Presented at ICASSP-94 - Adelaide, Australia

AUTOMATIC CODE GENERATION FOR HETEROGENEOUS MULTIPROCESSORS[1]

José L. Pino, Thomas M. Parks and Edward A. Lee

EECS Department, University of California, Berkeley CA 94720
{pino,parks,eal}@EECS.Berkeley.EDU

ABSTRACT

This paper describes the use of Ptolemy to automatically generate code for heterogeneous multiprocessor systems. The framework presented lets the designer migrate from simulation to code generation while developing an application that is specified by constructing a dataflow graph. From primitive send and receive actors, the framework can automatically construct three classes of interprocessor communication (IPC) interfaces. The first type of interface uses a synchronous dataflow (SDF) parallel scheduler to partition and schedule the graph across the available processors. The second type of interface allows hierarchical use of cooperating schedulers within an application. Finally, the third interface uses the send and receive actors as an interface between code generation and simulation systems in Ptolemy. To illustrate the framework, we present an example of a heterogeneous architecture consisting of a workstation with multiple Motorola 56001 processors. We present the relative strengths and weaknesses of each type of interface.

INTRODUCTION

Dataflow is a natural representation for signal processing algorithms. One of its strengths is that it exposes parallelism by only expressing the actual data dependencies that exist in an algorithm. Applications are specified by a dataflow graph where the nodes represent computational actors, and data tokens flow between them along the arcs of the graph. Ptolemy [1] is a framework that supports dataflow programming as well as other computational models (such as discrete event), in which program graphs have different semantics.

Code generation consists of two phases, scheduling and synthesis. In the scheduling phase, the dataflow graph is partitioned for parallel execution. We splice send and receive actors into the graph for interprocessor communication. These actors do the synchronization necessary for a self-timed implementation [2]. For each target processor, an ordering of actor invocations is determined. In the synthesis phase, the code segments associated with each actor are stitched together, following the ordering specified by the scheduler. Commercial systems which use this “threading” technique include Comdisco’s DPC [3] and CADIS’s Descartes [4]. The techniques we describe here are complementary to those in DPC and Descartes, and could, in principle, be used in combination with them. In particular, we focus on management of data passed between actors when synchronous dataflow is used. DPC, by contrast, does not use dataflow semantics.

There are several forms of dataflow defined in Ptolemy. In synchronous dataflow (SDF) [5], the number of tokens produced or consumed in one firing of an actor is constant. This property makes it possible to determine execution order and memory requirements at compile time. Thus these systems do not have the overhead of run-time scheduling (in contrast to dynamic dataflow) and have very predictable run-time behavior. Some of the many SDF scheduling algorithms implemented in Ptolemy include: uniprocessor list scheduling [5], uniprocessor loop scheduling [6], Hu-level multiprocessor scheduling, and Sih’s declustering multiprocessor scheduling [7].

Although SDF is sufficient to describe many signal processing algorithms, it cannot express data dependent control flow. A dataflow model which does allow this is boolean dataflow (BDF). This model subsumes SDF with two additional actors: switch and select. These actors allow the construction of data-dependent control flow structures such as do-while and if-then-else. Buck [8] shows how most of these graphs can be scheduled with bounded memory. Such graphs allow some dynamic dataflow expression with minimal run time overhead. Unfortunately, at the present time this scheduling works only for uniprocessor systems.

In this paper, we present three IPC interfaces that are constructed automatically from send and receive actors. The first interface uses send and receive actors to implement the IPC specified by an SDF parallel scheduler. In this configuration, we generate a stand-alone application where the entire dataflow graph is scheduled as a whole. The second interface, known as the CG Wormhole, also generates a stand-alone application. However, this interface isolates subsystems of the graph, using distinct schedulers on each side of the interface. CGWormholes are similar to the Ptolemy *Wormhole* construct, which is defined to be the interface between two different models of computation. CGWormholes, on the other hand, interface subgraphs which typically obey the same model of computation but are executed on different processors. The third type of interface is a Wormhole between code generation and simulation. We call this type of interface a CG-Sim-Wormhole. One of its primary uses is to embed actual hardware in high level simulations. The

1. This work was supported by grants from NSF (MIP 9201605), AT&T, ONR, and SRC (grant number 92-DC-008).

top-level simulation can any of the computation models available in Ptolemy, such as dynamic dataflow, discrete-event, or communicating processes.

