Journal of VLSI Signal Processing 43, 247–258, 2006

DOI: 10.1007/s11265-006-7274-2

Analysis of Dataflow Programs with Interval-limited Data-rates

JÜRGEN TEICH
University of Erlangen-Nuremberg, Germany

SHUVRA S. BHATTACHARYYA
Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park MD, 20742

Abstract. In this paper, we consider the problem of analyzing dataflow 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. Also, metamodeling formulations of interval limited dataflow are discussed, with special emphasis on the application to cyclo-static dataflow modeling.

Keywords: Scheduling dataflow models of computation, buffer memory analysis, interval rates

1. Motivation

The role of dataflow 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 dataflow 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 dataflow 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 [6], 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 in a valid schedule. Typically, a

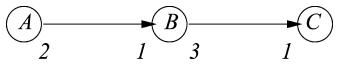

Figure 1. Simple SDF graph.

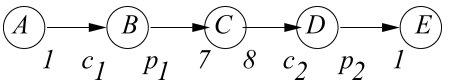

Figure 2. An example of a compact disc to digital audio tape sample rate conversion system that is formulated as an ILDF graph.

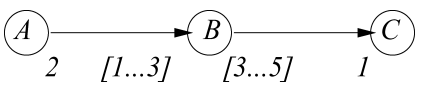

Figure 3. Simple ILDF graph.

(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 successive 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.

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 [11], one might parameterize the data rates of various filters to explore a range of different multirate 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.

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., $p(e) \in [p(e)_{\min} \ldots p(e)_{\max}] \cap \mathbf{N}$,[1] for each arc $e \in E$. Similarly, $c(e) \in [c(e)_{\min} \ldots c(e)_{\max}] \cap \mathbf{N}$. Finally, each arc $e \in E$ has associated a number $delay(e)$, denoting the initial number of tokens on arc e. Again, this number may not be a constant, but instead an interval such that $delay(e) \in [delay(e)_{\min} \ldots delay(e)_{\max}] \cap \mathbf{N}$.[2]

For example, Fig. 3 shows a graph with two intervals associated with the consumption rate of the arc from actor A to B and the production rate on the arc between actor B and C.

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 7 provides a review of related work.

2. Interval-rate SDF

Definition 2.1. 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.,

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

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.[3]

Third, an ILDF 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.

In order to explain how the buffer memory lower bound may be computed for ILDF graphs we use the following example of 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. 10 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 question addressed in the following is how to compute the BMLB for each arc efficiently without having to compute all combinations of $p(e)$ and $c(e)$ separately.

Theorem 3 .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 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 δ 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 bmlb' > 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 bmlb' > 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 maximal gcd value possible. The second loop, however, is necessary, as we need a combination of p and c where their product is maximal at the minimal gcd value. The first loop decreases p in the inner loop while the second loop decreases c in the inner loop. The following three examples try to make this procedure clear.

Example: Let $p \in [1 \ldots 9]$ and $c \in [1 \ldots 5]$. 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 BMLB computation is quite fast as in most cases, if the intervals are quite large, it is likely that the gcd becomes 1.

Example: Let $p \in [5 \ldots 7]$ and $c \in [3 \ldots 3]$. 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 and thus $BMLB(e)$.

The last example shows that the second loop is indeed necessary to correctly compute the BMLB value.

Example: Let $p \in [1 \ldots 3]$ and $c \in [7 \ldots 12]$. Starting with $c = 12$ and $p = 3$, the first loop part calculates

for $c = 12$, $p = 3$ a value of $bmlb' = 36/3 = 12$, then with $c = 12$, $p = 2$ a value of $bmlb = 24/2 = 12$ and finally stops with a break at $c = 12$, $p = 1$ with a value of $bmlb = 12/1 = 12$ because the gcd value of p and c is one. Hence, $bmlb' = 12$ until here. Then, the second nested loop starts with the initial value pair of $p = 3, c = 12$, this time decreasing c. At $p = 3, c = 11$, we obtain $bmlb' = 33/1 = 33$ and obtain again a break with the gcd being one. As can bee seen, it wouldn't have been correct to have stopped with the result of the first nested loop.

