate parameter. This can be used to implement recursion, as shown in figure 6. The first actor is a *distributor*, which collects two samples each time it fires, routing the first one to the top output and the second one to the lower output. The recursive invocation of this block accomplishes the decimation in time. The outputs of the distributor are connected to two *IfThenElse* blocks, represent one of two possible replacement subsystems. When the order parameter is larger than some threshold, the *IfThenElse* block replaces itself with a recursive reference to the galaxy within which it sits. When it gets below some threshold, then the *IfThenElse* block replaces itself with some direct implementation of a small order FFT. The *repeat* block takes into account the periodicity of the DFTs of order $N/2$ without duplicating the computation. The *expgen* block at the bottom simply generates the W_N^k sequence. The sequence might be precomputed, or computed on the fly.

A key observation about these higher-order blocks is that their substitution of the replacement blocks occurs at setup time not at runtime. Thus, they have *zero* run-time overhead. The recursive specification in figure 6 can be used even when extremely high performance is required and traditional implementations of recursion would be unacceptable.

2.4. Synchronous and dynamic dataflow

Our most heavily used model of computation for signal processing is synchronous dataflow (SDF) [1][2]. In this model, an application is described as a graph where nodes represent computations ("actors") and arcs represent the flow of data ("streams"). The actors produce and consume a fixed and known amount of data on each arc each time they fire. The synchronous dataflow model has the compelling advantage that the firing pattern of the actors can be completely determined at compile time.

Algorithms with predictable control flow have been successfully addressed using the synchronous dataflow (SDF) model of computation. Recently, however, our effort has broadened to include applications where control flow is not predictable. The objective is to preserve the benefits (especially efficiency) of predictable control flow whenever possible, but to support dynamic decision making, dynamic real-time response, and asynchrony. This will broaden the application domain to include telecommunications systems, real-time control, and hardware and software co-design. To do this, we are pursuing two lines of inquiry that avoid discarding the SDF model of computation in favor of one that is more general. The first is to mix models of computations, gaining generality through heterogeneity. The Ptolemy system is focused on supporting this. The second is to extend the analytical techniques of SDF to dynamic dataflow graphs. A *token flow model* [3][28] has been devised that replaces numeric solutions to the balance equations used in the SDF model with symbolic solutions. The dependence of control-flow on Booleans is represented symbolically.

2.5. Heterogeneous targets

We have recently demonstrated the capability in Ptolemy to jointly synthesize code for a host workstation and an attached signal processor. Thus, a real-time pro-

