In Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers, pages 122-126 vol.1, Pacific Grove, California, November 1995.

A Hierarchical Multiprocessor Scheduling System for DSP Applications

José Luis Pino[†], Shuvra S. Bhattacharyya[‡] and Edward A. Lee[†]

{pino,shuvra,eal}@EECS.Berkeley.EDU

[†]Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley
[‡]Semiconductor Research Laboratory, Hitachi America, Ltd.

Abstract

This paper discusses a hierarchical scheduling framework which reduces the complexity of scheduling synchronous dataflow (SDF) graphs onto multiple processors. The core of this framework is a clustering algorithm that decreases the number of nodes before expanding the SDF graph into a precedence 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 have developed the SDF composition theorem for testing if a clustering step is valid. The advantages of this framework are demonstrated with several practical, real-time examples.

1 Motivation

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 computations, 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).

There are several forms of dataflow defined in Ptolemy. In synchronous dataflow (SDF) [2], the number of tokens produced or consumed in one firing of a node 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, and have very predictable run-time behavior.

Figure 1 shows a simple SDF graph. In this graph, node A produces two tokens and node B consumes three tokens for each firing. In a *periodic* SDF schedule, the first-in/first-out (FIFO) buffer on each arc returns to its initial state after one schedule period. For each node x in a properly constructed SDF graph, there exists a positive integer $\mathbf{q}(x)$ such that node x must be invoked at least $\mathbf{q}(x)$ times in a each period of a periodic schedule [2]. For the example in figure 1, $\mathbf{q}(A) = 3$ and $\mathbf{q}(B) = 2$.

Given an SDF specification, we can construct a periodic schedule at compile-time that can be iterated an indefinite number of times without requiring unbounded memory. Such a schedule can be constructed by invoking each actor x exactly $\mathbf{q}(x)$ times, and ensuring that the data precedences defined by the SDF graph are respected. For figure 1, one such schedule is $AABAB$.

To schedule SDF graphs onto multiple processors, a **precedence DAG** (or simply "DAG") is constructed from the original SDF graph. For each node x in the original SDF graph, there are $\mathbf{q}(x)$ corresponding nodes in the precedence graph. Unfortunately, this expansion due to the repetition count of each SDF node can lead to an exponential growth of nodes in the DAG [3]. Such 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 [4]. Most uniprocessor SDF schedulers, on the other hand, do not require a DAG to be generated.

To limit the explosion of nodes when translating an SDF graph into a DAG graph, we apply *clustering* of connected subgraphs into larger grain *composite* nodes. The composite nodes will then be scheduled with one of the available uniprocessor schedulers. We cluster the nodes in

Figure 1: A simple SDF graph.

1058-6393/96 $5.00 © 1996 IEEE
Proceedings of ASILOMAR-29

a manner that simplifies the DAG without hiding much exploitable parallelism.

It is important to note that each resultant cluster is mapped onto a single processor. This observation motivates the modification of execution time minimizing clustering heuristics [5, 6, 7, 8] for use on the SDF graph. With these multiprocessor scheduling clustering heuristics, the resultant clusters are to be mapped onto a single processor. By clustering the SDF graph we also have the opportunity to use specialized uniprocessor SDF schedulers, which can optimize for such parameters as code size, buffer memory, and context switch overhead [9, 10, 11].

The structure of the paper is as follows. First, we introduce the *SDF composition theorem*, which allows us to test when it is legal to cluster two adjacent nodes the SDF graph. Next, we discuss the clustering heuristics that are used in the framework. Finally, we detail the hierarchical scheduling algorithm and present some performance measures on practical DSP examples that have been scheduled using the framework.

2 SDF composition

Unfortunately, not all clusterings of adjacent nodes in an SDF graph are possible. In fact, some clusterings will alter the SDF graph semantics by introducing *deadlock* into the graph. An SDF graph does not deadlock if and only if it has an *acyclic* precedence graph. Likewise, an SDF graph that does deadlock must have at least one cycle in the precedence graph. Therefore, we must not introduce cycles into the precedence graph by the clusterings we do. SDF precedence graph expansion is detailed in [7].

We have developed a theorem, called the *SDF composition theorem*, which establishes four clustering criteria that together provide a sufficient condition that a given clustering operation involving two adjacent nodes does not introduce deadlock. The first condition prevents the introduction of new cycles into both the SDF graph and the precedence graph. The last three conditions prevent the introduction of cycles into the precedence graph. These criteria are significantly more general than those that have been used in previous work on SDF clustering [9, 12], and can be tested efficiently. Due to lack of space, we refer the to [3] for the proof of the SDF composition theorem.

2.1 Notation

We use the following notational conventions when working with SDF graphs.