We illustrate the use of these interfaces with a heterogeneous target that consists of a workstation in combination with several digital signal processor (DSP) cards: Ariel S-56X cards installed on the SBus of a SPARC workstation. Each DSP card has a single Motorola 56001 digital signal processor, a Xilinx programmable logic cell array, serial ports, and a DMA port. In this target, the workstation serves as the interface between the DSP cards and the user, the network, and other resources.

TARGET SPECIFICATION

A key property of Ptolemy that makes specification of heterogeneous targets easier is its use of object-oriented programming techniques. In describing a multiprocessor target, we begin with the specification of each individual processor and build multiprocessor targets hierarchically from these objects. A target specification in Ptolemy manages the flow of the design process; i.e., it defines the methods to schedule the graph, compile and run the generated code, taking into account the target resources. A detailed description of the code generation framework in Ptolemy can be found in [9] and with emphasis on single processor targets in [10].

The fundamental building block of a multiprocessor target is a single processor target. In our example, we have multiple Ariel S-56X cards installed in a workstation. An S56X target describes one of the DSP cards. This target knows the exact memory resources available on the card and how download the code into the program memory of the DSP. The S56X target generates assembly code and allocates target-specific resources such as private memory. A CGC target describes the resources of the workstation and generates code in the C programming language. This target is a more general type of target than the S56X target. The code it generates can run on most general-purpose computers.

A multiprocessor target is built from other targets that it contains as children in a hierarchy. These children can be any type of target, from a simple single processor to a complex heterogeneous multiprocessor. The parent multiprocessor target specifies the shared resources and IPC mechanisms of the children. A homogeneous multi-DSP target description is detailed in [11].

The CGC-S56X heterogeneous target is a multiprocessor target containing multiple S56X target children together with a CGC target. The S56X target was used to implement a real-time ADPCM speech coder described in [10]. Simpler examples are provided below to illustrate key features of the different interfaces.

BASIC INTERFACE

When a dataflow graph is partitioned by an SDF multiprocessor scheduler, pairs of send and receive actors are automatically spliced into the graph wherever IPC occurs. Hence, the location of the splicing is typically determined by the scheduler and not the user. For multiprocessor targets, scheduling is performed at the top level; the children are provided with the resultant schedule for their processor(s). Any user-specified hierarchy is ignored in order to fully exploit the parallelism that exists in the graph.

However, there are disadvantages with using one of the presently available SDF parallel schedulers. In many practical situations, such as with heterogeneous targets, the user knows a reasonable actor partitioning over the processors. Parallel schedulers are hard to implement and do not perform many of the optimizations that are available with uniprocessor schedulers, such as code compaction[6] and buffer minimization[12]. Furthermore, schedulers such as BDF do not support multiprocessor platforms, but are very attractive to express data-dependent control flow with low run-time overhead.

INTER-CG INTERFACE

To allow the use of heterogenous schedulers in an application, we implement something similar to the Wormhole construct in Ptolemy. The use of Wormholes in a simulation context is detailed in [1]. Here we use it as an interface between schedulers in a code generation context. For example, in our heterogeneous target we could use a BDF scheduler (which works only for a uniprocessor) on the control processor (workstation) and a loop scheduler on the DSP card(s). The user can choose the most appropriate scheduler for each processor.

A CGWormhole is constructed whenever there is a scheduler change from one level of the user-specified graph hierarchy to another, as in figure 1. From the outside, the CGWormhole appears to be a monolithic actor. This actor is constructed from send and receive

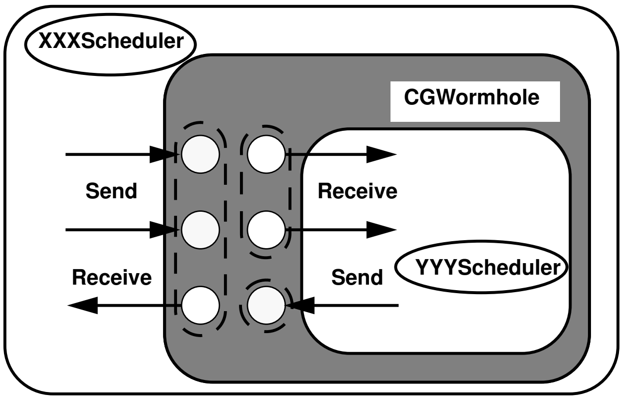

Figure 1. The CGWormhole Construct

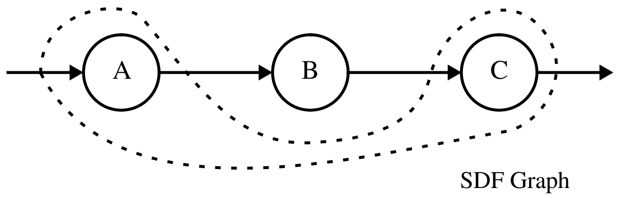

Figure 2. Potential deadlock Condition

actors for each arc leading into/out of the Wormhole. The inside interface is constructed by splicing two actors into the inner graph: a data source for all the receive actors and a sink for all the send actors.

