



Combining Extended Retiming and Unfolding for Rate-Optimal Graph Transformation

TIMOTHY W. O'NEIL

Department of Comp. Science, University of Akron, Akron, OH 44325-4003

EDWIN H.-M. SHA

*Department of Comp. Science, Erik Jonsson School of Eng. & C.S., Box 830688, MS EC 31,
University of Texas at Dallas, Richardson, TX 75083-0688*

Received December 1, 2003; Revised December 1, 2003; Accepted May 7, 2004

Abstract. Many computation-intensive iterative or recursive applications commonly found in digital signal processing and image processing applications can be represented by *data-flow graphs* (DFGs). The execution of all tasks of a DFG is called an *iteration*, with the average computation time of an iteration the *iteration period*. A great deal of research has been done attempting to optimize such applications by applying various graph transformation techniques to the DFG in order to minimize this iteration period. Two of the most popular are *retiming* and *unfolding*, which can be performed in tandem to achieve an optimal iteration period. However, the result is a transformed graph which is much larger than the original DFG. To the authors' knowledge, there is no technique which can be combined with minimal unfolding to transform a DFG into one whose iteration period matches that of the optimal schedule under a pipelined design. This paper proposes a new technique, *extended retiming*, which does just this. We construct the appropriate retiming functions and design an efficient retiming algorithm which may be applied directly to a DFG instead of the larger unfolded graph. Finally, we show through experiments the effectiveness of our algorithms.

Keywords: scheduling, data-flow graphs, retiming, unfolding, graph transformation, timing optimization

1. Introduction

For real-time or computation-intensive applications such as DSP, image processing and simulations for fluid dynamics, it is important to optimize the execution rate of a design. Because the most time-critical parts of such applications are loops, we must explore the parallelism embedded in the repetitive pattern of a loop. A loop can be modeled as a *data-flow graph* (DFG) [1, 2]. The nodes of a DFG represent tasks, while edges between nodes represent data dependencies among tasks. Each edge may contain a number of *delays* (i.e. registers) indicating loop-carried dependencies. This model is widely used in many fields, including circuitry [3],

digital signal processing [4] and program descriptions [5, 6].

To meet the desired throughput, it becomes necessary to use multiple processors or multiple functional units. In our previous work [7–10], we proposed an efficient algorithm, *extended retiming*, which transforms a DFG into an equivalent graph with maximum parallelization and minimum schedule length whenever the DFG meets certain very narrow criteria. Therefore, there remain applications for which our original framework will not deliver the best possible result. We wish to correct this exclusion in this paper.

The execution of all tasks of a DFG is called an *iteration*. A very popular strategy for maximizing

parallelism is to transform the original graph by scheduling multiple iterations simultaneously, a technique known as *unfolding* [11]. While the graph becomes much larger, the average computation time of an iteration (the *iteration period*) can be reduced. In our previous work, we demonstrated that extended retiming allows us to achieve an optimal iteration period when the iteration period is an integer. In this paper, we refine our original scheme so that extended retiming may be combined with unfolding. We then show that this combination achieves optimality for all cases. In fact, we will see that this combination attains an optimal result while doing a minimal amount of unfolding. We find that the combination of traditional retiming and unfolding does not correctly characterize the implementation using a pipelined design and, therefore, tends to give a large unfolding factor. Thus we not only maximize parallelism by using extended retiming, but we also minimize the size of the necessary transformed graph.

In addition to unfolding, one of the more effective graph transformation techniques is *retiming*, where delays are redistributed among the edges so that the function of the DFG G remains the same, but the length of the longest zero-delay path (the *clock period* of G , denoted $cl(G)$) is decreased. This technique was introduced in [3] to optimize the throughput of synchronous circuits, and has since been used extensively in such diverse areas as software pipelining [12, 13] and hardware-software codesign [14, 15]. We have shown previously that this traditional form of retiming [7] cannot produce optimal results when applied individually. The same is true for unfolding (as we will see), but the combination of traditional retiming and unfolding will achieve optimality [1].

To illustrate these ideas, consider the example of Fig. 1(a). The numbers inside the nodes represent computation times. The short bar-lines cutting the edge from node C to node B (hereafter referred to by the ordered pair (C, B)) represent inter-iteration dependencies between these nodes. In other words, the two lines cutting (C, B) tell us that task B of our current iteration depends on data produced by task C two iterations ago. This representation of such a dependency is called a *delay* on the edge of the DFG.

It is clear that the clock period of this graph is 14, obtained from the path from A to C . Since an iteration of the DFG may be scheduled within 14 time units as in Fig. 1(b), the iteration period of this graph is also 14. However, if we were to remove a delay from (C, A)

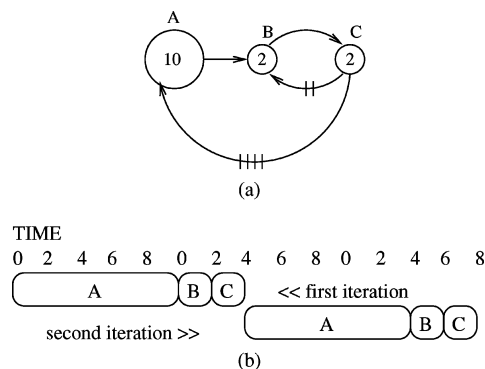


Figure 1. (a) A data-flow graph; (b) The schedule for the DAG part of Fig. 1(a).

and place it on (A, B) , the iteration period (i.e. clock period) would be reduced to 10 while not affecting the function of the graph. (This assumes the existence of multiple processors within the system. Indeed, we will not consider resource-constrained scheduling in this paper.) The example shows how retiming may be used to adjust the iteration period of a DFG.

How small can we make our iteration period? Since retiming preserves the number of delays in a cycle, the ratio of a cycle's total computation time to its delay count remains fixed regardless of retiming. The maximum of all such ratios, called the *iteration bound* [16], acts as a lower bound on the iteration period. In the case of Fig. 1(a), there are only two cycles, the small one between nodes B and C with time-to-delay ratio $\frac{4}{2} = 2$, and the large one involving all nodes with ratio $\frac{14}{4}$. Thus the iteration bound for the graph is $\frac{7}{2}$.

Since the computation times of all nodes are integral, it seems impossible to get a fractional iteration period. However, recall that the iteration period is the *average* time to complete an iteration. If we can complete two iterations of our graph in 7 time units, the average will equal our lower bound, and our graph will be rate-optimal. To get these iterations together in our graph, we must unfold the graph. If we can unfold our graph f times to achieve this lower bound, our schedule is said to be *rate-optimal*, and f is called a *rate-optimal unfolding factor*. We are interested in finding the minimum such unfolding factor.

As an example, let us unfold the graph of Fig. 1(a) by a factor of 2, as shown in Fig. 2(a). (We will discuss our algorithm for doing this in detail later.) We can schedule an iteration of this new graph—which is equivalent to scheduling two iterations of our original graph—in the same 14 time units. We have doubled the size of our

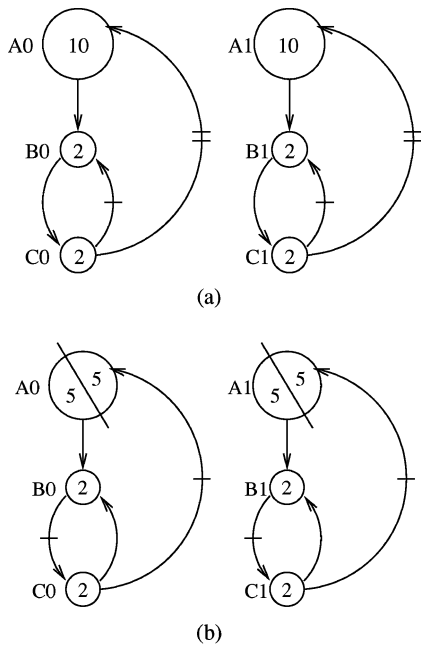


Figure 2. (a) The DFG of Fig. 1(a) unfolded by a factor of 2; (b) Figure 2(a) retimed by extended retiming to be rate-optimal; (c) The optimal schedule for the retimed graph.

graph, but we have also reduced our iteration period to 7. We can now retime this unfolded graph as we did above to reduce our clock period to 10, which further reduces the iteration period to 5. Unfortunately this is the best we can do by unfolding twice and using traditional retiming.

If we were permitted to move a delay inside of A, as shown in Fig. 2(b), our clock period would become 7, the iteration period would become $\frac{7}{2}$ and we would have optimized our graph, as we can see by the schedule in Fig. 2(c). This is the advantage of extended retiming over traditional retiming: we are allowed to move delays not only from edge to edge, but from edge to vertex. We see from this that the combination of traditional retiming and unfolding does not completely give

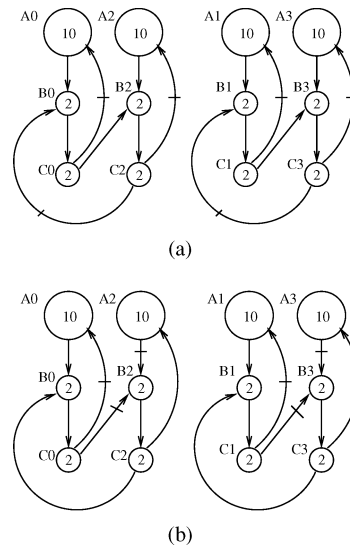


Figure 3. (a) The DFG of Fig. 1(a) unfolded by a factor of 4; (b) Figure 3(a) retimed by traditional retiming to be rate-optimal.

the correct representation of the graph’s schedule, especially when we assume a pipelined implementation.

We should note that, when we talk about moving delays inside of nodes, we do not mean that we are physically placing registers inside of functional units in a circuit. We are merely describing an abstraction for a graph which provides for a feasible schedule with loop pipelining, as this example shows. Traditional retiming is not powerful enough to capture such a schedule, and so may produce an inferior result.