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. 4 and Fig. 5 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. 5a), then C and Ω into a cluster Ω' shown in Fig. 5b), 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 $(2(3A)B(2C))(1E)(5D)$ is obtained. Bhattacharyya proved that under certain conditions stated in [2], the clustering leads to a schedule satisfying for each arc the BMLB property.

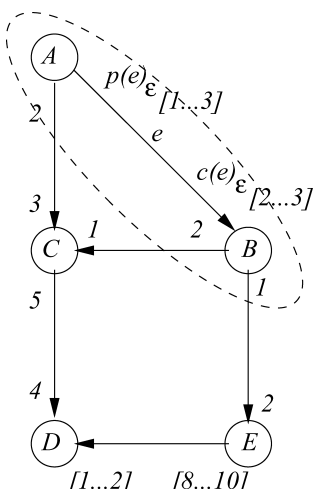

Figure 4. ILDF graph.

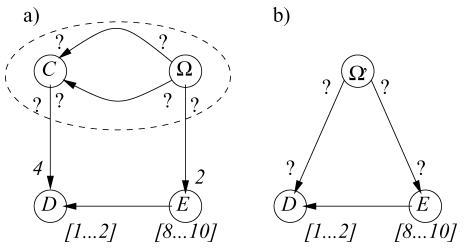

Figure 5. Clustering of an ILDF graph.

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

In this section, we show how 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. 5a), b). This problem is addressed in Fig. 6.

If we condense two nodes into a cluster as indicated in Fig. 6, the production and consumption numbers must be updated.

Theorem 3. 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

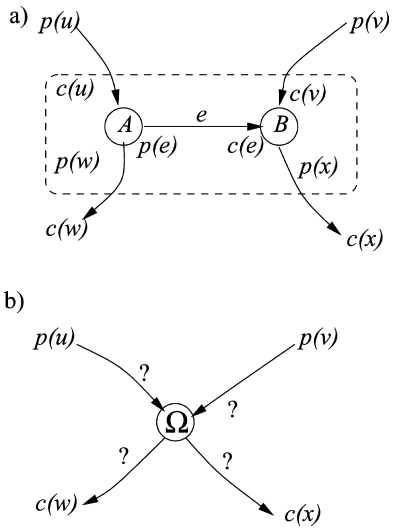

Figure 6. Clustering step.

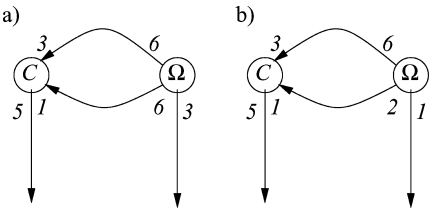

Figure 7. Local consistency violation a) and satisfaction b).

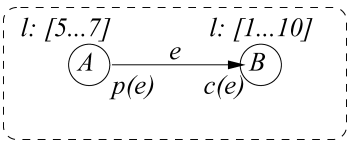

Figure 8. Latency intervals associated to actors.

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. 4. Assume we cluster actors A and B into a clustered node Ω as shown in Fig. 5a). 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. 7a) 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. 10. 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. 7a). 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. 4, we obtain the transformed graph as shown in Fig. 7b). Obviously, the parallel arcs do not provide objections against consistency here.

If we condense two nodes into a cluster as indicated in Fig. 6, 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. 8, for instance.

Theorem 4.1. *The* worst-case execution time *for a given clustering of two actors, e.g., A and B as shown in Fig. 8, and connected by the arc e = ((A, B)), is given by the following expression:*

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

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 3.2.
- 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.

Example: Figure 9 shows an ILDF graph with constant consumption and production rates however with uncertain execution time intervals annotated to each actor. For computing a schedule, we cluster those two nodes together that allow the highest repetition factor. According to PGAN, the first two nodes clustered together are nodes A and B. We obtain

$$WCET((A,B)) = \{\frac{10 \times 3}{\gcd(\{1,3\})} + \frac{3 \times 1}{\gcd(\{1,3\})}\} = 33 = lmax(\Omega)$$

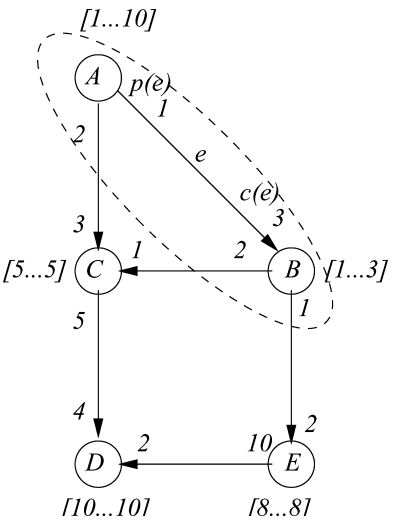

Figure 9. Execution time analysis during clustering.