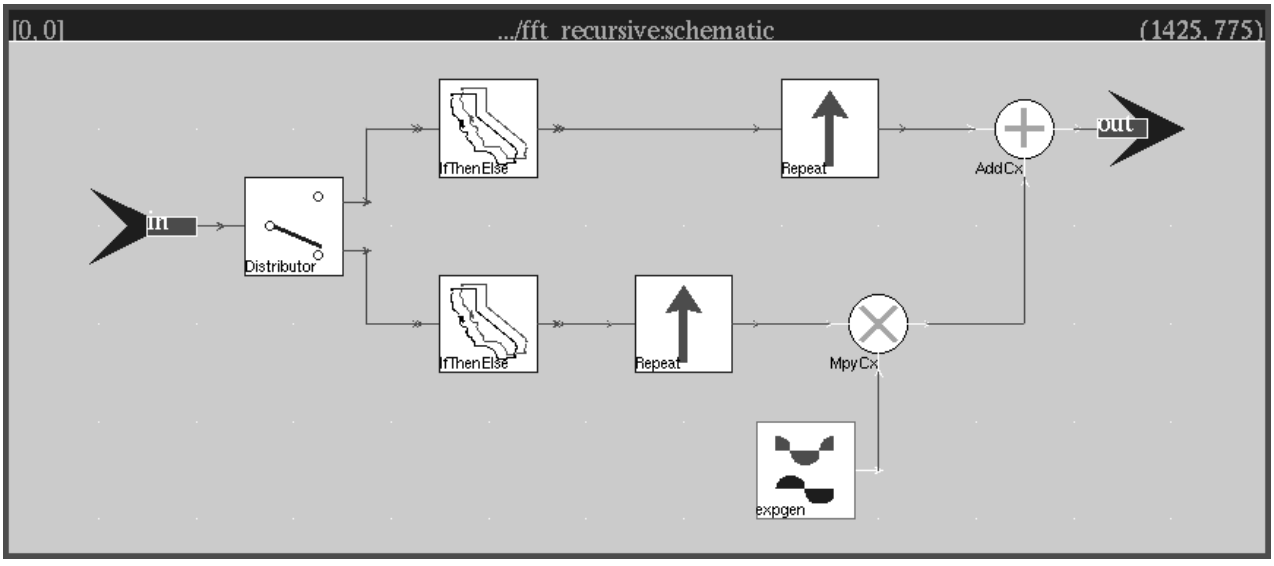

Figure 6. A recursive specification of an FFT implemented in the SDF domain in Ptolemy. The recursion is unfolded during the setup phase of the execution, so that the graph can be completely scheduled at compile time.

gram running on the attached hardware appears to the user as part of the process running on the workstation. This makes the real-time hardware transparently accessible, greatly enhancing our ability to rapidly prototype systems. Moreover, the methodology being developed to support this is generic, in that it can support joint software synthesis for a variety of heterogeneous processor configurations.

2.6. Real-time control

The dataflow capabilities in Ptolemy have advanced much further than other models of computation useful for prototyping. We are developing multithreaded dataflow and hierarchical finite state machine models for mixing real-time control with signal processing.

2.7. Optimized code generation

Synthesizing efficient assembly code for programmable DSPs has proved to be a rich area for innovation. We are currently focusing on problems associated with multirate systems that have radically different sample rates in different parts of the system, or with difficult sample-rate ratios.

3. Conclusions

In summary, the key idea in the Ptolemy project is to mix models of computation, rather than trying to develop one, all-encompassing model. The rationale is that specialized models of computation are (1) more useful to the system-level designer, and (2) more amenable to high-quality high-level synthesis of hardware and software. The Ptolemy kernel demonstrates one way to mix tools that have fundamentally different semantics, and provides a laboratory for experimenting with such mixtures.

The latest version of Ptolemy (designated 0.5[1]) has been publicly available and freely redistributable since February, 1994. More information about the Ptolemy project, plus access to all of the software and documentation, is available on the World Wide Web via the universal resource locator (URL) "http://ptolemy.eecs.berkeley.edu".

4. References

[1] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow," *IEEE Proceedings*, September, 1987.

[2] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE Transactions on Computers*, January, 1987.

[3] E. A. Lee, "Consistency in Dataflow Graphs", *IEEE Transactions on Parallel and Distributed Systems,* Vol. 2, No. 2, April 1991.

[4] J. Ousterhout, "Tcl: An Embeddable Command Language," *USENIX Conference Proceedings*, Winter, 1990.

[5] J. Ousterhout, "An X11 Toolkit Based on the Tcl Language," *USENIX Conference Proceedings*, Winter, 1991.

1. Since Ptolemy is research software, distributed free of charge and without support, all versions distributed by the University are designated 0.x.

5. Bibliography of recent publications

[6] "An Overview of the Ptolemy Project," Unpublished Memorandum, The Ptolemy Group, Department of EECS, University of California, Berkeley, March 7, 1994.

[7] "The Almagest — Ptolemy 0.5 Manual," *ibid*, March 17, 1994 (four volumes).

[8] M. J. Chen and E. A. Lee, "Design and Implementation of a Multidimensional Synchronous Dataflow Environment," Invited Paper, to appear in *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994.

[9] S. Sriram and E. A. Lee, "Static Scheduling of Communication Resources in Multiprocessor DSP Architectures," Invited Paper, to appear in *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994.

[10] P. K. Murthy and E. A. Lee, "Optimal Blocking Factors for Blocked, Non-Overlapped Multiprocessor Schedules", Invited Paper, to appear in *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994.

[11] A. Kalavade and E. A. Lee, "A Global Criticality / Local Phase based Algorithm for the Constrained Hardware/Software Partitioning Problem," to appear in *Codes/CASHE 94*, *Third International Workshop on Hardware/Software Codesign*, Grenoble, France, Sept. 22-24, 1994.