An unfolding of 2 combined with extended retiming optimizes the graph of Fig. 1(a). If we limit ourselves to traditional retiming, we must unfold the original DFG four times, as shown in Fig. 3(a). After we retime as in Fig. 3(b) in addition to unfolding, we can now schedule 4 iterations of the original graph in 14 time steps, reducing our iteration period without retiming to $\frac{7}{2}$. We see that traditional retiming tends to overestimate the rate-optimal unfolding factor, resulting in a graph that requires more resources for execution.

Note that the graph optimized by extended retiming and unfolding is half the size of that optimized by traditional retiming and unfolding. There is a very clear advantage in using extended retiming, but there are currently two drawbacks to this method:

1. As proposed, extended retiming only permits the placement of a single delay inside any node. This is too severe a limitation for what we want to do now.

For instance, note that if we had been allowed to remove two delays from (C, A) in Fig. 1(a) and use them to split node A into three pieces of sizes 4, 4 and 2, we would have achieved an almost optimal iteration period of 4 even before unfolding.

2. The only method for applying extended retiming and unfolding is the one we've outlined: unfold the graph and then retime. Since retiming is much more expensive than unfolding in terms of computation time, it is preferable to first apply extended retiming to the smaller original graph, then unfolding. However, we must be certain that the two operations can be performed in any order and still achieve optimality.

We also have the question of knowing exactly how much to unfold a graph before it can be optimized. As we've said, unfolding dramatically increases the size of the graph we're working with, so we don't want to do any more unfolding than is absolutely necessary.

We accomplish the following in this paper:

1. We will demonstrate a new form of retiming, *extended retiming*, which achieves an optimal result while requiring the use of a smaller unfolded graph, and thus fewer resources.
2. Our original definition of extended retiming was constructed with the assumption that at most one delay may be placed within a given node. We now modify this definition to accommodate the possibility that multiple delays are placed inside a node. This permits us to combine extended retiming with unfolding.
3. When we wish to apply unfolding and extended retiming to a graph, we have two options: first retime the graph then unfold it, or unfold it then retime the unfolded graph. We will show that these two methods are equivalent and construct the corresponding retiming functions.
4. Because of this equivalence, we are able to design an efficient extended retiming algorithm which can be applied directly to the original graph. In fact, we will show that we can construct an extended retiming for a graph G in $O(|V||E|)$ time where V and E are the vertex and edge sets for G , respectively. This improves our previous work, the application of which could produce a retiming function only for the larger unfolded graph.
5. Finally, we will demonstrate that the minimum rate-optimal unfolding factor for a data-flow graph is the denominator of the irreducible form of the graph's

iteration bound. Thus we have devised a technique that, when combined with unfolding by the minimum rate-optimal unfolding factor, transforms a graph into one whose iteration period matches that of the rate-optimal schedule. To the best of our knowledge, this is the first method that can do this.

The rest of the paper is organized as follows. In the next section, we will formalize fundamental concepts such as data-flow graphs, unfolding and clock period. The theme of this paper is presented next, along with some introductory results. Sections 4 and 5 contain the paper's most significant result, the proof of the interchangeability of extended retiming and unfolding. We then design our efficient algorithm and show its effectiveness when applied to graphs whose iterations bounds are one or larger, which encompasses all non-trivial examples. The minimal rate-optimal unfolding factor for a DFG is determined next, followed by detailed examples of this work. Finally, we summarize our work and provide questions for further study.

2. Background

In this section, we wish to present the definitions and results relating to unfolding and unfolded graphs. We will rely on this previously-presented background material [17, 11] heavily as we establish our new results.

2.1. Unfolding and Unfolded Graphs

Formally, a *data-flow graph* (DFG) is a finite, directed, weighted graph $G = \langle V, E, d, t \rangle$ where V is a set of computation nodes, E is a set of edges between nodes, $d : E \rightarrow \mathbf{N}$ is a function representing the delay count of each edge, and $t : V \rightarrow \mathbf{N}$ representing the computation time of each node. A *path* in G is a connected sequence of nodes and edges. We will use the notations $D(p)$ and $T(p)$ to represent the total computation time and total delay count of path p , respectively.

Now, let f be a positive integer. We wish to alter our graph so that f consecutive *iterations* (i.e. executions of all of a DFG's tasks) are visible simultaneously. To do this, we create f copies of each node, replacing node u in the original graph by the nodes u_1 through u_f in our new graph. This process is known as *unfolding* the graph G f times and results in the *unfolded graph* $G_f = \langle V_f, E_f, d_f, t_f \rangle$. The unfolding transformation was introduced in [11] and has been widely discussed

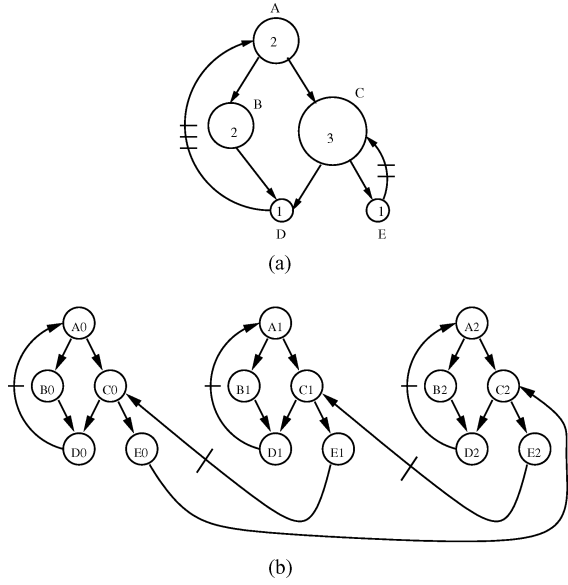


Figure 4. (a) A sample DFG; (b) This DFG unfolded by a factor of 3.

in the literature. As a brief example, if we unfold the graph in Fig. 4(a) by a factor of 3, the result is Fig. 4(b).

As we can see from this example, the vertex set V_f is simply the union of the f copies of each node in V . Since they are all exact copies, the computation times remain the same, i.e. $t_f(u_f) = t(u)$ for every copy u_f of $u \in V$. Each edge of G also corresponds to f copies in the unfolded graph. However, the delay counts of the copies do not match that of the original edge. In fact, an edge (u_i, v_j) having d delays in the unfolded graph represents a precedence relation between node u in the i th iteration and node v in iteration $d \cdot f + j$ in the original graph. This idea is formalized in the following theorem, which is proven in [1].

Theorem 2.1. Let $e = (u, v)$ be an edge in DFG G . Let f be an unfolding factor for G . Then:

1. For all $i, j \in \{0, 1, 2, \dots, f-1\}$, there exists an edge $e_f = (u_i, v_j)$ in G_f if and only if $d(e) = d_f(e_f) \cdot f + j - i$.
2. For all integers $i, j \in \{0, 1, 2, \dots, f-1\}$ with $j \equiv (i + d(e)) \pmod f$, there exists an edge $e_f = (u_i, v_j)$ in G_f with $d_f(e_f) = \lfloor \frac{d(e)}{f} \rfloor$ if $i \leq j$ and $\lceil \frac{d(e)}{f} \rceil$ otherwise.
3. The f copies of edge e in G_f are the edges $e_i = (u_i, v_{(i+d(e)) \pmod f})$ for $i = 0, 1, 2, \dots, f-1$.
4. The total number of delays of the f copies of edge e is $d(e)$, i.e. $d(e) = \sum_{i=0}^{f-1} d_f(e_i)$.

Despite implications in the literature to the contrary [11], it is possible to construct a DFG which cannot be unfolded to become rate-optimal. For example, consider the unit-time DFG of Fig. 5(a) with iteration bound 2. When unfolded by a factor of $f > 1$, it becomes the graph of Fig. 5(b). We see that there is a zero-delay path consisting of the nodes $A_{f-1}, B_{f-1}, C_{f-1}, B_{f-2}, C_{f-2}, \dots, B_0, C_0$ with computation time $2 \cdot f + 1$. Hence the iteration period of the unfolded graph will always be greater than or equal to $2 + \frac{1}{f}$, which is always strictly larger than the iteration bound. This shows that retiming is necessary for achieving an optimal result.

Finally, a classic result from [3] characterized the upper bound of a graph's cycle period in terms of the computation time of its longest zero-delay path. The analogous result for an unfolded graph is proven in [1].

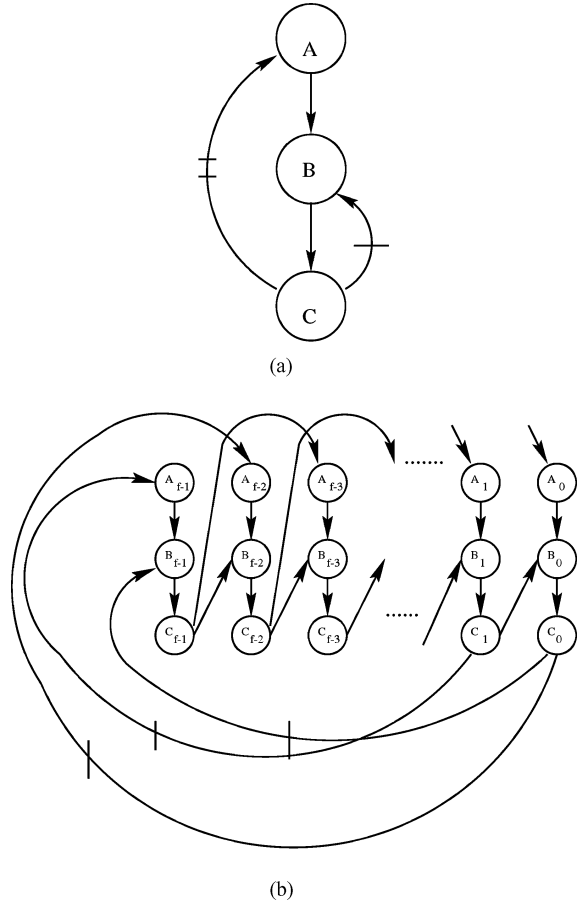


Figure 5. (a) A unit-time DFG; (b) This DFG unfolded by a factor of f .