- $G = (V, E)$: A directed graph, G, made up of the set of nodes V, and set of arcs E.

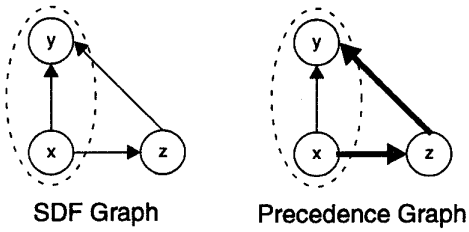

Figure 2: Example in violation of condition 1. A cycle is introduced in both the SDF and precedence graphs.

- κ_α : The number of samples consumed on the SDF arc α, per sink invocation.
- ρ_α : The number of samples produced on SDF arc α, per source invocation.
- δ_α : The number of initial samples ("delay") on the SDF arc α.
- $src(\alpha)$: The node that produces the tokens on arc α.
- $snk(\alpha)$: The node that consumes the tokens produced on arc α.
- If x and y are adjacent nodes in an SDF graph, then: $Q(x, y) = \frac{\mathbf{q}(x)}{gcd(\{\mathbf{q}(x), \mathbf{q}(y)\})}$. We can view $Q(x, y)$ as the number of times that node x is invoked in a single invocation of the cluster $\{x, y\}$.

Figures 2, 3 and 4 illustrate SDF graph clusterings that violate the conditions of the SDF composition theorem and thereby introduce deadlock. For the SDF graphs in these figures, $\rho_\alpha = 1$ and $\kappa_\alpha = 1$ for all arcs α, with the exception that $\kappa_{(x, y)} = 3$ in figure 4. In each of these figures, the cycle that is introduced into the precedence graph is depicted with wider arcs.

2.2 The SDF composition theorem

Suppose that G is a connected SDF graph, and (x, y) is an ordered pair of distinct, adjacent nodes in G. Then the graph that results from clustering $\{x, y\}$ into a single node, does not introduce cycles in the precedence graph if the following four conditions all hold.

1. **Cycle introduction condition**: There is no *simple path* from x to y that contains more than one arc. A simple path is one which does not visit any node along the path more than once. (Figure 2 depicts a clustering that violates this condition.)

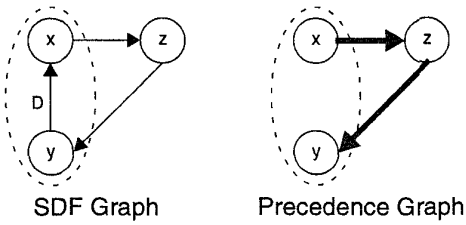

Figure 3: Example in violation of condition 2. A cycle is introduced in the precedence graph.

2. **Hidden delay condition**: If x and y are in the same strongly connected component, then both **a** *and* **b** must hold true. (Figure 3 depicts a clustering that violates this condition.)
   - **a.** at least one arc from x to y has zero delay
   - **b.** for some positive integer k, $\mathbf{q}(x) = k\mathbf{q}(y)$ or $\mathbf{q}(y) = k\mathbf{q}(x)$.
3. **First precedence shift condition**: If x is in a nontrivial strongly connected component C, then either **a** *or* **b** must hold true. (Figure 4 depicts a clustering that violates this condition.)

   a. for each $\alpha \in \left\{ \alpha' \middle| \begin{array}{c} snk(\alpha') = x \\ \textbf{and} \\ src(\alpha') \in C \\ \textbf{and} \\ src(\alpha') \notin \{x, y\} \end{array} \right\}$, there exists integers $k_1 > 0$ and $k_2 \geq 0$ such that $\rho_\alpha = k_1 Q(x, y)\kappa_\alpha$ and $\delta_\alpha = k_2 Q(x, y)\kappa_\alpha$.

   b. for each $\alpha \in \left\{ \alpha' \middle| \begin{array}{c} src(\alpha') = x \\ \textbf{and} \\ snk(\alpha') \in C \\ \textbf{and} \\ snk(\alpha') \notin \{x, y\} \end{array} \right\}$, there exists integers $k_1 > 0$ and $k_2 \geq 0$ such that $\kappa_\alpha = k_1 Q(x, y)\rho_\alpha$ and $\delta_\alpha = k_2 Q(x, y)\rho_\alpha$.
4. **Second precedence shift condition**: If y is in a nontrivial strongly connected component C, then either **a** *or* **b** must hold true.

   a. for each $\alpha \in \left\{ \alpha' \middle| \begin{array}{c} snk(\alpha') = y \\ \textbf{and} \\ src(\alpha') \in C \\ \textbf{and} \\ src(\alpha') \notin \{x, y\} \end{array} \right\}$, there exists integers $k_1 > 0$ and $k_2 \geq 0$ such that

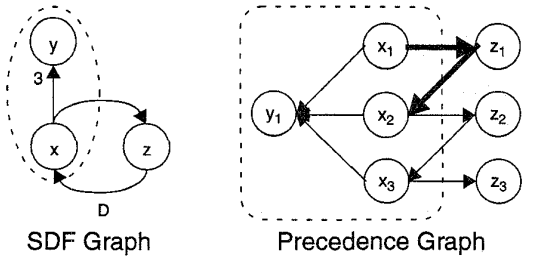

Figure 4: Example in violation of condition 3. A cycle is introduced in the precedence graph.