[12] A. Kalavade and E. A. Lee, "Manifestations of Heterogeneity in Hardware/Software Codesign", *Proc. of Design Automation Conference*, San Diego, CA, June, 1994.

[13] S. Bhattacharyya and E. A. Lee, "Scheduling Synchronous Dataflow Graphs for Efficient Looping", Journal of VLSI Signal Processing, vol. 6, pp. 271-288, June, 1994.

[14] S.-I. Shih, "Code Generation for a Video Signal Processor (VSP) Software Tool in Ptolemy", **MS Report**, Plan II, ERL Technical Report UCB/ERL M94/41, University of California, Berkeley, CA 94720, May 25, 1994.

[15] M. J. Chen, "Developing a Multidimensional Synchronous Dataflow Domain in Ptolemy", **MS Report**, ERL Technical Report UCB/ERL No. 94/16, University of California, Berkeley, CA 94720, May 6, 1994.

[16] E. A. Lee, "Signal Processing Experiments using Ptolemy — Instructor's Manual," Unpublished Memorandum,

Department of EECS, University of California, Berkeley, May, 1994.

[17] E. A. Lee, "Computing and Signal Processing: An Experimental Multidisciplinary Course", *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. VI, pp. 45-48, Adelaide, Australia, April, 1994.

[18] P. Murthy, S. Bhattacharyya, and E. A. Lee, "Minimizing Memory Requirements For Chain-Structured Synchronous Dataflow Programs," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. II, pp. 453-456, Adelaide, Australia, April, 1994.

[19] J. L. Pino, T. M. Parks, and E. A. Lee, "Automatic Code Generation for Heterogeneous Multiprocessors," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. II, pp. 445-448, Adelaide, Australia, April, 1994.

[20] J. Teich, S. Sriram, L. Thiele, and M. Martin, "Performance Analysis of Mixed Asynchronous-Synchronous Systems", *to appear in the Workshop on VLSI Signal Processing, 1994*, March 30, 1994.

[21] A. Lao, "Heterogeneous Cell-Relay Network Simulation and Performance Analysis with Ptolemy," Memorandum No. UCB/ERL M94/8, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, February 17, 1994.

[22] S. Bhattacharrya and E. A. Lee, "Looped Schedules for Dataflow Descriptions of Multirate Signal Processing Algorithms," **to appear** in *Formal Methods in System Design,* (updated from UCB/ERL Technical Report, May 21, 1993).

[23] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, "Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems", *International Journal of Computer Simulation*, special issue on "Simulation Software Development," Vol. 4, pp. 155-182, April, 1994.

[24] S. Bhattacharyya and E. A. Lee, "Memory Management for Synchronous Dataflow Programs," *IEEE Tr. on Signal Processing*, Vol. 42, No. 5, May 1994. Updated from: Technical Report UCB/ERL M92/128, EECS Dept., University of California, Berkeley, CA 94720, November 18, 1992.

[25] J. Pino, S. Ha, E. Lee, J. Buck, "Software Synthesis for DSP Using Ptolemy", invited paper in the *Journal on VLSI Signal Processing*, special issue on "Synthesis for DSP", **to appear**, 1994.

[26] S. Bhattacharyya and E. A. Lee, "Scheduling Synchronous Dataflow Graphs for Efficient Looping," *J. of VLSI Signal Processing,* Vol. 6, December 1993.

[27] S. Bhattacharyya, J. T. Buck, S. Ha, and E. A. Lee, "A Scheduling Framework for Minimizing Memory Requirements of Multirate DSP Systems Represented as Dataflow Graphs," in *VLSI Signal Processing VI*, IEEE Special Publications, New York, 1993.

[28] J. T. Buck, *Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model,* Tech. Report UCB/ERL 93/69, **Ph.D. Dissertation**, Dept. of EECS, University of California, Berkeley, CA 94720, 1993.

[29] A. Kalavade, and E. A. Lee, "A Hardware/Software Codesign Methodology for DSP Applications," *IEEE Design and Test*, September 1993.