Theorem 2.2. Let G be a DFG, c a potential cycle period and f an unfolding factor.

1. $cl(G_f) = \max\{T(p) : p \in G \text{ is a path with } D(p) < f\}$.
2. $cl(G_f) \leq c$ if and only if, for every path p in G with $T(p) > c$, $D(p) \geq f$.

2.2. Static Scheduling

Given a DFG G , a clock period c and an unfolding factor f , we construct the scheduling graph $G^s = \langle V, E, w, t \rangle$ by reweighting each edge $e = (u, v)$ according to the formula $w(e) = d(e) - \frac{f}{c} \cdot t(u)$. We then further alter G^s by adding a node v_0 and zero-weight directed edges from v_0 to every other node in G . Figure 6(b) shows the scheduling graph of the example in Fig. 6(a) when $c = 3$ and $f = 1$. It can be shown that, if $\frac{c}{f}$ is a feasible iteration period, then the scheduling graph contains no negative-weight cycles. Define $sh(v)$ for every node v to be the length of the shortest path from v_0 to v in this modified G^s . For example, in the graph of Fig. 6(b), we note that $sh(A) = 0$ and

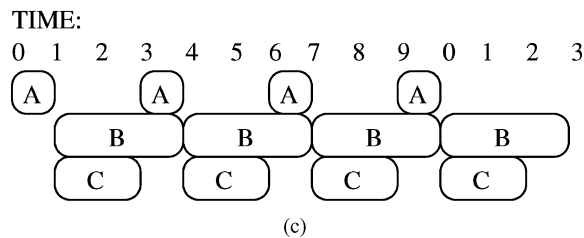
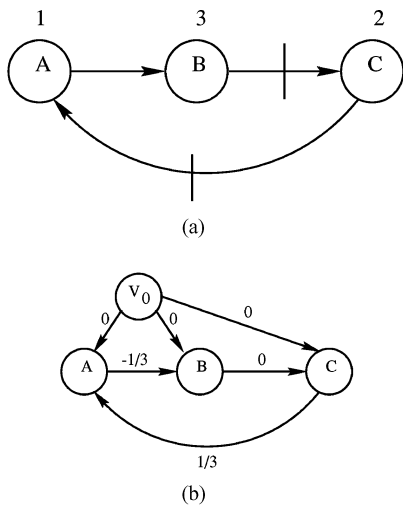


Figure 6. (a) A DFG; (b) The scheduling graph with $c = 3$ and $f = 1$; (c) The schedule with cycle period 3.

$sh(B) = sh(C) = -\frac{1}{3}$. It takes $O(|V||E|)$ time to compute $sh(v)$ for every node v [18].

The definitions for *iteration*, *iteration period*, *iteration bound* and *rate-optimal* have been given previously. Note that Fig. 6(c) demonstrates one rate-optimal schedule for the DFG in Fig. 6(a). The relationship between the iteration bound of a DFG and the DFG's scheduling graph is given in Lemma 3.1 of [18]:

Lemma 2.3. Let G be a DFG, c a clock period and f an unfolding factor. $B(G) \leq \frac{c}{f}$ if and only if the scheduling graph G^s contains no cycles having negative weight.

We formally define an *integral schedule* on a DFG G to be a function $s : V \times \mathbf{N} \rightarrow \mathbf{Z}$ where the starting time of node v in the i th iteration ($i \geq 0$) is given by $s(v, i)$. It is a *legal schedule* if $s(u, i) + t(u) \leq s(v, i + d(e))$ for all edges $e = (u, v)$ and iterations i . For example, the legal integral schedule of Fig. 6(c) is $S_3(v, i) = 3(i - sh(v))$ for all nodes v and iterations i , where the values for $sh(v)$ are derived from the graph of Fig. 6(b).

A legal schedule is a *repeating schedule for cycle period c and unfolding factor f* if $s(v, i + f) = s(v, i) + c$ for all nodes v and iterations i . It's easy to see that $S_3(v, i)$ is an example of a repeating schedule. A repeating schedule can be represented by its first iteration, since a new occurrence of this partial schedule can be started at the beginning of every interval of c clock ticks to form the complete legal schedule. If an operation of the partial schedule is assigned to the same processor in each occurrence of the partial schedule, we say that our schedule is *static*.

Since the iteration bound for the graph of Fig. 6(a) is 3, and all nodes of this graph are 3 or smaller, Theorems 2.3 and 3.5 of [18] tell us that the minimum achievable cycle period for this graph is 3. We can produce the static DFG schedule in Fig. 6(c) by constructing the scheduling graph and computing $sh(v)$ for each of the nodes. We can then use this information to create the schedule of Fig. 6(c) by applying the formula from the above discussion to create the schedule; for this example $S_3(A, 0) = 0$ and $S_3(B, 0) = S_3(C, 0) = 3 \cdot \frac{1}{3} = 1$.

3. Extended Retiming

In a previous paper [7], we demonstrated that traditional retiming and unfolding did not necessarily result in an optimal schedule when applied individually, and

devised a form of retiming which was equivalent to DFG scheduling. This definition and work depended on the assumption that unfolding did not take place, so any node of a graph could be split at most once.

We demonstrated in the Introduction that, while the combination of traditional retiming and unfolding always yields an optimal result, the combination of extended retiming and unfolding yields the same result in many cases while requiring the use of a much smaller graph. We now devise a new, more general form of our extended retiming that will work even when used in conjunction with unfolding. The definition used in [7] then becomes a special case of these results.

An *extended* (or *f*-*extended*) *retiming* of a DFG $G = \langle V, E, d, t \rangle$ is a function $r : V \rightarrow \mathbf{Z} \times \mathbf{Q}^f$ where, for all $v \in V$, $r(v) = i + (\frac{r_1}{t(v)}, \frac{r_2}{t(v)}, \dots, \frac{r_f}{t(v)})$ for some integers i, r_1, r_2, \dots, r_f where $0 \leq r_k < t(v)$ for $k = 1, 2, \dots, f$. We view the integer constant i as the number of delays that are pushed to each outgoing edge of v , while the f -tuple lists the positions of delays within the node v . Note that a value of zero within the f -tuple is merely a placeholder used to simplify our notation; we can't have a delay at this position. Also for simplicity we will express the f -tuple as $\frac{1}{t(v)}(r_1, r_2, \dots, r_f)$ or as a single fraction $\frac{r_i}{t(v)}$ when $f = 1$.

We can see from this definition that $r(v)$ can be viewed as consisting of an integer part and a fractional part. We will use the notation $\iota_r(v)$ to denote the value of this integer part, while $\mathfrak{N}_r(v)$ will be the number of non-zero coordinates in the f -tuple. We will also assume throughout this paper that the elements of an f -tuple are listed in increasing order.

For example, consider the graph of Fig. 7(a). A 3-extended retiming with $r(A) = 1 + \frac{1}{9}(0, 2, 5)$, $r(B) = 1$ and $r(C) = 0$ results in the retimed graph of Fig. 7(b). For this retiming, $\iota_r(A) = \iota_r(B) = 1$ and $\iota_r(C) = 0$, while $\mathfrak{N}_r(A) = 2$ and $\mathfrak{N}_r(B) = \mathfrak{N}_r(C) = 0$. Simple as-early-as-possible (AEAP) scheduling applied to Fig. 7(b) yields Fig. 7(c).

As with standard retiming, we will denote the DFG retimed by r as $G_r = \langle V, E, d_r, t \rangle$. When we define the delay count of the edge $e = (u, v)$ after retiming, we must remember to include delays within each end-node as well as delays along the edge itself. We also want the new definition to be analogous to our traditional one: the old delay count of the edge ($d(e)$), *plus* the number of delays drawn from each incoming edge ($\iota_r(u) + \mathfrak{N}_r(u)$), *minus* the number of delays pushed to each outgoing edge ($\iota_r(v)$).

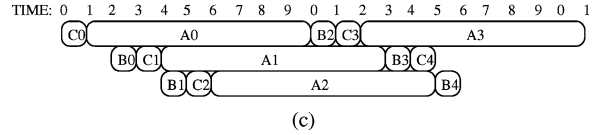
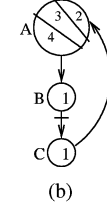
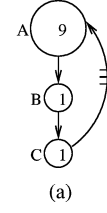


Figure 7. (a) Another sample DFG; (b) This DFG retimed; (c) The AEAP schedule for Fig. 7(b).

Previously, we defined a path p to be a connected sequence of nodes and edges, with $D(p)$ being the path's total delay count. If we now require $D_r(p)$ to count the delays both among the nodes and along the edges of p , we can easily obtain these properties:

Lemma 3.1. *Let G be a DFG without split nodes and r an extended retiming.*

1. *The retimed delay count on the edge $e = (u, v)$ is $d_r(u \rightarrow v) = d(e) + \iota_r(u) - \iota_r(v) + \mathfrak{N}_r(u)$.*
2. *The retimed delay count on the path $p : u \Rightarrow v$ is $D_r(u \Rightarrow v) = D(p) + \iota_r(u) - \iota_r(v) + \mathfrak{N}_r(u)$.*
3. *The retimed delay count on the cycle $\ell \in G$ is $D_r(\ell) = D(\ell)$.*

Given an edge $e = (u, v)$, we use $d_r(u \rightarrow v)$ to denote the total number of delays along an edge, including delays contained within the end nodes u and v . However, we will refer to the number of delays on the edge *not including* delays within end nodes as $d_r(e)$ as in the traditional case. Using the example from Fig. 7(b), let $e_1 = (A, B)$, $e_2 = (B, C)$ and $e_3 = (C, A)$. Then $d_r(A \rightarrow B)$ and $d_r(C \rightarrow A)$ are each 2 due to a split end-node, even though $d_r(e_1)$ and $d_r(e_3)$ are each zero, while $d_r(B \rightarrow C) = d_r(e_2) = 1$ since there is no split end-node for this edge. As with traditional

retiming, an extended retiming is *legal* if $d_r(e) \geq 0$ for all edges $e \in E$ and *normalized* if $\min_v t_r(v) = 0$. Note two things:

1. If r is an extended retiming, then $d_r(u \rightarrow v) = d_r(e) + \mathfrak{R}_r(u) + \mathfrak{R}_r(v)$ for all edges $e = (u, v) \in E$. Therefore, if r is legal, $d_r(u \rightarrow v) \geq 0$ for all edges $e = (u, v) \in E$ since $\mathfrak{R}_r(u)$ and $\mathfrak{R}_r(v)$ are both positive by definition.
2. Any extended retiming can be normalized by subtracting $\min_v t_r(v)$ from all values $t_r(v)$.

