SCHEDULING DYNAMIC DATAFLOW GRAPHS WITH BOUNDED MEMORY USING THE TOKEN FLOW MODEL

Joseph T. Buck and Edward A. Lee

{jbuck,eal}@EECS.Berkeley.EDU
Dept. of EECS, University of California, Berkeley, CA 94720

— from: *Proc. of ICASSP '93*, Minneapolis, April, 1993 —

ABSTRACT

This paper builds upon research by Lee [1] concerning the **token flow model***, an analytical model for the behavior of dataflow graphs with data-dependent control flow, by analyzing the properties of cycles of the schedule: sequences of actor executions that return the graph to its initial state. Necessary and sufficient conditions are given for the existence of a bounded cyclic schedule, as well as sufficient conditions for execution of the graph in bounded memory. The techniques presented in this paper apply to a more general class of dataflow graphs than existing methods.*

MOTIVATION

Dataflow graphs have proven to be an effective representation for problems in digital signal processing, both because the representation is natural for DSP researchers and because the representation exposes the parallelism of the algorithm and imposes minimal constraints on the order of its evaluation. Since the representation does not over-constrain the order of operations, a scheduler has the freedom it needs to adequately exploit deep pipelines, to maximize re-use of limited hardware resources, or to exploit parallel processing units. To get these benefits, compilers for pipelined or parallel machines often heavily rely on dataflow analysis.

When the actors in the dataflow graph are restricted to be synchronous (meaning that the number of tokens produced and consumed by each actor is fixed and known at compile time), powerful techniques exist for demonstrating the consistency of the graph, determining memory requirements, and scheduling its execution on one or more processors [3].

Graphs having only synchronous actors have completely deterministic control flow. Therefore, the synchronous dataflow (SDF) model is overly restrictive for digital signal processing, because while typical DSP algorithms require relatively little run-time decision-making, "little" is not the same as "none". We therefore seek to extend SDF techniques to work with more general dataflow graphs.

KEY QUESTIONS

The following questions can be asked about any dataflow graph:

1. Do cyclic schedules exist? A cyclic schedule is a sequences of actor executions that return the graph to its original state. Graphs that lack cyclic schedules because of differences in token flow rates are said to be *inconsistent* (see figure 1).
2. Does the graph have a *bounded* cyclic schedule? This question is important if the graph is to be scheduled with a hard real-time constraint.
3. Does the graph deadlock? A graph is deadlocked if it reaches a configuration in which no actor can be executed. This most often occurs due to directed loops with unsufficient delays (see figure 2).
4. Can the graph be scheduled to use bounded memory?

For synchronous dataflow graphs, algorithms exist to answer all four questions for any graph [3]. The questions are answered by solving the *balance equations* for the graph (the balance equations are discussed later in this paper in a more general form). If there is a nontrivial solution, then it specifies the number of executions of each actor required for a cyclic schedule (otherwise, the answer to question 1 is no). A precedence graph is then constructed, as described in [3]. If this construction fails, the graph is deadlocked. Otherwise, the graph does not deadlock, and questions 2 and 4 can be answered in the affirmative as well, because if any schedule exists, it is bounded in both schedule length and memory requirements.

When conditional actors are admitted, the questions become more difficult and interesting, particularly the question of bounded memory. The existence of cycles does not guarantee that the cycle length will be bounded or that memory requirements are bounded.

The token flow model admits actors for which the number of tokens produced and consumed on each arc is a function of the data values of certain Boolean tokens. This paper gives necessary and sufficient conditions for determining whether such a graph

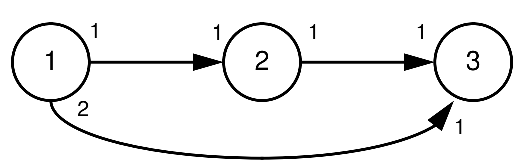

Figure 1. An inconsistent dataflow graph.