This means that each invocation of the clustered node Ω resulting from clustering nodes A and B leads to a worst case execution time of 33 time units. Similarly, the best case execution time may be computed as

$$BCET((A,B)) = \{\frac{1 \times 3}{\gcd(\{1,3\})} + \frac{1 \times 1}{\gcd(\{1,3\})}\} = 4 = lmin(\Omega)$$

The clustered graph is shown in Fig. 10a.

Next, the actors Ω and C are clustered together. For this new cluster Ω', the worst case and best case execution times are determined as:

$$WCET((\Omega,C)) = \{\frac{lmax(\Omega) \times 3}{3} + \frac{lmax(C) \times 6}{3}\} = 33 + 10 = 43 = lmax(\Omega')$$

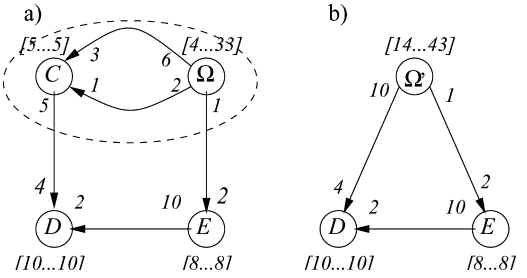

Figure 10. Execution time analysis during clustering.

and

$$BCET((\Omega, C)) = \{\frac{lmin(\Omega) \times 3}{3} + \frac{lmin(C) \times 6}{3}\}$$
$$= 4 + 10 = 14 = lmin(\Omega')$$

Each clustering step, this procedure is repeated resulting in the total WCET of 144 time units for the final schedule $(2\Omega')(1E)(5D)$). One can easily verify that this time amount for the overall schedule is equal to the addition of the worst case execution times of each actor: As A fires 6 times, B 2 times, C four times, D 5 times, and E 1 time, adding up the corresponding values of $lmax$ for each actor leads to $6 \times 10 + 2 \times 3 + 4 \times 5 + 5 \times 10 + 1 \times 8 + 144$.

The above procedure might be used also in order to define a new variant of PGAN that instead of clustering those two actors together that have the biggest common repetition factor, tries to optimize the execution time of a hardware and software target.

5. Experiments

In this section, we provide the results of experiments that show that the algorithm BMLB in Section 3 performs well and secondly, a modified version of PGAN to optimize the execution time of a hardware and software target.

5.1. Complexity of BMLB Calculation

For the first problem, 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 BMLB 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.

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.

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

As can be seen, the average number of gcd computations using the BMLB algorithm is 1.5 and almost constant and independent on the interval range of the experiments. The gain when using the BMLB 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.

5.2. PGAN for Execution Time Minimization

In this section, we show how the well-known PGAN algorithm that optimizes uniprocessor software schedules with the idea to cluster in each step those two actors together that have the highest repetitive factor may be extended easily to obtain, e.g., the best hardware/software partition with respect to execution time minimization.

For doing this, assume in the following for simplicity that the value $lmin(A)$ of actor A in an ILDF graph denotes the execution time obtained in case the actor is implemented in hardware. Similarly, assume that $lmax(A)$ to be the execution time in case A is realized on a software target, e.g., on a microprocessor of a SoC (System-on-a-Chip) implementation. Assume finally that each partition block has a capacity constraint, e.g., determined by the size of hardware available on an SoC.

Example: According to Eq. (6), the worst case and best case execution times obtained after clustering pairs of nodes in the ILDF graph in Fig. 9 are reflected in the following Table 2: PGAN decides which is the pair of nodes that would be best to be put in hardware in order to minimize the overall execution time. For simplicity, we assume here that the required amount of hardware resources needed for each actor is equal (unit cost model), although extensions with non-uniform size parameters are straightforward. Finally, we assume in this example that only two actors do fit simultaneously in the SoC. In case both nodes of a

Table 2. Best case (BCET) and worst case (WCET) values of .