We have defined a path above to be a connected sequence of nodes and edges. This definition assumes that a path includes all pieces of its initial and final nodes. On the other hand, we will define a connected sequence of nodes and edges which includes *only some* of the pieces of its initial and final nodes to be a *subpath*. For example, consider the graph of Fig. 7(b). Any path which begins or ends with node A must include all three pieces of node A , while a subpath may begin or end at any of A 's pieces and does not have to contain all of A . Thus a path is a subpath, but a subpath is not necessarily a path.

It should be clear that some pieces of the end-nodes of a path are missing when we discuss a subpath. If a node u is split by $\mathfrak{R}_r(u)$ delays, we can see that we are left with $\mathfrak{R}_r(u) + 1$ pieces which we can denote in order as $u^0, u^1, \dots, u^{\mathfrak{R}_r(u)}$. So if we have a subpath from u^j to v^k , the delay count of the subpath will equal that of the path from u to v , minus the j delays which separate the first $j + 1$ pieces of u from one another, minus the $\mathfrak{R}_r(v) - k$ delays separating $v^k, v^{k+1}, \dots, v^{\mathfrak{R}_r(v)}$. In short,

$$D_r(u^j \Rightarrow v^k) = D(p) + t_r(u) - t_r(v) + \mathfrak{R}_r(u) - \mathfrak{R}_r(v) + k - j,$$

where p is the path from u to v . Note that this is consistent with our previous lemma, since the path from u to v is the same as the subpath from u^0 to $v^{\mathfrak{R}_r(v)}$. Finally, note that, in a graph G with split nodes, $cl(G)$ is the maximum computation time among all zero-delay *subpaths* of G .

4. Unfolding Followed by Extended Retiming

In this section and the next, we will prove one of our major results: that the combination of extended retim-

ing and unfolding yields the same minimal iteration period, no matter which transformation is performed first. We will do this in two steps. First, we wish to show that, for every extended retiming of the unfolded graph which gives us a certain cycle period, we can construct a retiming on the original graph which yields the same cycle period.

4.1. The Main Idea

To illustrate our idea, consider the graph of Fig. 7(a) with iteration bound $\frac{11}{3}$. In order to achieve an optimal iteration period, we must unfold this graph three times, as we will show later. Since we are merely seeking to construct a simple example which demonstrates our method, we will only unfold twice, producing the graph in Fig. 8(a). The minimum cycle period that we can achieve under these circumstances is 8, since 8 is the smallest natural number which, when divided by 2 (the unfolding factor), exceeds the iteration bound.

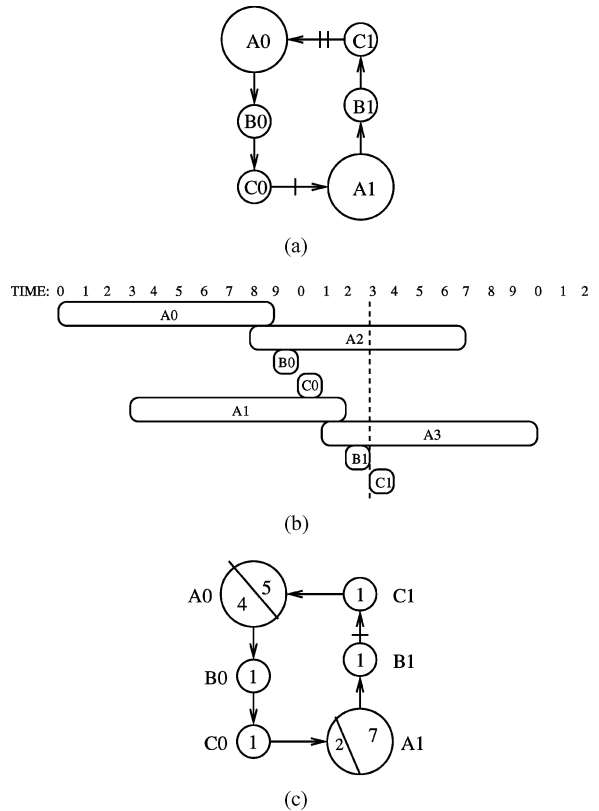


Figure 8. (a) The DFG of Fig. 7(a) unfolded twice; (b) This DFG's schedule with cycle period 8; (c) This DFG retimed.

Thus, we use the method of [1] to schedule this graph within 8 time units, as shown in Fig. 8(b). Note that we've unfolded this graph as much as we want for this exercise and will do no further unfolding. Thus, we can apply the result from [8], cutting the graph immediately before the first occurrence of the last node to enter the schedule (shown here with a dashed line) and reading the extended retiming immediately. This method gives us an extended retiming of

$$\begin{aligned} r(A0) &= 1\frac{5}{9}, & r(A1) &= 1\frac{2}{9}, \\ r(B0) &= r(B1) = r(C0) = 1, & r(C1) &= 0, \end{aligned}$$

whose application to the graph in Fig. 8(a) results in the graph in Fig. 8(c). If G is the graph in Fig. 7(a), then the graph in Fig. 8(c) which has first been unfolded then retimed is denoted as $(G_f)_r$.

We now wish to use this retiming on the unfolded graph to derive a retiming for the original graph. What we will do is to add them using a special addition operator \oplus on the set of extended retimings for a particular node which adds the integer parts of the retimings while concatenating the fractional parts. Formally, for each node u and positive integers i and j ,

$$\begin{aligned} r(u_i) \oplus r(u_j) &= \left(t_r(u_i) + \frac{1}{t(u_i)}(\alpha_1, \dots, \alpha_n) \right) \\ &\oplus \left(t_r(u_j) + \frac{1}{t(u_j)}(\beta_1, \dots, \beta_m) \right) \\ &= (t_r(u_i) + t_r(u_j)) \\ &\quad + \frac{1}{t(u)}(\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_m). \end{aligned} \quad (1)$$

Thus, for our example,

$$\begin{aligned} r(A) &= 1\frac{5}{9} \oplus 1\frac{2}{9} = 2 + \frac{1}{9}(5, 2), & r(B) &= 1 \oplus 1 = 2, \\ r(C) &= 1 \oplus 0 = 1. \end{aligned}$$

Normalizing this and reordering the 2-tuple into ascending order gives us a 2-extended retiming of $r(A) = 1 + \frac{1}{9}(2, 5)$, $r(B) = 1$ and $r(C) = 0$, a retiming similar to the one that we found earlier to achieve the graph in Fig. 9(a). (This graph, the version of G from Fig. 7(a) which is first retimed then unfolded, is denoted as $G_{r,f}$.)

Note that there are some retimings resulting from this method that need special attention. As an example,

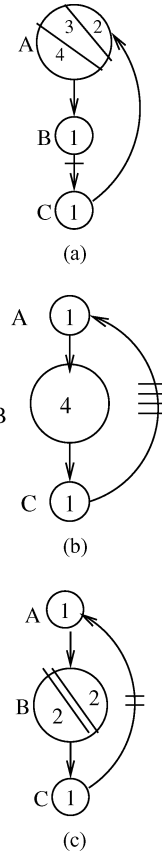


Figure 9. (a) The DFG of Fig. 7(a) retimed to have iteration period $8/2$; (b) A new data-flow graph; (c) The DFG of Fig. 9(b) retimed to have iteration period $3/2$.

we earlier unfolded the graph in Fig. 9(b) by a factor of two and retimed it via the function

$$r(A_i) = 1, \quad r(B_i) = \frac{1}{2}, \quad r(C_i) = 0$$

for $i=0, 1$, to produce an optimized graph. Adding these results in a retiming with $r(A)=2$, $r(B) = \frac{1}{4}(2, 2)$ and $r(C) = 0$. This may look odd, since we have the two delays inside node B next to each other (see Fig. 9(c)). Nonetheless, this is a legal extended retiming which results in an optimal retimed graph.

4.2. The Result

Now that we've established what we want to do, let's formalize our idea. In the next theorem, we demonstrate that, if it is possible for us to first unfold our graph and then retime it to achieve a desired cycle period, then we

may also achieve that cycle period by first retiming the graph and then unfolding. Because retiming takes more time than unfolding, it is desirable to retime the smaller graph first, rather than unfolding and then retiming a much larger graph.

Theorem 4.1. *Let G be a data-flow graph without split nodes, c a cycle period and f an unfolding factor. For every legal extended retiming r_f on the unfolded graph G_f such that $cl((G_f)_{r_f}) \leq c$, there exists a legal extended retiming r on the original graph G such that $cl(G_{r,f}) \leq c$.*

Proof: Using the operator defined in Eq. (1) above, define $r : V \rightarrow \mathbf{Z}$ as

$$r(u) = r_f(u_0) \oplus r_f(u_1) \oplus \dots \oplus r_f(u_{f-1})$$

for each node $u \in V$. Note that this definition and Eq. (1) imply that $t_r(u) = \sum_0^{f-1} t_{r_f}(u_i)$ and $\mathfrak{R}_r(u) = \sum_0^{f-1} \mathfrak{R}_{r_f}(u_i)$. We must show that this is a legal retiming and that $cl(G_{r,f}) \leq c$.