$\rho_\alpha = k_1 Q(x, y)\kappa_\alpha$ and $\delta_\alpha = k_2 Q(x, y)\kappa_\alpha$

b. for each $\alpha \in \left\{ \alpha' \middle| \begin{array}{c} src(\alpha') = y \\ \textbf{and} \\ snk(\alpha') \in C \\ \textbf{and} \\ snk(\alpha') \notin \{x, y\} \end{array} \right\}$, there exists integers $k_1 > 0$ and $k_2 \geq 0$ such that $\kappa_\alpha = k_1 Q(x, y)\rho_\alpha$ and $\delta_\alpha = k_2 Q(x, y)\rho_\alpha$.

Note that the conditions given in the SDF composition theorem may be satisfied for the ordered pair (y, x), even though they are not satisfied for (x, y). Thus, in general, both orderings should be tried before declaring a clustering operation valid.

3 Clustering Techniques

In this section we review our clustering techniques for SDF graphs.

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. We have implemented this technique in Ptolemy, where it has enabled the development of multiprocessor applications that have previously been impossible to synthesize using other SDF multiprocessing techniques.

The next clustering technique takes into account resource constraints. When mapping SDF graphs onto heterogeneous processors, a group of connected nodes 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.

The third clustering technique groups the nodes in a *well-ordered, uniform repetition count (URC)* SDF subgraph where the nodes do not have internal state. An acyclic graph is well ordered if it has only one topological sort, and a URC SDF subgraph is a subgraph in which the

$\mathbf{q}$-values of all nodes are identical. This clustering does not hide any of the available parallelism that will be exposed in the final DAG.

Finally, the last clustering technique is based on an adaptation of Sarkar's multiprocessor DAG scheduling heuristic to SDF graphs [6]. This is outlined in section 4.2 below.

4 Hierarchical Scheduling Algorithm

In this section, we detail the hierarchical scheduling algorithm. The algorithm is made up into three stages. The first stage is *initialization*, where some simple clustering heuristics are used which will not hide exploitable parallelism. The next stage is the *main loop*, where most of the clustering occurs. Finally in the *wrap up* stage, the individual schedulers are invoked.

4.1 Initialization

1. Cluster nodes that are on SDF well-ordered URC subgraphs without internal state [13].
2. Cluster nodes that share resource constraints which satisfy the SDF composition theorem.
3. Compute $\mathbf{q}(x)$ for each node x.
4. Construct the *acyclic SDF graph*, which involves removing each arc, α, where $\delta_\alpha \geq \kappa_\alpha \times \mathbf{q}(snk(\alpha))$ and then cluster the strongly connected components [3].
5. Compute the total IPC cost for each arc on the *acyclic SDF graph.*

4.2 Main Loop

1. Apply one step of Sarkar's multiprocessor clustering heuristic on the *acyclic SDF graph.*
2. Using the SDF composition theorem, test the resulting cluster candidate to make sure it does not introduce deadlock.
3. If the cluster candidate does not introduce deadlock, then perform the corresponding clustering operation, and update $\mathbf{q}$ accordingly.
4. Repeat 1,2 until the precendence graph is limited to a certain size or there are no more legal candidate clusters. A stopping condition that limits the precedence graph to a tractable size is :

   $\sum_{v_i \in V} \mathbf{q}(v_i) < K\max(|V|, P)$. In this equation K is a user-settable constant, $|V|$ is the number of nodes in the original SDF graph and P is the number of processors the target architecture.

4.3 Wrap up

1. Schedule SDF uniprocessor clusters with the loop scheduler of reference [9].
2. Schedule user specified clusters with the given scheduler.
3. Schedule clustered system with the user-specified multiprocessor scheduler.

5 Performance

The hierarchical scheduling framework for user specified clustering has been implemented in Ptolemy [13]. Four signal processing applications have been synthesized for a heterogeneous multiprocessor consisting of a RISC and a DSP processor. A table comparing the results of user specified hierarchical scheduling versus full DAG expansion multiprocessor scheduling is given in table 1.

