Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Detroit, MI, May 1995

HIERARCHICAL STATIC SCHEDULING OF DATAFLOW GRAPHS ONTO MULTIPLE PROCESSORS

José Luis Pino and Edward A. Lee

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

{pino,eal}@EECS.Berkeley.EDU

ABSTRACT

In this paper we discuss a hierarchical scheduling framework to reduce the complexity of scheduling synchronous dataflow (SDF) graphs onto multiple processors. The core of this framework is a clustering technique that reduces the number of actors before expanding the SDF graph into an directed acyclic graph (DAG). The internals of the clusters are then scheduled with uniprocessor SDF schedulers which can optimize for memory usage. The clustering is done in such a manner as to leave ample parallelism exposed for the multiprocessor scheduler. We illustrate this framework with a real-time example that has been constructed in Ptolemy.

1. INTRODUCTION

Dataflow is a natural representation for signal processing algorithms. One of its strengths is that it exposes parallelism by expressing only the actual data dependencies that exist in an algorithm. Applications are specified by a dataflow graph in which 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).

Generating a stand-alone application from a dataflow graph description consists of two phases: scheduling and synthesis [2]. 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 [3]. For each target processor, a sequence of actor firings is determined. In the synthesis phase, the code segments associated with each actor are stitched together, following the order specified by the scheduler. Commercial systems that use this "threading" technique include Comdisco's DPC [4] and CADIS's Descartes [5]. The techniques we describe here are complementary to those in DPC and Descartes, and could, in principle, be used in combination with them.

There are several forms of dataflow defined in Ptolemy. In synchronous dataflow (SDF) [6], 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. The production/consumption property on the arcs also provides a natural representation of multirate signal processing blocks [7]. In this paper, we will focus on scheduling SDF graphs onto multiple processors.

In the following sections, we will review scheduling of SDF graphs including uniprocessor scheduling and DAG construction. Then we will discuss the clustering techniques that make up the hierarchical scheduling framework. Finally, we look at extensions to current DAG schedulers that use declustering in order to break larger grain clusters to expose parallelism. In these circumstances we are able to delay (and possibly avoid) the expansion of SDF sub-graphs into the overall DAG.

2. SYNCHRONOUS DATAFLOW

Figure 1 shows a simple multirate SDF graph. In this graph, actor *A* produces two tokens and actor *B* consumes three tokens for each firing. In a valid SDF schedule, the first-in/first-out (FIFO) buffers on each arc return to their initial state after one schedule period. Balance equations are written for each arc and an integral repetitions vector is found that solves this system of equations [6]. In this simple example, the balance equation for the arc is: $2 \times R_A = 3 \times R_B$. Thus the repetition vector is: $\left[R_A \; R_B\right] = \left[3n \; 2n\right], n \in Z^+$. One of the valid schedules

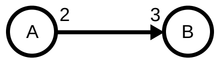

Figure 1. A simple SDF graph.

for this graph is *AABAB*. Given an SDF specification, we can find a schedule at compile-time that is iteratively repeated at run-time.

To schedule SDF graphs onto multiple processors, a directed acyclic graph (DAG) is constructed from the original SDF graph. The SDF graph exposes functional parallelism in the algorithm; the DAG in addition exposes the data parallelism available. The DAG for the graph of figure 1 is shown in figure 2. Notice that for each actor in the original SDF graph, there are multiple nodes in the DAG corresponding to the repetitions count derived from the balance equations.

Unfortunately, the expansion due to the repetition count of each SDF actor can lead to an exponential growth of nodes in the DAG. An SDF graph that exhibits this growth is shown in figure 3. The order of the number of nodes in the DAG is calculated to be $O(M^N)$. The growth is undesirable, especially considering that known optimal multiprocessor scheduling algorithms under precedence constraints have complexity that is exponential in the number of nodes in the DAG [8]. Most uniprocessor SDF schedulers, on the other hand, do not require a DAG to be generated for scheduling purposes.

To limit the explosion of nodes when translating an SDF graph into a DAG graph, we introduce clustering of connected subgraphs into larger grain "composite" actors. The composite actors will then be scheduled with one of the available uniprocessor schedulers. We cluster the actors in a manner that simplifies the DAG graph, without hiding much parallelism.