1. Let $e : u \rightarrow v$ be any edge in G ; we want to show that $d_r(e) \geq 0$. Now, by Theorem 2.1(3), there are f copies of this edge in the unfolded graph G_f , the edges $e_i : u_i \rightarrow v_{(i+d(e)) \bmod f}$ for $i = 0, 1, 2, \dots, f - 1$. Furthermore, since the extended retiming r_f is legal, $d_f(e_f) \geq t_{r_f}(v_f) - t_{r_f}(u_f) + \mathfrak{R}_{r_f}(v_f)$ for any edge e_f from this set. Therefore,

$$t_r(v) - t_r(u) + \mathfrak{R}_r(v) = \sum_{i=0}^{f-1} (t_{r_f}(v_{(i+d(e)) \bmod f}) - t_{r_f}(u_i) + \mathfrak{R}_{r_f}(v_i)) \quad (2)$$

and so $t_r(v) - t_r(u) + \mathfrak{R}_r(v) \leq \sum_0^{f-1} d_f(e_i) = d(e)$ by Theorem 2.1(4). Thus

$$\begin{aligned} d_r(e) &= d_r(u \rightarrow v) - \mathfrak{R}_r(u) - \mathfrak{R}_r(v) \\ &= d(e) + t_r(u) - t_r(v) - \mathfrak{R}_r(v) \geq 0 \end{aligned}$$

and r is a legal extended retiming by definition.

2. Let $p_i : u_i \Rightarrow v_j$ be any path in G_f with $T_{r_f}(p_i) > c$ for some integer $i \in [0, f - 1]$; then $D_{f,r_f}(u_i \Rightarrow v_j) \geq 1$ by Theorem 2.2. Since

$$\begin{aligned} D_{f,r_f}(u_i \Rightarrow v_j) &= D_f(p_i) + t_{r_f}(u_i) - t_{r_f}(v_j) \\ &\quad + \mathfrak{R}_{r_f}(u_i) \end{aligned}$$

by Lemma 3.1, we may conclude that

$$t_{r_f}(v_j) - t_{r_f}(u_i) - \mathfrak{R}_{r_f}(u_i) \leq D_f(p_i) - 1$$

for *this* integer i . Now, this path corresponds to a path $p : u \Rightarrow v$ in the original graph G and $j = (i + d(p)) \bmod f$. Thus the work we've done holds for any copy of this path in the unfolded graph. In other words, our inequality is true for *any* integer $i \in [0, f - 1]$. Thus, if we extend Eq. (2) and Theorem 2.1(4) from edges to paths, we see that

$$\begin{aligned} t_r(v) - t_r(u) - \mathfrak{R}_r(u) &\leq \sum_{i=0}^{f-1} (D_f(p_i) - 1) \\ &= D(p) - f, \end{aligned}$$

and so $D_r(u \Rightarrow v) = D(p) + t_r(u) - t_r(v) + \mathfrak{R}_r(u) \geq f$ whenever $T_r(u \Rightarrow v) = T(p) > c$. By Theorem 2.2, $cl(G_{r,f}) \leq c$. □

5. Extended Retiming Followed by Unfolding

The first half of our desired result was fairly simple; this half will be much more complicated. We will start by establishing a couple of facts from [17] which explore the relationship between traditional retiming and unfolding. We will then outline our method for expanding this result to deal with extended retiming, followed by a formal proof.

5.1. Traditional Retiming Followed by Unfolding

The statement and proof of our desired result (Theorem 5.1, below) for traditional retimings appears as Lemma 3.3 in [1], so we will only briefly discuss the main idea of that proof. Given a data-flow graph G , nodes u and v and positive integers i and j smaller than an unfolding factor f , we can show that there is a one-to-one correspondence between the edges $e_f = (u_i, v_j)$ in G_f and $e_{r,f} = (u_{(i-r(u)) \bmod f}, v_{(j-r(v)) \bmod f})$ in $G_{r,f}$. Let r_f be a retiming on G_f and let $d_{f,r_f}(e_f)$ be the delay count of e_f after we apply r_f to G_f . Now, given a retiming r on G , we want to find a function r_f such that $d_{f,r_f}(e_f) = d_{r,f}(e_{r,f})$ for all edges e_f in G_f , where $d_{r,f}(e_{r,f})$ is the delay count of the corresponding $e_{r,f}$ in $G_{r,f}$. Since $d_{f,r_f}(e_f) = d_f(e_f) + r_f(u_i) - r_f(v_j)$ by

the analogue of Lemma 3.1(1) for traditional retimings, our condition becomes

$$r_f(u_i) - r_f(v_j) = d_{r,f}(e_{r,f}) - d_f(e_f) \quad \forall e_f = (u_i, v_j). \quad (3)$$

Since (3) constitutes a consistent linear system with integer solution r_f , our theorem is proved.

5.2. The Main Idea

We now wish to prove the correctness of our assertion using this idea applied to extended retimings. We have seen that an extended retiming on a data-flow graph gives us instructions on how to split a node into smaller pieces. What we will do is to construct a new graph, replacing each split node with the proper smaller pieces. This new graph can then be retimed to have optimal clock period by a traditional retiming which can be derived from the given extended one. Having translated our original graph and extended retiming to a new graph and traditional retiming, we may then apply our above theorem, derive a retiming for the unfolded graph, and then translate back to our original graph.

Consider the sample data-flow graph G in Fig. 10(a) below. Recall that the 2-extended retiming with $r(A) = 1 + \frac{1}{9}(2, 5)$, $r(B) = 1$ and $r(C) = 0$ applied to G achieves an iteration period of 4. We now wish to use this function to construct an extended retiming for G unfolded twice.

We've seen that our extended retiming calls for us to split node A into three pieces with computation times 2, 3 and 4, respectively, while the other nodes remain whole. So, if we split A into three separate nodes with appropriate computation times, as shown in Fig. 10(b), the resulting graph can achieve rate optimality via traditional retiming. We will call such a graph an *extended graph*, and designate the extended graph produced by a DFG G and extended retiming r as $X^{G,r}$.

As we've pointed out, we should be able to retime $X^{G,r}$ to be rate optimal using only a traditional retiming, i.e. a retiming without a fractional component. Recall that the fractional part of $r(A)$ above called for the placement of delays between the first and second pieces and second and third pieces of A , with one delay passing through the node and onto the outgoing edge; thus three delays come into A while only one leaves. In our extended graph, this is equivalent to passing three de-

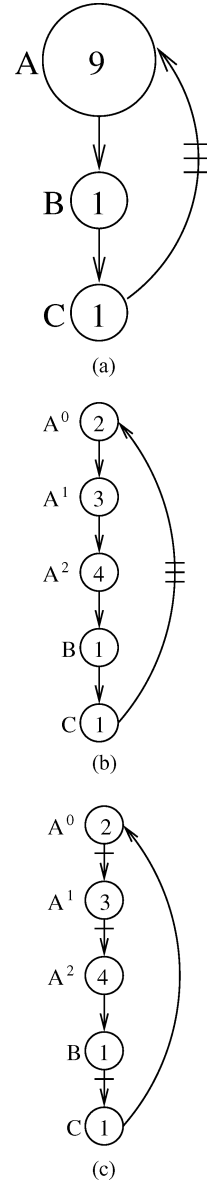


Figure 10. (a) Our sample DFG; (b) The resulting extended graph; (c) This extended graph retimed to have iteration period 4.

lays through node A^0 , leaving one on the edge (A^0, A^1) and passing the others through A^1 , and finally leaving one on (A^1, A^2) while the lone remaining delay passes through A^2 . Thus the traditional retiming ρ on $X^{G,r}$ defined as

$$\rho(A^0) = 3, \quad \rho(A^1) = 2, \quad \rho(A^2) = 1, \quad \rho(B) = 1, \quad \rho(C) = 0$$

is equivalent to the extended retiming r on G and results in a retimed extended graph with iteration period 4, as shown in Fig. 10(c). In keeping with our previous notation, the extended graph $X^{G,r}$ retimed by ρ will be named $X_{\rho}^{G,r}$.

The reader will note that this construction of the extended graph is acceptable in this instance because we have very specific information on how to split our nodes. The more straight-forward approach of splitting all nodes into unit-time pieces, applying traditional retiming, then regrouping is not efficient and may not even run in polynomial time. For example, if the computation time of a node is 1000 clock cycles, splitting this node into 1000 subnodes, retiming, and regrouping is extraordinarily complicated. Our proposed algorithm, which splits a node only when necessary, is clearly preferable. This concept of the extended graph is useful only for deriving properties for extended retiming, as we are doing here.

When we unfold $X_{\rho}^{G,r}$ twice, as shown in Fig. 11(a), the resulting graph has a clock period of 2×4 or 8 and unfolding factor $f = 2$. According to Lemma 3.3 of [1], we must be able to construct a retiming ρ_f for the twice-unfolded extended graph (shown in Fig. 11(b)) which results in a clock period of 8. We do so by matching each edge $e_f = (u_i, v_j)$ of $X_{\rho}^{G,r}$ (Fig. 11(a)) with an edge $e_{\rho,f} = (u_{(i-r(u)) \bmod f}, v_{(j-r(v)) \bmod f})$ from $X_{\rho,f}^{G,r}$ (Fig. 11(a)), then using this matching to construct the linear system of Eqs. (3) for this particular graph. All of this information is given in Table 1. We now solve this system for the values of ρ_f and find a retiming

Table 1. Table of matching edges and linear equations from Figs. 11(a) and (b).