In the four examples the scheduling time improved one to two orders of magnitude, while the makespan was not significantly increased. Through use of uniprocessor

System	SDF Graph Size	DAG Size	Scheduling Time in CPU Seconds	Makespan	P1: DSP Code Size Assembly	P2: Sparc Code Size C
FM-Synthesis 128 pt. spectrum	44	14 / 806 57 x smaller	0.47 / 4.35 9.25 x faster	28832 / 28832 no difference	408 / 408 same	34K / 420K 12 x smaller
bpsk (530 bps)	31	9 / 2628 292 x smaller	0.37 / 14.71 40 x faster	41566 / 41368 < 1% difference	424 / 32045 75 x smaller	14K / 56K 4 x smaller
4-QAM (320 bps) eye diagram	59	15 / 9267 618 x smaller	0.91 / 80.87 87 x faster	150123 / 150123 no difference	1421 / 87533 62 x smaller	38K / 63K 1.7 x smaller
4-QAM (640 bps)	52	10 / 3490 349 x smaller	0.69 / 20.1 29 x faster	40037 / 39707 < 1% difference	848 / 29720 35 x smaller	35K / 56K 1.6 x smaller

Table 1: Performance of the hierarchical scheduling framework for user-specified clustering.

schedulers on the final clusters, we are able to realize a significant improvement in memory usage. This improvement in memory is particularly evident in the acoustic 320 bps quadrature amplitude modulation (4-QAM) acoustical modem, where the multiprocessor schedule generated from the fully expanded DAG has one function call (or in-lined procedure) for each of its 9267 nodes as compared to only 59 function calls for the hierarchical schedule. In the case of all three modem examples, where the DSP card only has access to 16K of memory, this framework enabled the synthesis of applications previously not possible using full DAG expansion multiprocessor scheduling techniques.

6 Conclusions

We have developed a hierarchical scheduling framework for SDF graphs being mapped onto multiple processors. Using user specified clustering, this framework has dramatically improved the scheduling time and reduced the memory requirements needed in the generated system. In some cases, the hierarchical scheduling framework enabled the synthesis of applications previously impossible.

To test whether a given clustering step is valid, we have developed the SDF composition theorem. This theorem is significantly more general than those that have been developed in previous work and can be tested efficiently.

We plan to implement more automated clustering heuristics for use on the SDF graph before the SDF to DAG translation. As with the adaptation of Sarkar's clustering heuristic, these will be inspired by the DAG clustering heuristics found in other multiprocessor schedulers. The objective is to hide only the parallelism that would not be exploited in final multiprocessor scheduling phase, 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, 1994. http://ptolemy.eecs.berkeley.edu/papers/JEurSim

[2] E.A. Lee and D.G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, 1987.

[3] J. L. Pino, S. S. Bhattacharyya, and E. A. Lee, *A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs*, UCB/ERL M95/36, Electronics Research Laboratory, University of California at Berkeley, May 30, 1995. http://ptolemy.eecs.berkeley.edu/papers/erl-95-36

[4] 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, no. 4, 1992.

[5] S.J. Kim and J.C. Browne, "A general approach to mapping of parallel computations upon multiprocessor architectures," *International Conference on Parallel Processing*, vol. 3, University Park, PA, USA, Pennsylvania State Univ, 1988.

[6] V. Sarkar, *Partitioning and scheduling parallel programs for multiprocessors*, Cambridge, Mass.: MIT Press, 1989.

[7] G. C. Sih, *Multiprocessor scheduling to account for interprocessor communication*, Ph.D. Dissertation, UCB/ERL M91/29, Electronics Research Laboratory, University of California at Berkeley, 1991.

[8] H. Printz, *Automatic mapping of large signal processing systems to a parallel machine*, Ph.D. Dissertation CMU-CS-91-101, Carnegie Mellon, 1991.

[9] S. S. Bhattacharyya, J. T. Buck, S. Ha, and E. A. Lee, "Generating compact code from dataflow specifications of multirate signal processing algorithms," *IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications*, March, 1995.

[10] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Two complementary heuristics for translating graphical DSP programs into minimum memory implementations*, Memorandum UCB/ERL M95/3, Electronics Research Laboratory, University of California at Berkeley, January, 1995.

[11] S. Ritz, M. Pankert, and H. Meyr, "Optimum vectorization of scalable synchronous dataflow graphs," *Proceedings of the International Conference on Application-Specific Array Processors*, October, 1993.

[12] S.S. Bhattacharyya, *Compiling dataflow programs for digital signal processing*, Ph.D. Dissertation UCB/ERL M94/52, University of California at Berkeley, 1994.

[13] J.L. Pino and E.A. Lee, "Hierarchical static scheduling of dataflow graphs onto multiple processors," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Detroit, Michigan, IEEE, 1995. http://ptolemy.eecs.berkeley.edu/papers/hierStaticSched