Many multiprocessor schedulers use clustering heuristics to schedule the DAG graph onto multiple processors [9-12]. It is important to note that each resultant cluster is mapped onto a single processor. The purpose of this paper is to introduce some simple clustering techniques that can be used directly on the SDF graph, thereby reducing the number of SDF nodes that are expanded in the final

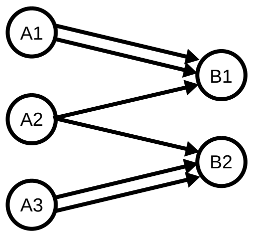

Figure 2. DAG for SDF graph in figure 1.

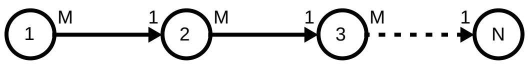

Figure 3. A SDF graph exhibiting $O(M^N)$ DAG nodes.

DAG graph. Clustering the SDF graph also gives us the opportunity to use specialized uniprocessor SDF schedulers. These uniprocessor schedulers can optimize for such parameters as code size and memory usage [13].

3. CLUSTERING TECHNIQUES

In this section we review our clustering techniques for SDF graphs. There are currently four clustering techniques: user specified, resource constraint limited, homogeneous SDF chain structured subgraphs, and multirate chain structured subgraphs for which all the actors have internal state.

The first clustering technique is by far the simplest: we allow the user to specify clusters that will be mapped onto a single processor. This clustering technique empowers the user with fundamental scheduling decisions. A potential problem is that the user can introduce artificial deadlock. However, this error is easily caught at compile time [14]. When we automatically cluster subgraphs, we must ensure that the constructed clusters do not introduce artificial deadlock. To do this we can apply the four cluster conditions listed in [13].

The next clustering technique takes into account resource constraints. When mapping SDF graphs onto heterogeneous processors, a group of connected actors may be required to be mapped onto a particular processor. Here, we are free to cluster these SDF subgraphs as long as we do not introduce artificial deadlock.

In the last two clustering techniques, we cluster chains of SDF actors. A chain structured SDF subgraph is one in which each actor is connected to at most two other actors. One source actor is connected to all the input arcs and one destination actor is connected to all the output arcs. Thus, when clustering chain structured SDF subgraphs, we do not group over *branch* or *merge* SDF actors (actors which have multiple sources or destinations), where functional parallelism is exposed.

The third clustering technique groups the actors in a homogeneous chain structured SDF subgraph where the actors do not have internal state (or equivalently, have self loop arcs). A homogeneous SDF subgraph is one in which

