In Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation, Samos, Greece, July 2004.

Analysis of Dataflow Programs with Interval-limited Data-rates

Jürgen Teich[1]* and Shuvra S. Bhattacharyya[2]**

[1] University of Erlangen-Nuremberg, Germany,
teich@cs.fau.de, http://www12.informatik.uni-erlangen.de
[2] University of Maryland, U.S.A.,
ssb@eng.umd.edu, http://www.eng.umd.edu

Abstract In this paper, we consider the problem of analyzing data flow programs with the property that actor production and consumption rates are not constant and fixed, but limited by intervals. Such interval ranges may result from uncertainty in the specification of an actor or as a design freedom of the model. Major questions such as *consistency* and *buffer memory requirements* for single-processor *schedules* will be analyzed here for such specifications for the first time.

1 Motivation

The role of data flow models of computation is becoming increasingly important for modeling and synthesis of digital processing applications. Our paper is concerned with analyzing dataflow graphs whose component data rates are not known precisely in advance. Such models are often given due to imprecise specifications or due to uncertainties in the implementation. Examples of imprecise specifications include unknown execution times of tasks, unknown data rates, unknown consumption and production behavior of modules and many more. For example, a speech compression system may have a fixed overall structure. However, the subsystem data rates are typically influenced by the size of the speech segment that is to be processed [1]. Here, we will propose a data flow model of computation that is able to model such uncertain behavior.

To understand such models, it useful to first review principles of the *synchronous data flow* (SDF) model of computation, where production and consumption rates of data flow actors, representing computational blocks, consume fixed, known amounts of data (tokens) upon each invocation. For such graph models, often represented by a graph in which actors are connected by directed arcs that transport tokens, many interesting results have been shown such as 1) *consistency*, 2) *memory bounds and memory analysis*, and 3) *scheduling algorithms*. Consistency, e.g., is a static property of an SDF-graph specification which is necessary in order to guarantee the existence of a finite sequence of actor firings, also called a *schedule*. Typically, an SDF specification is compiled by constructing a *valid schedule*, i.e., a schedule that fires each actor at least once, does not deadlock, and produces no net change in the number of tokens queued on each edge. For each actor firing, a corresponding code block is instantiated from a library to produce machine code.

In [5], efficient algorithms are presented to determine whether or not a given SDF graph is consistent or not and to determine the minimum number of firings of each actor

* The first author was supported in this work by the Deutsche Forschungsgemeinschaft (DFG) under grant Te163/5-2.
** The second author was supported in this work by the US National Science Foundation (grant numbers 0325119 and 9734275), and the Semiconductor Research Corporation (contract number 2001-HJ-905).

in a valid schedule. Typically, a (sequential) schedule may be represented by a string of actor firings such as $A(2B)(6C)$ for an SDF graph with 3 actors named A, B, and C in Fig. 1. Here, a parameterized term $(nS_1S_2 \ldots S_k)$ specifies n sucessive firings of the subschedule $S_1S_2 \ldots S_k$, and may be translated into a loop in the target code. Each parenthesized term $(nS_1S_2 \ldots S_k)$ is called *schedule loop*. A *looped schedule* is a finite sequence $V_1V_2 \ldots V_k$, where each V_i is either an actor or a schedule loop.

For a given SDF graph G, lower bounds for the amount of required program memory have been shown to correspond to so-called *single-appearance schedules*, where each actor appears exactly once in the schedule term, e.g., $A(2B(3C))$ for the graph shown in Fig. 1.

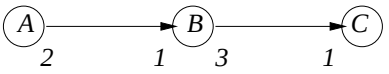

Figure 1. Simple SDF graph

Furthermore, data memory requirements, resulting from implementing each arc communication via a FIFO, are given by the sum of the maximum number of tokens, resulting on each arc during the execution of a schedule.

In [2], algorithms such as PGAN (pairwise grouping of adjacent nodes) and a complementary algorithm called RPMC (recursive partitioning by minimum cuts) have been presented to create schedules with the goal to generate single-appearance schedules in the first line with the second goal to minimize the amount of data memory needed.

With results such as these in mind, we propose a powerful intermediate representation model for a broad class of non-deterministic dataflow graphs. This model, called ILDF, standing for *interval-rate, locally-static dataflow*. In ILDF graphs, the production and consumption rates on graph edges remains constant throughout execution of the graph (*locally static*), but these constant values are not known exactly at compile time; instead it is only known what their minimum and maximum values (*interval-rate*) are.