Edge		Corresp.		Equation
e_f	$d_f(e_f)$	$e_{\rho,f}$	$d_{\rho,f}(e_{\rho,f})$	
(A_0^0, A_0^1)	0	(A_0^0, A_0^1)	1	$\rho_f(A_0^0) - \rho_f(A_0^1) = 1$
(A_0^1, A_0^2)	0	(A_0^1, A_0^2)	0	$\rho_f(A_0^1) - \rho_f(A_0^2) = 0$
(A_0^2, B_0)	0	(A_0^2, B_0)	0	$\rho_f(A_0^2) - \rho_f(B_0) = 0$
(B_0, C_0)	0	(B_0, C_0)	1	$\rho_f(B_0) - \rho_f(C_0) = 1$
(C_0, A_1^0)	1	(C_0, A_1^0)	0	$\rho_f(C_0) - \rho_f(A_1^0) = -1$
(A_1^0, A_1^1)	0	(A_0^0, A_1^1)	0	$\rho_f(A_1^0) - \rho_f(A_1^1) = 0$
(A_1^1, A_1^2)	0	(A_1^1, A_0^2)	1	$\rho_f(A_1^1) - \rho_f(A_0^2) = 1$
(A_1^2, B_1)	0	(A_0^2, B_0)	0	$\rho_f(A_1^2) - \rho_f(B_1) = 0$
(B_1, C_1)	0	(B_0, C_0)	0	$\rho_f(B_1) - \rho_f(C_1) = 0$
(C_1, A_0^0)	2	(C_1, A_1^0)	0	$\rho_f(C_1) - \rho_f(A_0^0) = -2$

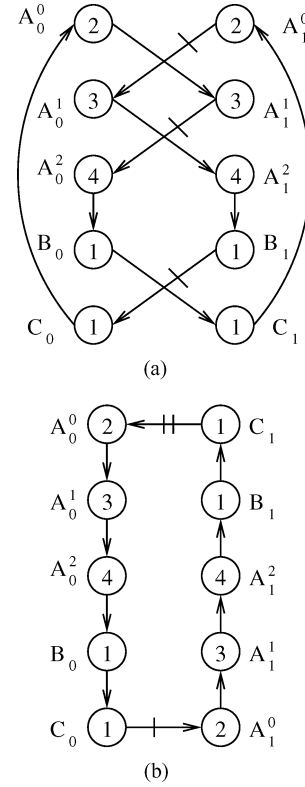


Figure 11. The extended graph unfolded twice: (a) after retiming first; (b) with no retiming.

with

$$\begin{aligned} \rho_f(A_0^0) &= 2, \\ \rho_f(C_0) &= \rho_f(A_1^2) = \rho_f(B_1) = \rho_f(C_1) = 0, \\ \rho_f(A_0^1) &= \rho_f(A_0^2) = \rho_f(B_0) = \rho_f(A_1^0) = \rho_f(A_1^1) \\ &= 1, \end{aligned}$$

which leads to the retimed graph $(X_{\rho_f}^{G,r})_{\rho_f}$ shown in Fig. 12(a).

We now use this retiming to construct a retiming r_f for our original unfolded graph. Note that, in Fig. 12(a), we have placed a delay on the edge (A_0^0, A_0^1) . However, since node A_0^0 was merely our way of representing the first part of node A_0 in our unfolded graph, our extended graph is calling for us to place a delay within A_0 which separates this first piece from the rest of the node. Since this piece takes two time units, we define our extended retiming so that the fractional part of $r_f(A_0)$ is $\frac{2}{9}$. We also have $\rho_f(A_0^2) = 1$, which calls for us to push a delay

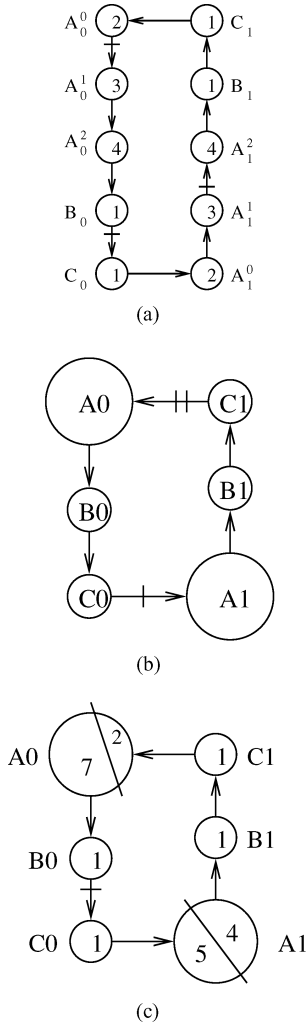


Figure 12. (a) The unfolded extended graph of Fig. 11(b) now retimed by ρ_f ; (b) Our original graph unfolded twice; (c) This graph retimed.

through the last piece of A_0 onto the outgoing edge, and so the integer part of $r_f(A_0)$ must be 1. Similarly, the delay on (A_1^1, A_1^2) separates the last piece of A_1 from the initial part of the node. Since this first part has a total computation time of 5, and since we do not push a delay through A_1^2 , we need to have $r_f(A_1) = \frac{5}{9}$. Since we split no other nodes when we constructed our extended graph, we can let $r_f(u) = \rho_f(u)$ for all remaining nodes u , giving us a final retiming of

$$\begin{aligned} r_f(A_0) &= 1\frac{2}{9}, & r_f(B_0) &= 1, & r_f(C_0) &= 0, \\ r_f(A_1) &= \frac{5}{9}, & r_f(B_1) &= 0, & r_f(C_1) &= 0. \end{aligned}$$

Applying this retiming to our original unfolded graph, shown in Fig. 12(b), results in the graph of Fig. 12(c), which indeed has a clock period of 8.

Fortunately we typically don't have to do this in practice. However, the fact that we can do it if necessary shows us exactly what we were hoping to find: if we can retime then unfold to achieve optimality, we can also unfold then retime to achieve the same optimal result.

5.3. The Result

We now formally prove our desired result, using the ideas we've just run through. Along the way we want to establish that the functions ρ and r_f that we constructed above do, in fact, constitute legal retimings.

Theorem 5.1. *Let $G = \langle V, E, d, t \rangle$ be a data-flow graph without split nodes, c a cycle period and f an unfolding factor. For every legal extended retiming r on G such that $cl(G_{r,f}) \leq c$, there exists a legal extended retiming r_f on the unfolded graph G_f such that $cl((G_f)_{r_f}) \leq c$.*

Proof: Using G and r , construct the extended graph $X^{G,r} = \langle V', E', d', t' \rangle$ as we did in the example, splitting each node v for which $\mathfrak{R}_r(v) > 0$ into pieces of appropriate size, connected by zero-weight edges. Now define the function $\rho : V' \rightarrow \mathbf{Z}^+$ as follows:

1. If $v \in V$ is a vertex with $\mathfrak{R}_r(v) = 0$, then $v \in V'$ as well and we let $\rho(v) = r(v) = t_r(v)$.
2. Otherwise let $\rho(v^j) = t_r(v) + \mathfrak{R}_r(v) - j$ for $j = 0, 1, 2, \dots, \mathfrak{R}_r(v)$.

We wish to show that ρ is a legal traditional retiming of $X^{G,r}$ and $cl(X_{\rho,f}^{G,r}) \leq c$.

1. Let $e = (u, v)$ be any edge of E' , the edge set of $X^{G,r}$. There are two possibilities for e :
 - (a) If e does not correspond to an edge in E , then $u = \alpha^j$ and $v = \alpha^{j+1}$ for some node α of V and some integer j between 0 and $\mathfrak{R}_r(\alpha) - 1$. By definition $d'(e) = 0$ and $\rho(u) - \rho(v) = 1$, so $d'_\rho(e) = d'(e) + \rho(u) - \rho(v) = 1$.
 - (b) Assume that e corresponds to an edge $\varepsilon = (\alpha, \beta)$ in E . Since r is a legal extended retiming, $d_r(\varepsilon) \geq 0$, and we can deduce that $d_r(\varepsilon) = d(\varepsilon) + t_r(\alpha) - t_r(\beta) - \mathfrak{R}_r(\beta) \geq 0$.

We have constructed e such that $d'(e) = d(\varepsilon)$ in this case. Furthermore we note two things:

- i. If α is not a split node in G then $u = \alpha$; otherwise $u = \alpha^{\mathfrak{R}_r(\alpha)}$, the last piece of α . In either case $\rho(u) = \iota_r(\alpha)$ by definition.
- ii. Similarly if β is not split then $v = \beta$ and $\rho(v) = \iota_r(\beta)$; otherwise $v = \beta^0$, the first piece of β , and $\rho(v) = \iota_r(\beta) + \mathfrak{R}_r(\beta)$.

Thus we have two subcases:

- i. If β is split, then $d'_\rho(e) = d(\varepsilon) + \iota_r(\alpha) - \iota_r(\beta) - \mathfrak{R}_r(\beta) = d_r(\varepsilon) \geq 0$.
- ii. Otherwise β is not split, $\mathfrak{R}_r(\beta) = 0$ and $d'_\rho(e) = d(\varepsilon) + \iota_r(\alpha) - \iota_r(\beta) = d_r(\varepsilon) \geq 0$.

In any case $d'_\rho(e) \geq 0$ for any edge e in $X^{G,r}$ and so ρ is a legal traditional retiming by definition.

2. If $cl(X_{\rho,f}^{G,r}) > c$, then there is a zero-delay path $p : u \Rightarrow v$ in $X_{\rho,f}^{G,r}$ with $T'(p) > c$. This p corresponds to some zero-delay subpath π in $G_{r,f}$ with $T(\pi) > c$, and hence $cl(G_{r,f}) > c$. Thus, by contapositive argument, $cl(G_{r,f}) \leq c$ implies that $cl(X_{\rho,f}^{G,r}) \leq c$.

Since the retimed and unfolded graph $X_{\rho,f}^{G,r}$ has a clock period bounded above by c , by Lemma 3.3 of [1], we have a legal traditional retiming ρ_f such that $cl((X_f^{G,r})_{\rho_f}) \leq c$.

We now use the extended graph $X^{G,r}$ and the legal traditional retiming ρ_f and construct the extended retiming r_f of G_f via the algorithm we described earlier. We must show that r_f is legal and that $cl((G_f)_{r_f}) \leq c$.