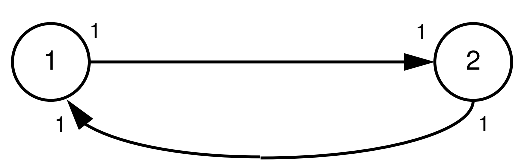

Figure 2. A deadlocked dataflow graph.

This work was supported by a grant from the Semiconductor Research Corporation (93-DC-008)

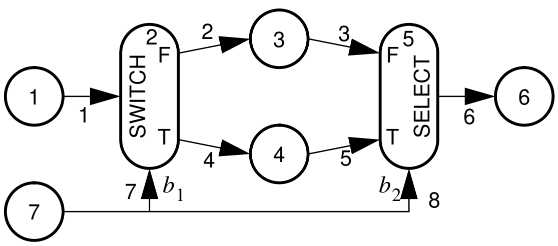

Figure 3. An if-then-else dataflow graph. The numbers adjacent to the arcs merely identify them.

has a bounded cycle length, and sufficient conditions for bounded memory.

PRIOR WORK

We define dynamic dataflow actors to be actors for which the number of tokens consumed or produced by a firing cannot be specified statically. Typical examples are the SWITCH and SELECT which route tokens depending on a Boolean input, as shown in figure 3. Because of the difficulty in determining memory requirements of a dataflow graph with dynamic actors, several authors have proposed restricting their use to particular patterns or schema that can be shown to result in a bounded storage requirement [2,4]. Thus, for example, Gao *et. al.* permit such actors only in conditional or loop schema [2]. They call graphs that require bounded memory *well-behaved.*

In [1], Lee demonstrates how to assure that the long-term average flow rates on each arc of a dynamic dataflow graph are consistent, but as is shown by Gao *et. al.* [2], Lee's "strongly consistent" criterion is not sufficient to guarantee bounded memory requirements.

In the token flow model of [1], the number of tokens produced or consumed by an actor is either constant, as in the SDF model, or given by a symbolic function of the proportion of TRUE tokens on the Boolean streams in the system (see figure 4). The precise conditions on actors is that the number of tokens produced or consumed on each arc must either be constant, or a function of a Boolean-valued token produced or consumed on another arc, which is called a control arc. A control arc that controls an output arc may itself be an input or output arc; if it controls an input arc, it must be an input. This restriction enables a scheduler to execute actors correctly by looking only at the Boolean streams. Actors that satisfy these conditions are called Boolean-controlled dataflow actors, or BDF actors for short.

The p_i quantities that appear in the figure are open to several interpretations. Loosely, p_i is the long-term proportion of TRUE tokens in the boolean stream b_i, which supplies the control inputs. Several interpretations are possible: in a probabilistic interpretation, it is the marginal probability that a token selected from b_i is true, though this requires that the boolean stream be stationary in the mean to be well-defined. Fortunately, we do not require this; for most dataflow graphs, we can interpret the quantity as the proportion of TRUE tokens in a well-defined finite sequence of actor executions corresponding to a *complete cycle*, a sequence of actor executions that returns the graph to its original state.

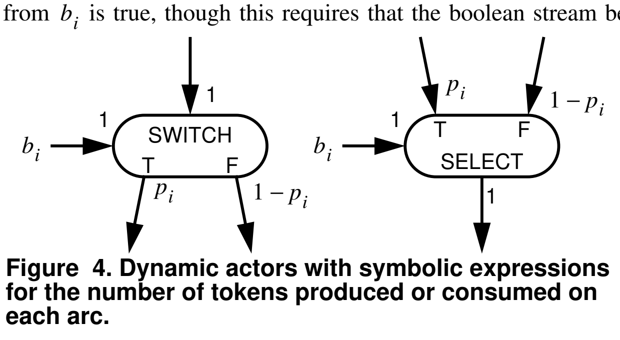

Figure 4. Dynamic actors with symbolic expressions for the number of tokens produced or consumed on each arc.