Locally static behavior arises naturally in reconfigurable dataflow graphs, such as those arising using parameterized dataflow semantics [1]. For example, a speech compression system may have a fixed overall structure with subsystem data rates that are influenced by the size of the speech segment that is to be processed [1]. Similarly, during rapid prototyping of a filter bank application [9], one might parameterize the data rates of various filters to explore a range of different mulitirate topologies.

Figure 2 shows a compact disc to digital audio tape sample rate conversion system that is formulated as an ILDF graph. Two of the conversion stages, B and D, are not fully specified at compile time to allow for run-time experimentation during rapid prototyping. Using ILDF parameterized schedules, different versions of these filters can be evaluated without having to re-schedule and re-compile the application.

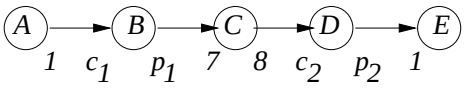

Figure 2. An example of a compact disc to digital audio tape sample rate conversion system that is formulated as an ILDF graph.

Reasonable interval ranges for the unknown consumption and production values are $c_1, c_2 \in [3, 7]$, and $p_1, p_2 \in [2, 10]$. In addition, to achieve the desired overall rate conversion, the values must satisfy $(p_1p_2)/(c_1c_2) = 20/21$.

An ILDF representation of such applications with careful compile-time analysis can help to streamline run-time management (e.g., scheduling and memory allocation) and verification of such applications. This is the main motivation for our work on ILDF.

More precisely, in ILDF, the consumption and production rates, denoted $c(e)$ and $p(e)$ in the following for each arc $e \in E$ of a directed graph $G(V, E)$ with actor set V and arc set E, are not constants and not statically fixed. Instead, the only information that we have is that the numbers range within intervals, e.g., $p(e) \in [p(e)_{\min} \dots p(e)_{\max}] \cap \mathbf{N}$,[3] for each arc $e \in E$. Similarly, $c(e) \in [c(e)_{\min} \dots c(e)_{\max}] \cap \mathbf{N}$.

The uncertainty given by the introduction of rate intervals may be the result of an unknown or activation-dependent behavior of an actor, or the freedom of the specification to later fix the parameter if possible or necessary in order to guarantee certain properties.

For ILDF graphs, we want to address similar issues as with SDF graphs:

- consistency
- memory bounds
- valid schedule

In Section 2, we define the ILDF model. In Section 3, we assume that a valid schedule is possible and given by an algorithm such as PGAN for which we investigate the problem of determining the data memory requirements. We deduce expressions and algorithms to determine the maximum required data memory requirements. If the execution time of an actor is also bounded by an interval, we finally also analyse worst-case and best-case execution times of a given schedule in Section 4. Experimental results are given in Section 5. Section 6 provides a review of related work.

2 Interval-rate SDF

Definition 1. *An ILDF graph $G(V, E)$ is a locally-static dataflow graph where the production value $p(e)$, consumption value $c(e)$, and delay $d(e)$ of each edge $e \in E$ are constrained by intervals, i.e.,*

- $c(e) \in [cmin(e) \dots cmax(e)] \cap \mathbf{N}$,
- $p(e) \in [pmin(e) \dots pmax(e)] \cap \mathbf{N}$.
- $d(e) \in [dmin(e) \dots dmax(e)] \cap \mathbf{N}$.

2.1 Consistency

Since the production and consumption values of an ILDF graph are not precisely known in advance, it is generally not possible to determine whether a particular execution of an ILDF graph will proceed in a consistent manner (i.e., with avoidance of deadlock, and with balanced data production and consumption along the graph edges). We can speak of three different levels of consistency for ILDF graphs. First, an ILDF graph is *consistent*, or *inherently consistent*, if for every valid setting of production, consumption, and delay (P-C-D) values (any setting that conforms to the production, consumption, and delay intervals associated with the graph edges), the corresponding synchronous dataflow graph is consistent. Inherent consistency occurs, for example, in chain-structured ILDF graphs, such as the ILDF application shown in Figure 2. Conversely, an ILDF graph is *inconsistent*, or *inherently inconsistent*, if for every valid setting of P-C-D values, the corresponding synchronous dataflow graph is inconsistent. An ILDF graph that contains a delay-free cycle is an example of an inherently inconsistent graph. Third, an ILDF