1. Let $e : u_i \rightarrow v_i$ be an edge in $(G_f)_{r_f}$. Without loss of generality assume that both u_i and v_i are split nodes. Then e corresponds to the edge $\varepsilon : u_i^0 \rightarrow v_i^0$ in $(X_f^{G,r})_{\rho_f}$ and so $d_{r_f}(e) = d_{\rho_f}(\varepsilon) \geq 0$. By definition r_f is a legal extended retiming.
2. Similarly, any zero-delay subpath p in $(G_f)_{r_f}$ corresponds to a zero-delay path π in $(X_f^{G,r})_{\rho_f}$. By choice of ρ_f we must have $T(p) = T(\pi) \leq c$, and so $cl((G_f)_{r_f}) \leq c$. □

6. Finding an Extended Retiming from a Static Schedule

As we've said throughout this paper, we currently have one method for finding an extended retiming which can be combined with a non-trivial unfolding factor

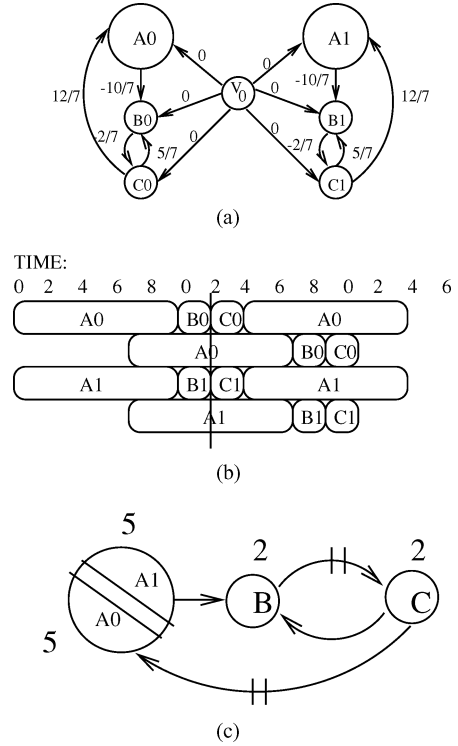


Figure 13. (a) Scheduling graph for Fig. 2(a); (b) Cut schedule for this DFG; (c) The retimed DFG.

to achieve optimality. In this section we will develop another, more efficient algorithm. We demonstrate our methods using the graph in Fig. 1(a) with a clock period of 7 and unfolding factor 2.

Our current procedure calls for us to unfold the graph twice (as in Fig. 2(a)) and then schedule it with a clock period of 7. We use DFG scheduling as defined in [18], starting with the construction of the scheduling graph in Fig. 13(a). We note from this graph that $sh(A0) = sh(A1) = 0$, $sh(B0) = sh(B1) = -\frac{10}{7}$ and $sh(C0) = sh(C1) = -\frac{12}{7}$. We next construct the schedule of Fig. 13(b) according to the formula $S_7(v, i) = 7(i - sh(v))$. Since this is the schedule for our unfolded graph and we will do no further unfolding, we can apply the result from [8], cutting the graph immediately before the last nodes to enter the schedule (i.e. the two copies of C) and instantly reading a legal retiming with $r(A0) = r(A1) = 1\frac{5}{10}$, $r(B0) = r(B1) = 1$ and $r(C0) = r(C1) = 0$. It is this function which yields the graph in Fig. 2(b) when applied to the graph in Fig. 2(a).

This function is now used to construct a legal retiming on the original graph. We add the retimings for all

copies of a particular node together using our special \oplus operator. Thus, for our example, $r(A) = 1 \frac{5}{10} \oplus 1 \frac{5}{10} = 2 + \frac{1}{10}(5, 5)$, $r(B) = 1 \oplus 1 = 2$ and $r(C) = 0 \oplus 0 = 0$. Applying this to our original graph in Fig. 1(a) results in the graph in Fig. 13(c), with two delays inside of node A next to each other. The result is an optimized graph, but the process requires a great deal of time and space because we are working with the much larger unfolded graph.

In Theorems 4.1 and 5.1, we demonstrated that the order of application didn't matter; we could derive an optimal result either by unfolding then retiming or by applying retiming first. Therefore, it makes sense that we should be able to construct a method similar to the above one, but which is applied to the original graph. Let us attempt to do what we did above without the unfolding. In other words, we propose to construct our static schedule as before, based on the original graph this time. We will cut this resulting schedule and read our retiming as before.

We begin by applying this proposed algorithm to the graph in Fig. 1(a). The scheduling graph with clock period 7 and unfolding factor 2 is displayed as Fig. 14(a);

note that $sh(A) = 0$, $sh(B) = -\frac{20}{7}$ and $sh(C) = -\frac{24}{7}$ in this case. This graph is now scheduled according to the formula $S_{7/2}(v, i) = \lceil \frac{7}{2}(i - sh(v)) \rceil$ and cut before C's initial entrance, as in Fig. 14(b). The function that we now read has $r(A) = 1 + \frac{1}{10}(1, 5, 8)$, $r(B) = 1$ and $r(C) = 0$. When applied to the original graph (as in Fig. 14(c)), this function appears to be a legal retiming which does optimize the graph.

Having established what we want to do, we must formalize this method and prove its result is a legal retiming which optimizes a DFG. Let $S(v, i) = \lceil \frac{c}{f}(i - sh(v)) \rceil$ be the integral schedule with clock period c and unfolding factor f . $S(v, i)$ gives the starting time of node v in the i th iteration. Thus the time at which the prologue ends, which we will denote as M and is where we want to make our cut, equals the starting time of the last node to enter the static schedule. In other words, $M = \max_v S(v, 0)$. (See that $M = S(C, 0) = 12$ above.) We now wish to count the number of either whole or partial occurrences of each node to the left of this cut. The i th copy of node v begins to the left of the cut if $S(v, i) < M$. A copy of a node is complete if $M - S(v, i) \geq t(v)$; otherwise it is partial. Clearly each complete copy of a node adds 1 to the eventual retiming function. On the other hand, if a copy is cut, we only want to add the fraction of the node to the left of the cut, which is found by dividing the piece's computation time by the computation time of the whole node. At the end, we combine the contributions from a node's copies via our \oplus operator, finally arriving at the retiming formula

$$r(v) = \bigoplus_{i: S(v,i) < M} \min \left\{ 1, \frac{M - S(v, i)}{t(v)} \right\}. \quad (4)$$

Since the computation of this formula uses our shortest path algorithm, it makes sense that it has the same time complexity as that algorithm, namely $O(|V||E|)$. Let us consider this formula when applied to node A of Fig. 1(a). As we can see from our schedule in Fig. 14(b), the first four iterations of A are to be considered when constructing the node's retiming:

1. $S(A, 0) = 0$ and $\min\{1, \frac{12}{10}\} = 1$.
2. $S(A, 1) = \lceil \frac{7}{2} \cdot 1 \rceil = 4$ and $\min\{1, \frac{12-4}{10}\} = \frac{8}{10}$.
3. $S(A, 2) = \lceil \frac{7}{2} \cdot 2 \rceil = 7$ and $\min\{1, \frac{12-7}{10}\} = \frac{5}{10}$.
4. $S(A, 3) = \lceil \frac{7}{2} \cdot 3 \rceil = 11$ and $\min\{1, \frac{12-11}{10}\} = \frac{1}{10}$.

Combining these figures gives us $r(A) = 1 \oplus \frac{8}{10} \oplus \frac{5}{10} \oplus \frac{1}{10} = 1 + \frac{1}{10}(1, 5, 8)$, exactly the same answer we found

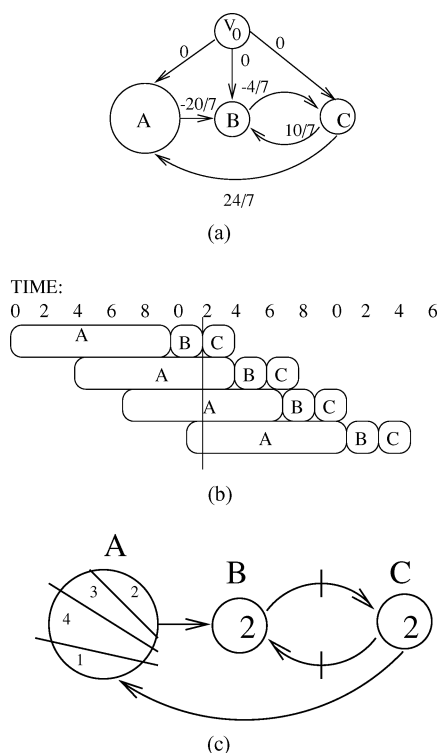


Figure 14. (a) Scheduling graph for Fig. 1(a); (b) Cut schedule for this DFG; (c) The retimed DFG.

by simply examining the schedule table. Our formula appears to accurately describe this situation.

To further our confidence in this proposition, we can also check that (4) matches our previous formula from [7, 8], a legal retiming in the case where we have no unfolding. In this case, a simple closed form can be derived.

Lemma 6.1. *Let G be a DFG and c a positive integer with $t(v) \leq c$ for all nodes v . In the case when $f = 1$, Eq. (4) is equivalent to the formula $r'(v) = \lfloor sh(v) - X \rfloor + \min\{1, \frac{c}{t(v)}((sh(v) - X) - \lfloor sh(v) - X \rfloor)\}$ where $X = \min_v sh(v)$.*

Proof: First note that $M = -c \cdot X$ and $S(v, i) = c(i - sh(v))$ by definition when $f = 1$. Now, suppose that node v is split in both iterations i and $i + k$ for some integer $k > 0$. Since the i th iteration is split, $0 < M - S(v, i) < t(v)$, implying that $0 < sh(v) - i - X < \frac{t(v)}{c} \leq 1 \leq k$. Thus $-k < sh(v) - (i + k) - X < 0$, and so $M - S(v, i + k) < 0$. This places the $(i + k)$ th copy of v as starting after time M , contradicting our assertion that it was also to be split in our schedule, and we conclude that any node is split in at most one iteration in the case where $f = 1$ and $t(v) \leq c$ for all nodes v .