Node pair	BCET	WCET
(A, B)	4	33
(A, C)	13	40
(B, C)	6	13
(B, E)	10	14
(D, E)	58	58
(C, D)	70	70

cluster are implemented in hardware, we assume the BCET values according to Table 2 for execution time estimation. Similarly, we assume the WCET values for the implementation of the cluster in software. This way, the implementation of an actor in hardware can be seen as an acceleration within a schedule.

As can be seen in the above table, clustering the pair (D, E) or (C, D) together and implementing them in hardware would lead to no benefit in the execution time as BCET and WCET values are equal. The best benefit is given by those two nodes which difference between WCET and BCET times the repetition factor of this cluster is maximum. In the example, cluster (A, B) would be executed twice, hence a benefit in execution time of $2 \times (WCET(A, B) - BCET(A, B)) = 2 \times 29 = 58$ time units would arise. The gain for clustering (A, C) to a hardware block would be $2 \times (40 - 13) = 54$ time units. Among all pairs of adjacent nodes in Table 2, clustering A and B leads to the best execution time reduction with respect to the complete software schedule with an execution time for the resulting schedule $(2\Omega(2C))(1E)(5D)$ of $2 \times (4 + 2 \times WCET(C)) + WCET(E) + 5 \times WCET(D) = 28 + 8 + 50 = 86$ as compared to 144 time units in case all actors would be executed in software.

Note that in case more than two nodes are able to be migrated to hardware, the sucessive clustering technique would just continue by creating new clusters added to the hardware partition or extending the previous.

6. Application of Interval Limiting to Other Dataflow Models

So far in this paper, we have considered interval limiting of dataflow graph parameters in synchronous dataflow graphs. Similar limiting of intervals can be applied to other dataflow models, yielding more flexible operation of the associated models. In this sense, the interval limiting modeling concept that we explore in this paper can be viewed as a metamodling technique where interval ranges are applied to unspecified parameter values, or even dynamically varying graph attributes. Of course different analysis techniques will be needed depending on the core dataflow model that is used (so far in this paper, our analysis has used synchronous dataflow), and the form of interval limiting that is applied.

If interval limiting is applied in this manner to a given dataflow model of computation *XYZ*, then we refer to the resulting extended version of the model as *IL-XYZ*. In this paper, we have focused so far on IL-SDF, which we have referred to as ILDF for short.

For example, consider cyclo-static dataflow (CSDF) in which actor execution proceeds in phases, where the number of tokens produced and consumed from input/output ports is constant in each phase, but can vary from one phase to the next. Such variation of token production and consumption is allowed in CSDF as long as the sequence of phases is periodic [3]. More precisely, each actor A in a CSDF graph has associated with it a *fundamental period* $\tau(A) \in \{1, 2, \ldots\}$. This fundamental period specifies the number of phases in one minimal period of the cyclic production/consumption pattern of A. For each input edge e to A, the scalar SDF attribute $c(e)$ is replaced by a $\tau(A)$-tuple $C_{e,1}, C_{e,2}, \ldots, C_{e,\tau(A)}$, where each $C_{e,i}$ is a nonnegative integer that gives the number of data values consumed from e by A in the ith phase of each period of A. Similarly, for each output edge e, the production quantity specifier $p(e)$ is replaced by a $\tau(A)$-tuple $P_{e,1}, P_{e,2}, \ldots, P_{e,\tau(A)}$, which gives the numbers of data values produced in successive phases of A.

Well-developed CSDF-based designs have been shown to have various useful features such as buffering efficiency and compactness of representation, and CSDF semantics have been incorporated into commercial tools for DSP system design.

In IL-CSDF, the number of phases of an actor is in general interval-limited, and furthermore, as in ILDF, the token consumption and production during each phase is also interval-limited. Therefore, for each actor A in an IL-CSDF graph, we have statically-known lower and upper limits, $\tau_l(A)$ and $\tau_u(A)$, respectively, such that $\tau(A) \in [\tau_l(A) \ldots \tau_u(A)] \cap \mathbf{N}$.