[3] Let **N** denote the set of the natural numbers in the following.

graph is *conditionally consistent* if it is neither inherently consistent nor inherently inconsistent. In other words, an ILDF graph is conditionally consistent if there is at least one valid setting of P-C-D values that gives a consistent synchronous dataflow graph, and there is at least one valid P-C-D setting that leads to an inconsistent synchronous dataflow graph. In general, the particular form of consistency that an ILDF graph exhibits depends both on the topology of the graph, and the production, consumption, and delay intervals.

3 Memory analysis

In general, for an SDF graph, the buffer memory lower bound of a delayless arc $e \in E$ is given by

$$BMLB(e) = \frac{p(e)c(e)}{\gcd(\{p(e), c(e)\})} \tag{1}$$

For example, in Fig. 1, we obtain $BMLB((A,B)) = 2$, $BMLB((B,C)) = 3$.[4] A valid single-appearance schedule that achieves the total lower bound of 5 memory units (sum) is $A(2B(3C))$. In this consistent SDF graph, actor A thus fires 1 time, actor B two times, and actor C 6 times.

Let's see how the buffer memory lower bound may be computed for ILDF graphs. For example, let's look at an ILDF graph with just two actors A and B and one arc $e = (A,B)$. Let $p(e) = 2$, and $c(e) \in [1 \ldots 3]$. The BMLB depends obviously on the value of the consumption rate. Corresponding to Eq. (1), we obtain $BMLB(e) = 3$ in case $c(e) = 1$, $BMLB(e) = 2$ if $c(e) = 2$, and $BMLB(e) = 6$ if $c(e) = 3$. Hence, if we do not know the actual rate, we must reserve at least

$$BMLB(e) = \max_{p(e),c(e)} \frac{p(e)c(e)}{\gcd(\{p(e), c(e)\})} \tag{2}$$

memory space for arc e.

Example: Consider the ILDF graph in Fig. 3 and the highlighted arc $e = (A,B)$. Under the implicit assumption of consistency, we would like to know the minimal memory requirements for arc e if $p(e) \in [1 \ldots 3]$ and $c(e) \in [2,3]$. Obviously, the lower bound is obtained for the combination $p(e) = 3, c(e) = 2$ or for $p(e) = 2, c(e) = 3$ with 6 units.

The natural question is how one can compute the BMLBs for each arc efficiently without having to compute all combinations of $p(e)$ and $c(e)$ separately?

Theorem 1 (BMLB computation). *Given an ILDF graph $G(V,E)$ with a rate interval associated with each production number $p(e)$ and each consumption number $c(e)$. The maximum in Eq. (2) determining $BMLB(e)$ is determined by the largest product of $p(e)c(e)$ where $\gcd(p(e), c(e))$ reaches its minimal possible value.*

Proof. Let $p \in [pmin(e) \ldots pmax(e)]$ and $c \in [cmin(e) \ldots cmax(e)]$ be a combination, where $\gcd(p(e), c(e))$ reaches its minimum value and let p, c satisfy the condition that there is no distinct pair $p' \geq p$, $c' \geq c$ in the above interval ranges with larger product $p'c'$ and with equal gcd according to the assumption in the theorem. Let $bmlb$ be the corresponding value of the buffer memory lower bound according to Eq. (1). We will show in the following that decreasing either p or c will produce lower values of the

[4] Note that the value $BMLB(e)$ is given as the lcm (least common multiple) of the values $p(e)$ and $c(e)$.