To determine whether a graph is strongly consistent, the "balance equations" require us to determine a number of repetitions for each actor that will cause the number of tokens produced and consumed on each arc to be equal. In general, this solution will be a symbolic function of the p_i. If the balance equations have nontrivial solutions regardless of the values of the boolean proportions, the graph is said to be strongly consistent. If solutions only exist for some values, the graph is *weakly consistent*; such graphs are usually errors. The graph in figure 3 is strongly consistent, as we will now show. We form the topology matrix $\Gamma(\vec{p})$, in which the element γ_{ij} indicates the number of tokens written by actor i onto arc j, which is in general a function of $\vec{p}$, the vector of p_i values. Negative entries indicate that the actor consumes rather than produces tokens. For figure 3 we have the following topology matrix:

$$\Gamma(\vec{p}) = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & (1-p_1) & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & (p_2-1) & 0 & 0 \\ 0 & p_1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -p_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} \quad \textbf{(1)}$$

We now attempt to find a repetition vector $r(\vec{p})$ such that

$$\Gamma(\vec{p})\, r(\vec{p}) = \mathbf{0}.$$

We consider only nontrivial solutions (note that the zero vector is always a solution). In practice, fast techniques exist to find solutions to the balance equations; the techniques of [3] are readily extended to the BDF case.

If all actors except the SWITCH and SELECT in figure 3 produce and consume a single token when they fire, then the graph can be executed with bounded memory. However, as Gao *et. al.* show in [2], if actors 3 and 4 read and write two tokens per execution, this graph is still strongly consistent but now requires unbounded storage (consider, for example, what happens if actor 7 produces a single TRUE token followed by a large number of FALSE tokens). So strong consistency does not imply bounded memory requirements.

Strong consistency is based on the long term proportion of TRUE tokens in the Boolean streams. A probabilistic interpretation for this is given in [8]. If we replace these long term proportions with a symbol representing the number (rather than proportion) of true tokens over some short term firing sequence, we can obtain necessary and sufficient conditions for existence of a bounded cyclic schedule. This, in turn, is sufficient (though not necessary) for bounded memory.

Using the techniques of [1], we would notice that b_1 and b_2 are produced by the same actor and therefore have the same proportion of true values, p_b. Solving the balance equations given above for the average rates of tokens produced and consumed, we obtain the solution vector

$$r(\vec{p}) = k(1, 1, 1-p_b, p_b, 1, 1, 1)$$

where k is arbitrary. One interpretation for this vector is that it gives the relative firing rates for the actors in figure 1. As this solution exists regardless of the value of p_b, the graph is said to be strongly consistent.

If we solve the balance equations for Gao's example, where actors 3 and 4 consume and produce two tokens, we obtain

$$k(2, 2, 1-p_b, p_b, 2, 2, 2) .$$

Again, this solves the balance equations regardless of the value of p_b, even though we can show that executing this graph requires unbounded memory. We distinguish the two cases by solving for the number of actor executions in the cycle as a function of the number, rather than the proportion, of true tokens on boolean streams. For a firing sequence consisting of n firings of actor 7, we obtain

$$(n, n, n-n_T, n_T, n, n, n)$$

for the case where actors produce and consume a single token. Here n is the number of boolean tokens produced and n_T is the number of true valued tokens. The smallest integer solution has $n = 1$, and thus defines a minimal cycle (n_T may be either 0 or 1 for the cycle, depending on the value of the boolean token produced). For Gao's example the balance equations yield

$$(n, n, \frac{n-n_T}{2}, \frac{n_T}{2}, n, n, n) .$$

This vector represents the number of firings of each actor, given that actor 7 fires n times. Since we can't have fractional firings, each term in the above vector must be an integer. Since no fixed value of n can guarantee this, an unbounded cycle length may result. Note that this alone is not sufficient to prove that the memory requirement is unbounded, however. All that we have shown is that there is no upper bound on the number of actor firings required to return the graph to its original state.

By solving for the number of executions of each actor in a cyclic schedule and demonstrating that it is bounded, we establish one of the two conditions needed for assuring that the graph can be scheduled with bounded cycle length. We must also establish that the graph does not deadlock. This is done by constructing the annotated acyclic precedence graph, a graph giving the data dependencies of the actor executions (see [1] for how this is done). These two conditions are necessary and sufficient for existence of a bounded cyclic schedule.