The strength of this type of interface is clear: we are able to use the most appropriate scheduler for any part of the application. This advantage, however, comes at the possible loss of exploitable parallelism. Another potential problem is that we could introduce deadlock as shown in figure 2. In this figure, the dotted lines represent an arbitrary user-specified boundary. Here, the resultant top-level graph would be deadlocked, even though the original graph was not. This condition is returned to the user as an error at the time of scheduling.

CG-SIMULATION INTERFACE

The last type of interface is between code generation and simulation in the Ptolemy environment. The code generation environment synthesizes and compiles code from a user-specified graph. For simulation, the graph is interpreted within Ptolemy. This interface is similar to the previous one in that a CGWormhole is constructed; however, in this case we must use the EventHorizon interface with the Wormhole to transfer tokens to Ptolemy. The EventHorizon interface is shown in figure 3 and is described in detail in [1].

For this interface we restrict the outside system to be a simulation. To construct the code generation side, we splice in CGC actors that are designed to communicate directly to the EventHorizon. If the inside target is CGC,

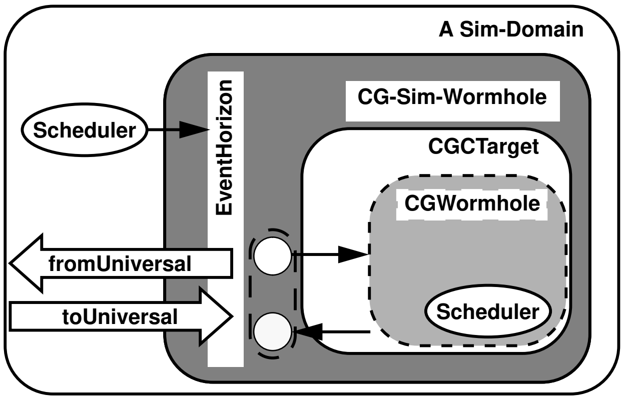

Figure 3. Interface to Simulation: EventHorizon

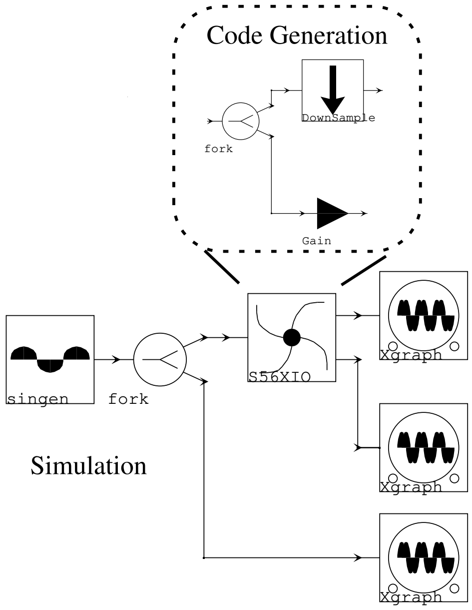

Figure 4. CG-Sim Multirate Wormhole

the interface is fully specified. However, if it is not CGC, a CGWormhole is constructed between CGC and the intended target, as detailed in the previous section. Finally, the C code for this interface is dynamically linked into the Ptolemy binary.

As an example, we use the CG-Sim-Wormhole as a simple test/monitoring system for the DSP card. This example demonstrates multiple streams of data flowing between the workstation and DSP at different rates. Figure 4 shows the hierarchical graph detailing the example. The S56XIO block, which appears expanded above the main graph, is executed on the DSP card installed in the workstation. All other blocks run in a Ptolemy SDF simulation. Externally, the S56XIO block appears to be an SDF simulation block; however, before the simulation is run, code is automatically generated and downloaded to the DSP card.

When data reaches the input of the S56XIO block, it is downloaded to the DSP card. Here it is forked; on one branch it is downsampled by a ratio of 2:1 and on the other it is multiplied by a gain of 0.5. The results are then uploaded to the workstation for display using XGraph blocks.

A primary motivation for the construction of this interface is to embed an actual hardware implementation of an algorithm into a high-level simulation. Other uses of this CG-Sim-Wormhole include hardware acceleration, algorithm development/migration and debugging. This work is similar to the hardware and simulation interface presented in [13]. The major distinction is that this interface is one of a trio of interfaces all built automatically from the same send and receive actors.