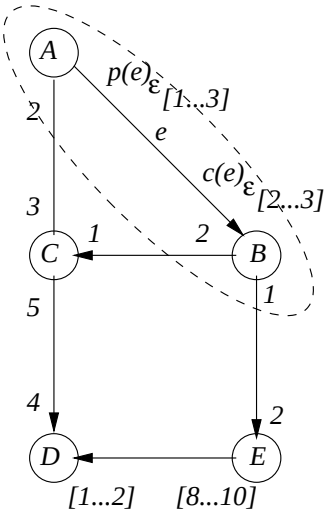

Figure 3. ILDF graph

buffer memory lower bound according to Eq. (1). Hence, the maximum according to Eq. (1) cannot be determined by such pairs. Indeed, if we decrease p by $\delta \in \mathbf{N}_0$ or c by $\gamma \in \mathbf{N_0}$, we prove that we will obtain a BMLB value $bmlb'$ that is smaller than or equal to $bmlb$. We have

$$bmlb' = \frac{(p-\delta)(c-\gamma)}{\gcd(p-\delta, c-\gamma)} \tag{3}$$

Relating $bmlb$ and $bmlb'$ gives

$$\frac{bmlb}{bmlb'} = \frac{(pc)}{(p-\delta)(c-\gamma))} \times \frac{\gcd(p-\delta, c-\gamma)}{\gcd(p,c)} \tag{4}$$

$$\geq 1 \times 1 \tag{5}$$

The first fraction is rational and greater or equal to 1 as $delta$ and γ are both natural numbers. The same holds for the second fraction as we assumed that the denominator of the second fraction is the smallest possible gcd value. This finishes the proof.

The following algorithm allows to efficiently compute $BMLB(e)$ according to Eq. (2) due to the above observation.

Input: $pmin(e), pmax(e), cmin(e), cmax(e)$
Output: $BMLB(e)$
$bmlb$ = 1;
for $c = cmax(e)$ downto $cmin(e)$
for $p = pmax(e)$ downto $pmin(e)$
$bmlb' = bmlb(p, c)$;
if $blmb' > bmlb$
$bmlb = bmlb'$;
if $\gcd(p, c) = 1$ break;
od
od
for $p = pmax(e)$ downto $pmin(e)$
for $c = cmax(e)$ downto $cmin(e)$

$bmlb' = bmlb(p, c)$;
if $blmb' > bmlb$
$bmlb = bmlb'$;
if $\gcd(p, c) = 1$ break;
od
od
return(bmlb);

Note that the two loops are not completely identical. Each loop can be quit once we obtain a combination of p and c where their gcd is 1 as this the minimal gcd value possible. The second loop, however, is necessary, as we need a combination of p and e where their product is minimal. The first loop decreases p in the inner loop while the second loop decreases c in the inner loop. The following two examples try to make this procedure clear.

Example: Let $p \in [1, 9]$ and $c \in [1, 5]$. Starting with $c = 5$ and $p = 9$, we obtain immediately a break of the first nested loop block for the values $c = 5$, $p = 9$ already because their gcd is 1. Then, the second loop nest is executed but this loop is also immediately exited. Hence, $BMLB(e) = 5 * 9 = 45$ for this example. Although enumerative, the BLMB computation is quite fast as in most cases, if the intervals are quite large, it is likely that the gcd becomes 1.

Example: Let $p \in [5, 7]$ and $c \in [3, 3]$. Starting with $c = 3$ and $p = 7$, we obtain also a break immediately. Note that although the pair $c' = 3$, $p' = 5$ also has the same and minimal gcd value, we do not have to care because $bmlb = 3 * 7 = 21$ whereas $bmlb' = 3 * 5 = 15$. Hence, p and c determine the maximum und thus $BMLB(e)$.

3.1 Scheduling by PGAN

Clustering [2] by pairwise grouping adjacent nodes is a frequently applied technique in order to obtain single-appearance schedules that minimize data memory requirements. In the original algorithm, a cluster hierarchy is constructed by clustering exactly two adjacent actors at each step. At each clustering step, a pair of adjacent actors is chosen that maximizes the number of times this clustered subgraph may be executed subsequently. Fig. 3 and Fig. 4 show the application of PGAN. For fixed values $p((A, B)) = 1, c((A, B)) = 3$, and $p((E, D)) = 10, c((E, D)) = 2$, PGAN clusters A and B first into cluster Ω shown in Fig. 4a), then C and Ω into a cluster Ω' shown in Fig. 4b), and so on, until the graph condenses into a single node. By traversing the so-created hierarchy from top to bottom, the single-appearance schedule $(2(3A)B(2C))(5D)E)$ is obtained. Bhattacharyya proved that under certain conditions stated in [2], the clustering leads to a schedule satisfying for each arc the BMLB property.