Furthermore for each possible phase index $i \in [1, 2, \ldots, \tau_u(A)]$, we in general have minimum and maximum numbers of tokens produced during phase i for each edge incident to A. Thus, for each input

edge e to A, and each possible phase index $i \in [1, 2, \ldots, \tau_u(A)]$, there are upper and lower bounds $C_{i,l}(e, A)$ and $C_{i,u}(e, A)$ such that the number of tokens consumed by A from e during phase i is always guaranteed to lie in the interval $[C_{i,l}(e, A) \ldots C_{i,u}(e, A)]$

Similarly, for each output edge e of A, and each phase index $i \in [1, 2, \ldots, \tau_u(A)]$, there are known bounds $P_{i,l}(e, A)$ and $P_{i,u}(e, A)$ such that the number of tokens produced by A onto e during phase i is always guaranteed to lie within $[P_{i,l}(e, A) \ldots P_{i,u}(e, A)]$.

This formulation of IL-CSDF therefore supports both interval-limited numbers of phases as well as interval-limited token transfer with each phase.

An example of a useful IL-CSDF actor that involves an integer-limited number of phases is an IL-CSDF downsampler. In IL-CSDF, a downsampler actor D with downsampling factor k generally involves upper and lower phase count specifiers $\tau_l D$ and $\tau_u D$, which are based on the minimum and maximum downsampling rates for the actor. The token consumption pattern is simply one token consumed on every phase. The token production pattern is of the form $(1, 0, 0, \ldots, 0)$, where one token is produced every k phases. Therefore, for the input edge e_I of D, we have $C_{i,l}(e_I, D) = C_{i,u}(e_I, D) = 1$ for every phase index i. On the other hand, for the output edge e_O, we have that $P_{i,l}(e_O, D) = P_{i,u}(e_O, D) = 1$ for $i = 1$, while $P_{i,l}(e_O, D) = P_{i,u}(e_O, D) = 0$ for $i \neq 1$.

Similar to this description of the IL-CSDF downsampler functionality, a characterization can be developed for an IL-CSDF actor for interval-limited upsampling.

A general class of useful IL-CSDF actors are those that operate on vector data where the vectors are passed as successive tokens along edges, and the vector lengths are not known beforehand but are known to be constrained within intervals. Such actors will usually exploit the capability for interval-limited production and consumption within a given phase, and may also utilize the IL-CSDF provision for having interval-limited numbers of phases. For example, IL-CSDF actors for upsampling and downsampling of vector data will, in their most general form, require both interval ranges for the numbers of phases (based on the upsampling and downsampling rates), and for the token transfer within a given phase (based on the vector length).

As with ILDF graphs, edge delays can also have interval ranges associated with them. Since these delays relate entirely to the initial state of the inter-actor communication links rather than the execution behavior of actors, delays in IL-CSDF are characterised in the same way as ILDF; i.e., the phased behavior inherent in CSDF and IL-CSDF does not affect the characterization of IL-CSDF delays.

Extensions of the consistency concepts discussed earlier in this paper can be developed to formulate inherent consistency, conditional consistency, and inherent inconsistency specifications of IL-CSDF graphs. Furthermore, it is promising to adapt meta-modeling techniques, such as parameterized dataflow, that are compatible with CSDF (e.g., see [1] for elaboration on the utility of integrating CSDF and parameterized dataflow) to exploit the interval information available in IL-CSDF models. Adaptations of heuristics such as APGAN to IL-CSDF can also be envisioned where interval information can be used in the process of comparing alternative clustering candidates throughout the iterative, pairwise clustering process.

Exploration of modeling applications and analysis techniques for interval-limited enhancements of cyclo-static dataflow and other important dataflow models of computation is a broad and useful direction for further investigation. Among existing dataflow modeling approaches, those that are most promising for integration with the interval limiting concepts developed in this paper include cyclo-static dataflow, as discussed above; multi-dimensional dataflow [7]; parameterized dataflow [1]; and blocked dataflow [5].

7. 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 [10], 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 [9], 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 [8], and cyclo-dynamic [12] dataflow offer dynamically varying production and consumption rates by incorporating various other data-dependent modeling constructs.

8. 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.

Acknowledgments

Jürgen Teich was supported in this work by the Deutsche Forschungsgemeinschaft (DFG) under grant Te163/5-2. Shuvra S. Bhattacharyya 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).