DATA-DEPENDENT ITERATION; CLUSTERING

Data-dependent iteration results in cycles of unbounded length even though only bounded memory is required. Such graphs can

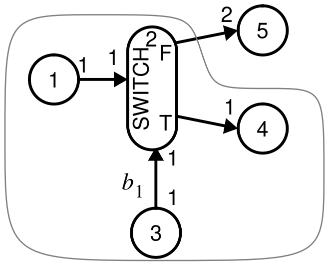

Figure 5. A graph with data-dependent iteration that is readily handled by clustering.

proved to require bounded memory by applying a clustering algorithm. Because of space limitations, the algorithm will only be described qualitatively here. The goal of the clustering algorithm is to map the graph into its traditional control structures such as if-then-else and do-while, whenever possible. The substructures are treated as atomic actors from their exterior. If the interior of each control structure has a bounded cyclic schedule, the graph can be scheduled in bounded memory. For example, in the dataflow equivalent of a do-while, we wish to check that each iteration of the loop executes a bounded number of actors and subclusters.

We say that two adjacent actors have the same sample rate if they are connected by an arc and the source star always writes the same number of tokens on the arc as the destination star reads. The clustering algorithm operates by alternately applying two types of transformations.

The first series of transformations is called the *merge pass*. In this phase, adjacent actors with the same sample rate are merged into a single cluster, where possible. Actors may not be merged if this would create deadlock, or if the resulting cluster would not be a BDF actor (for example, it may depend on a control arc which is hidden by the merge operation).

The second series of transformations is called the *loop pass*. In this phase, clusters are iterated, made conditional, or placed in a do-while loop as necessary to cause them to match the sample rates of their neighbors.

The two types of transformations are alternated until no more changes are possible. At this point, if the interior of each cluster has a schedule of bounded length, and the top-level cluster does as well, then the entire graph can be scheduled with bounded memory. This technique is, in a sense, the reverse of that of Gao and Dennis [2,4] in that the analysis finds the fundamental constructs rather than requiring that the graph be built up out of them.

For example, the graph in figure 5 does not correspond to any of the traditional dataflow loop schema. However, actors 1 through 4 can be combined into a single cluster because of their common sample rate. If this cluster is then executed repeatedly until a false token is produced by actor 3, the cluster as a whole will produce one token, and thus operate exactly like a synchronous dataflow actor.

STATE ENUMERATION

For graphs with delays on Boolean arcs, such as in figure 6, the techniques presented so far do not suffice; clustering reduces the graph to a simpler system, but it is still necessary to assure that the simpler system can be scheduled in bounded memory. We can show that the graph in the figure is strongly consistent by assuming that $p_1 = p_2$, but the skew between the two boolean streams (one is the delay of the other) may present problems. One technique that can be used is to enumerate the states that the system

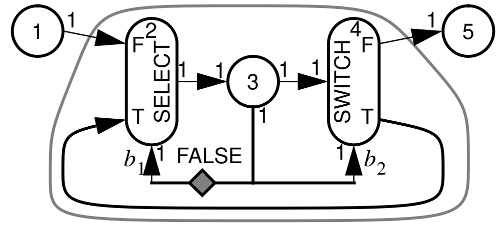

Figure 6. A do-while construct. Both clustering and state enumeration are required to handle this case.

can reach by simulating the execution of the graph by a dynamic dataflow machine. This type of search suffices, for the system in this figure, to show that the subsystem consisting of actors 2, 3, and 4 never need have more than one token on any arc. Furthermore, it behaves as a whole like a synchronous actor, reading one token from actor 1 and writing one token to actor 5.

An alternative approach is to construct a *preamble,* which is a sequence of actor executions that eliminates any delays on boolean arcs. In this case, executing actors 1 and 2 in the preamble will suffice; this transforms the system to a system that has bounded cycle length.