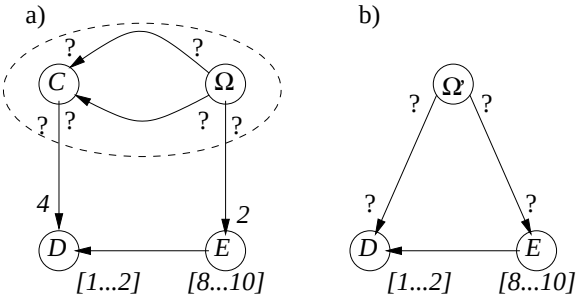

Figure 4. Clustering of an ILDF graph

In the presence of unknown data rates, the application of PGAN to construct *parameterized schedules* has been explored in [1]. In general, a paramterized schedule

is like a looped schedule, except that iteration counts of loops can be symbolic expressions in terms of one or more graph variables. In ILDF, we adapt this concept of parameterized schedule to incorporate intervals for iteration counts that are not known precisely at compile time, but for which lower and upper bounds are known. For example, $A([2,4]B([1,3]C))$ specifies an ILDF parameterized schedule with a nested loop, where the outer loop iteration count ranges from 2 to 4 at runtime, while the inner loop iteration count ranges from 1 to 3.

Even if we do not yet know the meaning of consistency for ILDF graphs, we want to show here how we are able to compute the minimal data memory requirements assuming that a valid schedule is given by clustering, i.e., a given clustering order of adjacent nodes.

The major question on which we want to focus here is therefore the determination of intervals of clustered nodes, see Fig. 4a), b). This problem is addressed in Fig. 5.

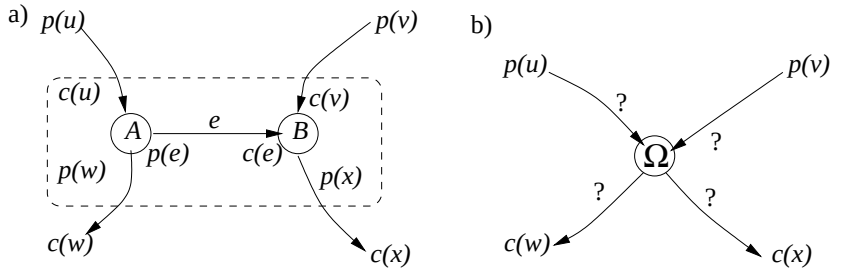

Figure 5. Clustering step

If we condense two nodes into a cluster as indicated in Fig. 5, the production and consumption numbers must be updated.

Theorem 2 (Clustered intervals). *Given an ILDF graph $G(V'E)$ and one arc $e \in E$ which is to be clustered into a cluster node Ω. Let $src(e)$ be the source, $snk(e)$ denote the target node of e. Let*

- $u \in E : snk(u) = src(e) \ \wedge src(u) \neq snk(e)$,
- $v \in E : snk(v) = snk(e) \ \wedge src(v) \neq src(e)$,
- $w \in E : src(w) = src(e) \ \wedge snk(w) \neq snk(e)$,
- $x \in E : src(x) = snk(e) \ \wedge snk(x) \neq src(e)$.

Then by clustering the nodes adjacent to e into a single clustered node Ω, the weights of arcs u, v, w, and x are changed as follows:

- $p'(u) = p(u), c'(u) = \frac{c(u)c(e)}{\gcd(\{p(e),c(e)\})}$, $snk(u) = \Omega$,
- $p'(v) = p(v), c'(v) = \frac{c(v)p(e)}{\gcd(\{p(e),c(e)\})}$, $snk(v) = \Omega$,
- $p'(w) = \frac{p(w)c(e)}{\gcd(\{p(e),c(e)\})}, c'(w) = c(w)$, $src(w) = \Omega$,
- $p'(x) = \frac{p(x)p(e)}{\gcd(\{p(e),c(e)\})}, c'(x) = c(x)$, $src(x) = \Omega$.

The rate intervals of the changed attributes of the arcs of type u, v, w, and x may be computed by computing the minimum and the maximum values over the given previous intervals.

Proof. The first part is to prove that the clustering process leads to the above transformed values $p'(u), c'(u), p'(v), \ldots$. This part is proven in [2]. The claim of the last sentence is an obvious fact.