Notes

1. Let **N** denote the set of the natural numbers in the following and $\mathbf{N}_0$ the natural numbers including 0.
2. A formal definition of an ILDF graph will be given in Section 2.
3. Note that this holds even for constant values of consumption and production rates because it is known that no actor in such a cycle will ever receive tokens that enable it to fire. Hence, no actor within this cycle will ever be enabled to fire.
4. Note that the value $BMLB(e)$ is given as the lcm (least common multiple) of the values $p(e)$ and $c(e)$.

References

1. B. Bhattacharya and S. S. Bhattacharyya. "Parameterized dataflow modeling for DSP systems." *IEEE Transactions on Signal Processing*, 49(10), 2001, pp. 2408–2421.
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), 1996, pp. 397–408.
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*, 1993.
5. D. Ko and S. S. Bhattacharyya. "Modeling of block-based DSP systems." in *Proceedings of the IEEE Workshop on Signal Processing Systems*, pages 2003, pp. 381–386, Seoul, Korea.
6. E. A. Lee and D. G. Messerschmitt. "Synchronous dataflow." *Proceedings of the IEEE*, 75(9), 1987, pp. 1235–1245.
7. P. K. Murthy and E. A. Lee. "Multidimensional synchronous dataflow." *IEEE Transactions on Signal Processing*, 50(8), 2002, pp. 2064–2079.
8. 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.
9. 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.
10. 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*, 1993.
11. P. P. Vaidyanathan. "Multirate Systems and Filter Banks." *Prentice Hall,* 1993.
12. P. Wauters, M. Engels, R. Lauwereins, and J. A. Peperstraete. "Cyclo-dynamic dataflow." in *EUROMICRO Workshop on Parallel and Distributed Processing*, 1996.

Jürgen Teich received his masters degree (Dipl.-Ing.) in 1989 from the University of Kaiserslautern (with honours).

From 1989 to 1993, he was PhD student at the University of Saarland, Saarbrücken, Germany from where he received his PhD degree (summa cum laude). His PhD thesis entitled 'A Compiler for Application-Specific Processor Arrays' summarizes his work on extending techniques for mapping computation intensive algorithms onto dedicated VLSI processor arrays.

In 1994, Dr. Teich joined the DSP design group of Prof. E. A. Lee and D.G. Messerschmitt in the Department of Electrical Engineering and Computer Sciences (EECS) at UC Berkeley where he was working in the Ptolemy project (PostDoc).

From 1995 to 1998, he held a position at Institute of Computer Engineering and Communications Networks Laboratory (TIK) at ETH Zürich, Switzerland, finishing his Habilitation entitled 'Synthesis and Optimization of Digital Hardware/ Software Systems' in 1996.

From 1998 to 2002, he was full professor in the Electrical Engineering and Information Technology department of the University of Paderborn, holding a chair in Computer Engineering.

Since 2003, he is appointed full professor in the Computer Science Institute of the Friedrich-Alexander University Erlangen-Nuremberg holding a chair in Hardware-Software-Co-Design.

Dr. Teich has been a member of multiple program committees of well-known conferences and workshops. He is member of the IEEE and author of a textbook on Co-Design edited by Springer in 1997.

His research interests are massive parallelism, embedded systems, Co-Design, and computer architecture.

Since 2004, Prof. Teich is also an elected reviewer for the German Science Foundation (DFG) for the area of Computer Architecture and Embedded Systems. Prof. Teich is involved in many interdisciplinary national basic research projects as well as industrial projects. He is supervising 19 PhD students currently.

Shuvra S. Bhattacharyya is an associate professor in the Department of Electrical and Computer Engineering and the Institute for Advanced Computer Studies (UMIACS) at the University of Maryland, College Park. He is also an affiliate associate professor in the Department of Computer Science. Dr. Bhattacharyya is coauthor or coeditor of four books and the author or coauthor of more than 100 refereed technical articles. His research interests include VLSI signal processing, embedded software, and hardware/software co-design. He received the B.S. degree from the University of Wisconsin at Madison, and the Ph.D. degree from the University of California at Berkeley. Dr. Bhattacharyya has held industrial positions as a Researcher at the Hitachi America Semiconductor Research Laboratory (San Jose, California), and as a Compiler Developer at Kuck & Associates (Champaign, Illinois).