If the graph cannot be scheduled in bounded memory, the state enumeration procedure described above will not terminate. One possible solution to this difficulty is to impose an upper bound to the number of tokens that may appear on each arc, according to some heuristic, and to assume that there is a problem if this bound is exceeded. A technique similar to this is used in Ptolemy's dynamic dataflow scheduler [5].

It is also possible to terminate the state enumeration algorithm "in the other direction", with an indication that unbounded memory is definitely required, by a type of mathematical induction. In graphs such as Gao's example discussed earlier, it is possible to show that, by starting in a state with N tokens on a particular arc, we must always reach a state that has $N+1$ tokens on that same arc but is otherwise the same, for any value of N, given certain boolean outcomes.

SYSTEMS REQUIRING UNBOUNDED MEMORY

Researchers pursuing this problem have used the label "well-behaved" to describe systems with bounded memory requirements, but perfectly valid problems may require unbounded memory; consider a parser for a context-free programming language, for example. The traditional solution to such problems has been to use dynamic scheduling with dynamic memory allocation. However, in any such problem there are usually subsystems that each require bounded memory, so it is still advantageous to cluster the graph so as to be able to use static memory allocation for arcs that can be demonstrated to require bounded memory. The clustering algorithm described in this paper accomplishes this task readily.

Some complex and irregular graphs will not be successfully clustered by our algorithm, and state enumeration requires a heuristic to avoid exploring the (possibly infinite) state space forever. Because of this, some graphs that have bounded-memory schedules may not be handled successfully by these techniques. If so, some dynamic memory allocation will be used even though it is not actually required. However, graphs composed only of the "well-behaved" structures appearing in the dataflow literature are handled successfully.

It must be noted that the general problem of determining whether a BDF graph can be scheduled with bounded memory is undecidable (equivalent to the halting problem); this is because BDF graphs are Turing-equivalent. Given this, it should not be surprising that the techniques presented here give only a partial solution.

FURTHER WORK

We are currently implementing these scheduling techniques for use in the Ptolemy system, a multi-paradigm simulation and prototyping environment [5]. The first application will be code generation for digital signal processing problems from dynamic dataflow graphs with a single-processor target. We hope to demonstrate an efficient, high-quality code generator to increase the power of those previously implemented in Ptolemy and its predecessor system, Gabriel [7]. The looped structure produced by the clustering algorithm, with a bit of postprocessing to eliminate duplicate tests, is suitable for code generation for a single processor.

Extension to multiple-processor scheduling by extending techniques such as those of [6] to support data-dependent actors are also contemplated. For the special case of bounded-cycle-length graphs, minimax scheduling is the logical criterion, especially in hard-real-time systems. For the more general case of data-dependent iteration, we plan to apply techniques from [9].

REFERENCES

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

[2] G.R. Gao, R. Govindarajan, P. Panangaden, "Well-Behaved Dataflow Programs for DSP Computation,", *Proc. ICASSP 1992,* San Francisco, California, March 1992.

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

[4] J.B. Dennis, "First Version Data Flow Procedure Language", Technical Memo MAC TM61, May, 1975, MIT Laboratory for Computer Science

[5] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, "Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems", *International Journal of Computer Simulation*, to appear.

[6] G. Sih, "Multiprocessor Scheduling to Account for Interprocessor Communication", Ph.D. Thesis, Memorandum No. UCB/ERL M91/29, UC Berkeley, CA 94720, April 22, 1991

[7] J. Bier, E. Goei, W. Ho, P. Lapsley, M. O'Reilly, G. Sih and E.A. Lee, "Gabriel: A Design Environment for DSP," *IEEE Micro*, October 1990, Vol. 10, No. 5, pp. 28-45.

[8] J. Buck and E. A. Lee, "The Token Flow Model," presented at *Data Flow Workshop*, Hamilton Island, Australia, May, 1992.

[9] S. Ha, "Compile-Time Scheduling of Dataflow Program Graphs with Dynamic Constructs," Ph.D. Dissertation, EECS Dept., University of California, Berkeley, CA 94720, April 1992.

[10]