Example: Consider the ILDF graph in Fig. 3. Assume we cluster actors A and B into a clustered node Ω as shown in Fig. 4a). We have two arcs of category x and one arc of category w. The new values $p'(w)$ and $p'(x)$ are shown in Fig. 6a) for the example values $p((A,B)) = 3, c((A,B) = 2$.

3.2 Local consistency clustering

We may observe that not any arbitrary combination of pairs $p(e)$ and $c(e)$ leads to a consistent behavior. E.g., let $p((A,B)) = 3$, $c((A,B)) = 2$ in Fig. 3. This leads also to the memory lower bound of 6. If we cluster now A and B into a clustered node Ω, then the transformed weights are obtained as shown in Fig. 6a). Obviously, these fixed weights do not allow consistency because if actor Ω fires once, producing 6 tokens on the upper arc (Ω, C), and 6 tokens also on the second arc (Ω, C), whereas C consumes 3 tokens on the first and only token on the second arc, definitely, there is no valid schedule.

On the other hand, if we choose $p((A,B)) = 1, c((A,B)) = 3$ in Fig. 3, we obtain the transformed graph as shown in Fig. 6b). Obviously, the parallel arcs do not provide objections against consisteny here.

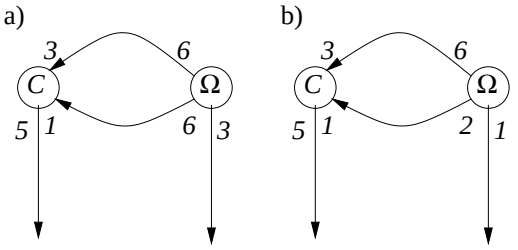

Figure 6. Local consistency violation a) and satisfaction b)

If we condense two nodes into a cluster as indicated in Fig. 5, the production and consumption numbers must be updated. Therefore, in general, if a schedule generated by clustering is given, also the formula for computing the data buffer memory lower bound may be refined by restricting the search only to those pairs of values, that do not create local consistency violations after clustering.

4 Execution time analysis

In the following, we assume given an ILDF graph $G(V,E)$ with corresponding rate intervals and a given looped schedule S, obtained e.g., by PGAN. In order to calculate the influence of timing uncertainty, we may add another property to each actor, the so-called *latency interval*, denoted $l(v) \in [lmin(v)...lmax(v)] \cap \mathbf{N}$ for each actor $v \in V$, see Fig. 7, for instance.

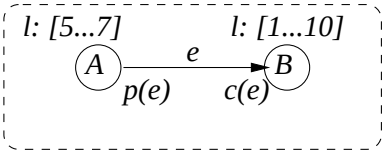

Figure 7. Latency intervals associated to actors

Theorem 3. *The* worst-case execution time *for a given clustering of two actors, e.g., A and B as shown in Fig. 7, and connected by the arc $e = ((A,B))$, is given by the following expression:*

$$WCET((A,B)) = \max_{p(e),c(e)} \{\frac{lmax(A)c(e)}{\gcd(\{p(e),c(e)\})} + \frac{lmax(B)p(e)}{\gcd(\{p(e),c(e)\})}\} \qquad (6)$$

Explanations:

- As the number of firings of each actor A and B that are clustered together into a cluster node Ω is uncertain due to the rate-intervals of the production and consumption numbers, and the sequential computation time is additive in case of sequential execution, we have to calculate the maximum of the sum of the execution times of both.
- If there is no correlation given between the execution time of an actor in dependence on the production and consumption numbers, we get the worst case execution time if we take the maximum of execution time $lmax(A)$, and $lmax(B)$ in order to obtain the WCET.
- For a given clustering, the procedure may be repeated then for the next clustering step by calculating the production and consumption numbers of the clustered graph as indicated by Theorem 2.
- The best case execution time $BCET$ of a two node cluster may be obtained by replacing the max by a min and $lmax$ by $lmin$ in Eq. (6). This leads to a new latency interval $l(\Omega) \in [BCET((A,B)) \ldots WCET\ ((A,B))]$ for the clustered node Ω.

The above procedure might be used in order to define a new variant of PGAN as follows: Instead of clustering those two actors together that have the biggest common repetition factor, we could construct a clustering such to optimize the WCET instead or a combintation thereof.