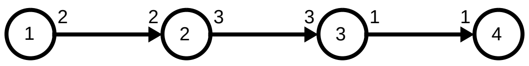

Figure 4. A homogeneous chain structured SDF graph

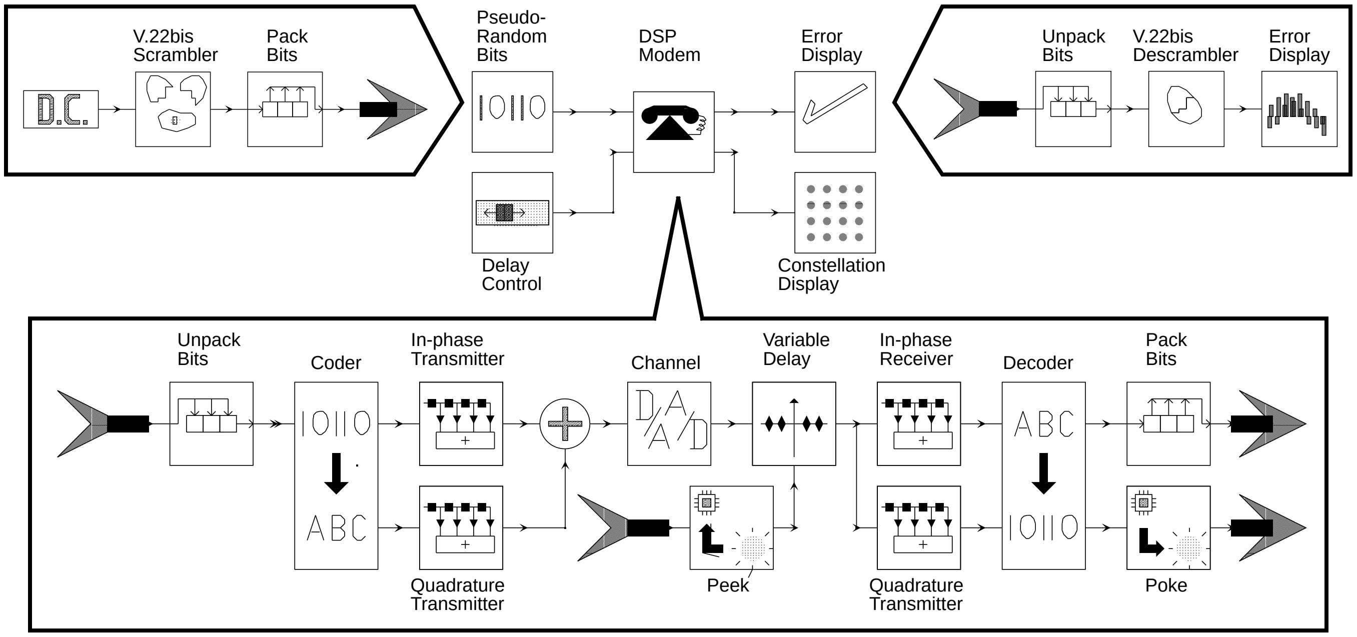

Figure 5. A QPSK acoustic modem. The top center block diagram is the top-level modem schematic. The hierarchy of *Pseudo-Random Bits*, *DSP Modem*, and *Error Display* blocks is expanded in the accompanying block diagrams. All of the blocks except for the *DSP Modem* execute on the host workstation. The *DSP Modem* executes on the Ariel S-56X DSP board.

the number of outputs produced on an arc is equal to the number of inputs consumed on that arc (see figure 3). This clustering does not hide any of the available parallelism that will be exposed in the final DAG. These clusters obey the linear clustering heuristic described in [11] for DAG multiprocessor scheduling.

Finally, the last clustering technique groups the actors in chain structured subgraphs made up of multirate actors having internal state. Internal state introduces dependencies between the various repetitions of an actor in a schedule period, thereby reducing the data parallelism available. For example, if the actors in the SDF graph shown in figure 1 had internal state, then the resultant DAG shown in figure 2 would have additional precedence arcs connecting *A1*→*A2*→*A3* and *B1*→*B2*. Unlike the previous three clustering heuristics, this technique can hide some of the available parallelism although in this case, it does not.

4. ACOUSTICAL MODEM EXAMPLE

In this section we detail a quadrature phase-shift keying(QPSK) acoustical model [15] which is scheduled onto two heterogeneous processors (RISC, DSP). The SDF specification is shown in figure 5. A pseudo-random bit stream is generated on the workstation and then packed into a DSP word stream(22 bits/word). The stream of words is sent to a DSP which unpacks each word to form a bit stream. These bits are then encoded into a symbol (2 bits/symbol). The DSP transmits and then receives the symbol stream over a speaker/microphone channel. The received symbols are then decoded, packed and sent back to the workstation, where the errors are displayed to the user. The user can control the alignment of the symbol period and examine the resultant constellation using the peek/poke mechanism described in [16]. All of the transmitter and receiver filters are polyphase FIR filters with interpolation and decimation factors of 32 samples respectively.

Note that the SDF graph shown in figure 5 is expressed hierarchically. There are a total of 38 SDF actors; the corresponding DAG has a total of 2100 nodes. Since we are able to use SDF uniprocessor schedulers on the SDF subgraph clusters, for this example, we are able to obtain a *single appearance schedule* which leads to very compact code. A single appearance schedule is an SDF schedule in which each actor only appears once [13]. To obtain the single appearance schedule, three uniprocessor schedulers and one multiprocessor scheduler were used by the hierarchical scheduling framework. By using the cluster hierarchy, the multiprocessor scheduler only had to schedule a DAG with 8 nodes. The multiprocessor schedule generated from the fully expanded DAG graph, has one function call (or inlined procedure) for each of its 2100 nodes as compared to the 38 function calls for the hierarchical schedule.

Finally, the makespans of the schedules generated using the hierarchical scheduler and traditional full DAG expansion varied by less than 4% (the hierarchical being longer). The difference is due to the fact that a cluster of SDF actors is treated as an atomic SDF actor by the outside scheduler. Hence all of the inputs must be available before the cluster fires. Furthermore, none of the outputs will be available until a cluster schedule iteration is completed. We are investigating cyclo-static dataflow [17] as a method to eliminate the variance in makespans. In cyclo-static dataflow, each actor firing has distinct phases. Each phase produces and consumes a fixed amount of data at each input and output. The cluster schedule could then be expressed in schedule phases, thereby allowing part of the cluster to fire as soon as some of the input was available (versus waiting for all of the input for all of the phases).