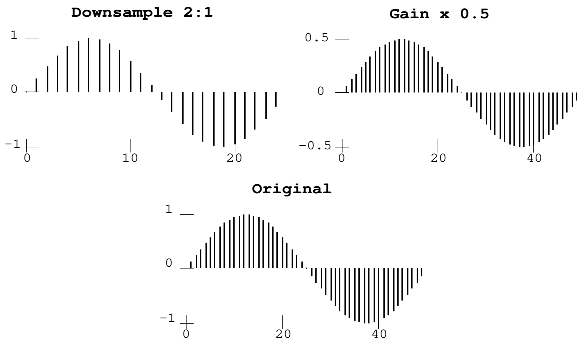

Figure 5. Multirate Sine Output

CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an extensible framework to generate code for heterogeneous systems. This environment promotes code reuse in the form of actor and target libraries. The send and receive actors defined for a particular target can be used to construct any of the interfaces described. The first type of interface is one in which pairs of send and receive actors are spliced into the graph as specified by an SDF parallel scheduler. Allowing the scheduler to fully partition the graph exposes maximal parallelism and will not introduce deadlock. However, this configuration is inappropriate in systems with high sample rate changes. The next type of interface, a CGWormhole, mixes scheduling algorithms in a code generation application. For example, in a multiprocessor system where we would like to use dynamic constructs with minimal run-time over head, we could specify that the BDF scheduler be used for a particular subgraph and an SDF parallel scheduler for the remainder of the graph. The last type of interface allows code generation applications to communicate with simulations in Ptolemy. With this interface, an actual hardware implementation can be used within a high level simulation. These interfaces can be used in conjunction with one another. For example, a uniprocessor boolean dataflow scheduler could be used on the outside of a CGWormhole, and a multiprocessor SDF scheduler could be used on the inside to parallelize the inner subgraph. The major focus for future work will be in deciding how to partition the processors for multiple CGWormholes.

REFERENCES

[1] J. Buck, S. Ha, E.A. Lee, and D.G. Messerschmitt, “Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems,” *International Journal of Computer Simulation*, special issue on Simulation Software Development, to appear 1994.

[2] E.A. Lee and S. Ha, “Scheduling strategies for multiprocessor real-time DSP.,” *GLOBECOM '89*, 1989, p. 1279-83 vol.2.

[3] D.G. Powell, E. A.Lee, and W.C. Newman, “Direct Synthesis of Optimized DSP Assembly Code from Signal Flow Block Diagrams,” *ICASSP*, vol. 5, San Francisco, CA, IEEE, 1992, p. 553-556.

[4] S. Ritz, M. Pankert, and H. Meyr, “High level software synthesis for signal processing systems,” *Proceedings of the International Conference on Application Specific Array Processors*, Berkeley, CA, USA, IEEE Comput. Soc. Press, 1992, p. 679-693.

[5] E.A. Lee and D.G. Messerschmitt, “Synchronous data flow,” *Proceedings of the IEEE*, vol. 75, no. 9, 1987, p. 1235-1245.

[6] S.S. Bhattacharyya, *Scheduling synchronous dataflow graphs for efficient iteration*, Master's Thesis, University of California at Berkeley, 1991.

[7] G.C. Sih and E.A. Lee, “Declustering: A New Multiprocessor Scheduling Technique,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 6, 1993, p. 625-637.

[8] J.T. Buck and E.A. Lee, “Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model,” *ICASSP*, Minneapolis, IEEE, 1993.

[9] J.L. Pino, S. Ha, E.A. Lee, and J.T. Buck, “Software Synthesis for DSP Using Ptolemy,” *Journal of VLSI Signal Processing*, Synthesis for DSP, 1993, to appear.

[10] J.L. Pino, *Software Synthesis for Single-Processor DSP Systems Using Ptolemy*, Master's Thesis Memorandum UCB/ERL M93/35, University of California at Berkeley, 1993.

[11] S. Sriram and E.A. Lee, “Design and Implementation of an Ordered Memory Access Architecture,” *ICASSP*, Minneapolis, MN, IEEE, 1993.

[12] P.K. Murthy, S.S. Bhattacharyya, and E.A. Lee, “Minimizing Memory Requirements for Chain-Structured Synchronous Dataflow Programs,” *ICASSP*, Adelaide, South Australia, 1994.

[13] M. Pankert, S. Ritz, and H. Meyr, “Integration of digital signal processing hardware into a system level simulation environment,” *Proceedings of the European Simulation Multiconference*, York, U.K., 1992, p. 147-151.