5 Experiments

In this section, we provide the results of experiments that show that the algorithm BLMB in Section 3 performs well. For doing this, we have created a test series as follows: Let $p \in [pmin, ...pmax]$ and $c \in [cmin, ..., cmax]$ where $pmin, pmax$, $cmin, cmax$ are random numbers in a given interval from 1 to $Z \in \mathbf{N}$. Then, for different values of Z, we generated $N = 1000$ times two random intervals given by $pmin, pmax, cmin, cmax$ and evaluated the number of gcd evaluations performed by algorithm BMLB with respect to an exhaustive search for the maximum using any pair of values inside the random intervals. The following table provides the result of these experiments performed for 5 different values of Z and for $N = 1000$ experiments each. The table lists the average number x of gcd computations of BLMB over N sample intervals in the given range and the average number y of gcd computations in a total search for the maximum value determining the buffer memory lower bound for given two intervals.

Range of intervals	x	y
$[1...10^1]$	1.42	15.50
$[1...10^2]$	1.53	1167.00
$[1...10^3]$	1.57	1.04×10^4
$[1...10^4]$	1.52	1.10×10^7
$[1...10^5]$	1.57	4.21×10^8

Table 1. Average number of gcd computations x performed over $N = 1000$ samples of random intervals in each indicated interval range using the BMLB algorithm and average number of gcd computations y when using exhaustive interval search in the same interval

As can be seen, the average number of gcd computations using the BLMB algorithm is 1.5 and almost constant and independent on the interval range of the experiments. The gain when using the BLMB algorithm is biggest for large intervals. It can thus be seen that buffer memory computations can be done quite efficiently for dataflow graphs with interval firing rates.

6 Related work

Various alternative dataflow modeling strategies with different objectives have been developed for more general or more precise modeling of dataflow graphs beyond synchronous dataflow; a partial review of these approaches is provided here. In cyclo-static dataflow [3], production and consumption rates can be specified as tuples of integers that correspond to distinct execution phases of the incident actors. In scalable synchronous dataflow [8], actor specifications are augmented with vectorization parameters that enable schedulers to control the degree of block processing performed by the actors. In synchronous piggybacked dataflow [7], actors access global states by passing special pointers alongside regular data tokens. In parameterized dataflow [1], dynamic reconfiguration of actor and subsystem parameters is allowed through separation of functionality into subgraphs that perform reconfiguration and subgraphs that are periodically reconfigured. Boolean [4], bounded dynamic [6], and cyclo-dynamic [10] dataflow offer dynamically varying production and consumption rates by incorporating various other data-dependent modeling constructs.

7 Conclusions and future work

We have presented a form of dataflow, called interval-rate, locally-static dataflow (ILDF). In ILDF graphs, the token production and consumption rates on graph edges remain constant throughout execution the graph, but these constants are not known exactly at compile time. We have motivated the use of ILDF as an intermediate representation for an important class of non-deterministic dataflow graphs, and have described a number of application examples. We have analyzed worst-case data memory requirements, and worst- and best-case execution time performance of schedules for ILDF graphs. Many useful directions for further work emerge from this study, including the development of algorithms for constructing efficient uniprocessor and multiprocessor schedules for ILDF graphs, and integration of ILDF concepts to work with other dataflow models of computation, particularly the more dynamic ones.

References

1. B. Bhattacharya and S. S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, 49(10):2408–2421, October 2001.
2. S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.
3. G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, February 1996.
4. J. T. Buck and E. A. Lee. Scheduling dynamic dataflow graphs using the token flow model. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, April 1993.
5. E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.
6. M. Pankert, O. Mauss, S. Ritz, and H. Meyr. Dynamic data flow and control flow in high level DSP code synthesis. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1994.
7. C. Park, J. Chung, and S. Ha. Efficient dataflow representation of MPEG-1 audio (layer iii) decoder algorithm with controlled global states. In *Proceedings of the IEEE Workshop on Signal Processing Systems*, 1999.
8. S. Ritz, M. Pankert, and H. Meyr. Optimum vectorization of scalable synchronous dataflow graphs. In *Proceedings of the International Conference on Application Specific Array Processors*, October 1993.
9. P. P. Vaidyanathan. *Multirate Systems and Filter Banks*. Prentice Hall, 1993.
10. P. Wauters, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-dynamic dataflow. In *EUROMICRO Workshop on Parallel and Distributed Processing*, January 1996.