5. CONCLUSIONS

In this paper, we have introduced a hierarchical scheduling framework for SDF graphs being mapped onto multiple processors. This framework can drastically reduce the number of nodes present in the final DAG graph, which is used for parallel scheduling. We have shown a practical example where this scheduling technique has greatly improved the final schedule.

We plan to augment the SDF graph clustering techniques by specializing some of the DAG clustering heuristics found in multiprocessor schedulers for direct use on the SDF graph. The objective is to hide only that parallelism that would not be exploited anyway, and in doing so, simplifying the DAG.

ACKNOWLEDGMENTS

This research is part of the Ptolemy project, which is supported by the Advanced Research Projects Agency and the U.S. Air Force (under the RASSP program, contract F33615-93-C-1317), Semiconductor Research Corporation (project 94-DC-008), National Science Foundation (MIP-9201605), Office of Naval Technology (via Naval Research Laboratories), the State of California MICRO program, and the following companies: Bell Northern Research, Cadence, Dolby, Hitachi, Mentor Graphics, Mitsubishi, NEC, Pacific Bell, Philips, Rockwell, Sony, and Synopsys.

José Luis Pino is also supported by AT&T Bell Laboratories as part of the Cooperative Research Fellowship Program.

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*, vol. 4, pp. 155-182, 1994.

[2] J. L. Pino, S. Ha, E. A. Lee, and J. T. Buck, "Software synthesis for DSP using Ptolemy," *Journal of VLSI Signal Processing to appear in special issue on Synthesis for DSP*, 1993.

[3] E. A. Lee and S. Ha, "Scheduling strategies for multiprocessor real-time DSP," presented at IEEE Global Telecommunications Conference and Exhibition. Communications Technology for the 1990s and Beyond, Dallas, TX, USA, 1989.

[4] D. G. Powell, E. A.Lee, and W. C. Newman, "Direct synthesis of optimized DSP assembly code from signal flow block diagrams," presented at IEEE International Conference on Acoustics, Speech, and Signal Processing, San Francisco, CA, 1992.

[5] S. Ritz, M. Pankert, and H. Meyr, "High level software synthesis for signal processing systems," presented at International Conference on Application Specific Array Processors, Berkeley, CA, USA, 1992.

[6] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, pp. 1235-1245, 1987.

[7] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Multirate signal processing in Ptolemy," presented at IEEE International Conference on Acoustics, Speech, and Signal Processing, Toronto, Ont., Canada, 1991.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*. New York: W.H. Freeman, 1991.

[9] A. Gerasoulis and T. Yang, "A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 276-291, 1992.

[10] P. D. Hoang and J. M. Rabaey, "Scheduling of DSP programs onto multiprocessors for maximum throughput," *IEEE Transactions on Signal Processing*, vol. 41, pp. 2225-2235, 1993.

[11] S. J. Kim and J. C. Browne, "A general approach to mapping of parallel computations upon multiprocessor architectures," presented at International Conference on Parallel Processing, University Park, PA, USA, 1988.

[12] G. C. Sih and E. A. Lee, "Declustering: A new multiprocessor scheduling technique," *IEEE Transactions on Parallel and Distributed Systems*, 1992.

[13] S. S. Bhattacharyya, J. T. Buck, S. Ha, and E. A. Lee, "A compiler scheduling framework for minimizing memory requirements of multirate DSP systems represented as dataflow graphs," University of California at Berkeley UCB/ERL M93/31, April 25, 1993 1993.

[14] J. L. Pino, T. M. Parks, and E. A. Lee, "Automatic code generation for heterogeneous multiprocessors," presented at IEEE International Conference on Acoustics, Speech, and Signal Processing, Adelaide, South Australia, 1994.

[15] E. A. Lee and D. G. Messerschmitt, *Digital Communication*, 2nd ed. Boston: Kluwer Academic Publishers, 1994.

[16] J. L. Pino, T. M. Parks, and E. A. Lee, "Mapping multiple independent synchronous dataflow graphs onto heterogeneous multiprocessors," presented at IEEE Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, 1994.

[17] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Static scheduling of multi-rate and cyclo-static DSP-applications," presented at IEEE International Workshop on VLSI Signal Processing, La Jolla, California, 1994.