Given a node v , the only way a copy of v does not appear in the prologue is if $M = S(v, 0)$ and $r(v) = 0$. In this case $sh(v) = X$ and $r'(v) = 0$ as well. This leaves us the case where a copy of v does appear in the prologue, which splits into two subcases:

1. If no copy of v is split, then there exists an integer $i > 0$ such that $M - S(v, i - 1) \geq t(v)$ while $S(v, i) - M \geq t(v)$. In this case $r(v) = i$. See that $M - S(v, k) = c(sh(v) - k - X)$ for any k , and so $M - S(v, i - 1) \geq t(v)$ implies that $sh(v) - X \geq (i - 1) + \frac{t(v)}{c}$. Similarly $M - S(v, i) \leq -t(v)$ implies that $sh(v) - X \leq i - \frac{t(v)}{c}$. Since $\frac{t(v)}{c} \in (0, 1]$, we thus conclude that $\lfloor sh(v) - X \rfloor = i - 1$, and so $sh(v) - X - \lfloor sh(v) - X \rfloor = sh(v) - X - (i - 1) \geq \frac{t(v)}{c}$, or $\frac{c}{t(v)}(sh(v) - X - \lfloor sh(v) - X \rfloor) \geq 1$. Therefore $r'(v) = (i - 1) + 1 = i = r(v)$.
2. On the other hand assume that the i th copy of v is split. Therefore $M - S(v, i - 1) \geq t(v)$ while $0 < M - S(v, i) < t(v)$. See that $r(v) = i + \frac{M - S(v, i)}{t(v)}$ in this case. Now, $M - S(v, i)$ lying in the interval $(0, t(v))$ implies that $i < sh(v) - X < i + \frac{t(v)}{c} \leq i + 1$, so $\lfloor sh(v) - X \rfloor = i$. Furthermore, $M - S(v, i) = c(sh(v) - i - X) = c((sh(v) - X) - \lfloor sh(v) - X \rfloor)$,

$$\text{and so } r(v) = \lfloor sh(v) - X \rfloor + \frac{c}{t(v)}((sh(v) - X) - \lfloor sh(v) - X \rfloor) = r'(v).$$

Thus, for all nodes v , $r(v) = r'(v)$ when $f = 1$ and $t(v) \leq c$ for all v . \square

We now are very confident of our assertion, but must still show that (4) is, in fact, a legal extended retiming which minimizes the iteration period of a data-flow graph. Recall that these definitions are based on the t_r and \mathfrak{R}_r functions for the retiming r in question. Therefore, before proceeding to our primary result, we must find the closed forms of these functions for our proposed formula.

Lemma 6.2. *Let r be the extended retiming given by Eq. (4) above. Then $t_r(v) = \lfloor \frac{f}{c}(M - t(v)) + sh(v) \rfloor + 1$ and $\mathfrak{R}_r(v) = \lfloor \frac{f}{c}(M - 1) + sh(v) \rfloor + 1 - t_r(v)$.*

Proof:

1. To find the number of complete copies of v in the prologue, we want to find the largest i such that $M - S(v, i) \geq t(v)$. This will give us the actual iteration number of the last copy of v in the prologue; for the count we must then add 1. Now, since $S(v, i) = \lceil \frac{c}{f}(i - sh(v)) \rceil$ by definition, we see that this quantity under the ceiling function is bounded above by $M - t(v)$, or $i \leq \frac{f}{c}(M - t(v)) + sh(v)$. Since i must be an integer, we now take the floor function of the right-hand side of this inequality to find the largest possible value for i .
2. Similarly, we must know the iteration number of the first copy of v which does not start in the prologue, i.e. the smallest i such that $S(v, i) \geq M$. To find the number of copies of v that are split in our schedule, we then need only subtract $t_r(v)$, the number of complete copies of v in our prologue, from i . Now, again by definition of $S(v, i)$, $S(v, i) \geq M$ implies that $\frac{c}{f}(i - sh(v)) > M - 1$ or $i > \frac{f}{c}(M - 1) + sh(v)$. If the quantity on the right-hand side of this inequality is not an integer, then we want i equal to the ceiling of this quantity, which is the same as the floor of this quantity plus 1. On the other hand, if this quantity is an integer, we must have i equal to 1 plus the right-hand side, which is the same as 1 plus the floor of the right-hand side. In any case $i = \lfloor \frac{f}{c}(M - 1) + sh(v) \rfloor + 1$ and our result is shown. \square

With this result we can now show:

Theorem 6.3. *Let $G = \langle V, E, d, t \rangle$ be a DFG with iteration period $\frac{c}{f} \geq 1$, i.e. with clock period c and unfolding factor f . Then the retiming r described by Eq. (4) is a legal extended retiming on G such that $cl(G_{r,f}) \leq c$ if and only if the scheduling graph G^s contains no negative-weight cycle.*

Proof:

- Assume that $cl(G_{r,f}) \leq c$. By the known properties of the iteration bound from [11], we have

$$B(G) = \frac{B(G_{r,f})}{f} \leq \frac{cl(G_{r,f})}{f} \leq \frac{c}{f},$$

and so $\frac{T(\ell)}{D(\ell)} \leq \frac{c}{f}$ for all cycles ℓ in G . Thus $W(\ell) = D(\ell) - \frac{f}{c} \cdot T(\ell) \geq 0$ for all cycles ℓ in G^s , and so the scheduling graph contains no negative-weight cycle.

- On the other hand assume that G^s contains no negative-weight cycle.

1. To prove the legality of r , we must show that $d_r(e) = d_r(u \rightarrow v) - \mathfrak{R}_r(u) - \mathfrak{R}_r(v) = d(e) + \iota_r(u) - \iota_r(v) - \mathfrak{R}_r(v) \geq 0$ for any edge $e = (u, v)$ of G . By the definition of the scheduling graph, $sh(v) \leq sh(u) + d(e) - \frac{f}{c} \cdot t(u)$, and so $d(e) + sh(u) - sh(v) \geq \frac{f}{c} \cdot t(u)$. Thus, by Lemma 6.2 above,

$$\begin{aligned} d_r(e) &= d(e) + \iota_r(u) - \iota_r(v) \\ &\quad - \left\lfloor \frac{f}{c}(M-1) + sh(v) \right\rfloor - 1 + \iota_r(v) \\ &= d(e) + \left\lfloor \frac{f}{c}(M-t(u)) + sh(u) \right\rfloor \\ &\quad + 1 - \left\lfloor \frac{f}{c}(M-1) + sh(v) \right\rfloor - 1 \\ &\geq \left\lfloor \frac{f}{c}(1-t(u)) + (d(e) + sh(u) - sh(v)) \right\rfloor \\ &\geq \left\lfloor \frac{f}{c} \right\rfloor \geq 0. \end{aligned}$$

2. Let $p : u \Rightarrow v$ be a path in G with $T(p) > c$. We wish to show that $D_r(p) = D(p) + \iota_r(u) - \iota_r(v) + \mathfrak{R}_r(u) \geq 1$. Again, due to the construction of the scheduling graph, $sh(v) \leq sh(u) + D(p) - \frac{f}{c}(T(p) - t(v))$, and so $D(p) + sh(u) - sh(v) \geq \frac{f}{c}(T(p) - t(v))$. Therefore, by Lemma 6.2 again,

$$\begin{aligned} D_r(p) &= D(p) + \iota_r(u) - \iota_r(v) \\ &\quad + \left\lfloor \frac{f}{c}(M-1) + sh(u) \right\rfloor + 1 - \iota_r(u) \\ &= D(p) + \left\lfloor \frac{f}{c}(M-1) + sh(u) \right\rfloor + 1 \\ &\quad - \left\lfloor \frac{f}{c}(M-t(v)) + sh(v) \right\rfloor - 1 \\ &\geq \left\lfloor \frac{f}{c}(t(v)-1) + (D(p) + sh(u) - sh(v)) \right\rfloor \\ &\geq \left\lfloor \frac{f}{c}(t(v)-1) + \frac{f}{c}(T(p) - t(v)) \right\rfloor \\ &= \left\lfloor \frac{f}{c}(T(p)-1) \right\rfloor \\ &> \left\lfloor \frac{f}{c}(c-1) \right\rfloor = \left\lfloor f - \frac{f}{c} \right\rfloor = f - 1 \end{aligned}$$

since $f \leq c$ by assumption. Thus $D_r(p) \geq f$ whenever $T(p) > c$, and by Theorem 2.2, $cl(G_{r,f}) \leq c$. \square

Let's now summarize what we've proven so far:

Theorem 6.4. *Let G be a data-flow graph with $B(G) \geq 1$ which has no split nodes. Let f and c be positive integers. The following statements are equivalent:*

1. The iteration bound $B(G) \leq \frac{c}{f}$.
2. The scheduling graph G^s contains no cycle having negative delay count.
3. There exists a legal extended retiming r on G such that $cy(G_{r,f}) \leq c$.
4. There exists a legal extended retiming r_f on the unfolded graph G_f such that $cy((G_f)_{r_f}) \leq c$.
5. There exists a legal, integral, repeating, static schedule for G with unfolding factor f and cycle period c .

Proof: The equivalence of (1) and (2) is given by Lemma 2.3 (Lemma 3.1 of [18]). The equivalence of (2) and (3) is Theorem 6.3 above. The combination of Theorems 4.2 and 5.1 yield the equivalence of (3) and (4). Finally, the equivalence of (1) and (5) is demonstrated by Theorem 3.5 of [18]. \square

As a final aside, we above raised the question of exactly how many delays may be placed inside a node. We can now derive our answer in the general case:

Lemma 6.5. *Let G be a DFG, with $B(G) = \frac{c}{f}$ the iteration bound in lowest terms. Let k be the smallest positive integer such that $t(v) \leq kc$ for all nodes v of G . Then at most kf delays may be placed inside of any node as a result of extended retiming.*

Proof: Assume by way of contradiction that $kf + 1$ or more delays are to be placed inside of node v . This implies that this many copies of v are cut in the schedule table. Let i be the smallest iteration number of one of these copies, so that $0 < M - S(v, i) < t(v)$. By our assumption, the $(i + kf)$ th copy of v is also split, implying that $0 < M - S(v, i + kf) < t(v)$. However, $S(v, i + kf) = \lceil \frac{c}{f}(i - sh(v) + kf) \rceil = \lceil \frac{c}{f}(i - sh(v)) + kc \rceil = S(v, i) + kc$ by definition, and so $t(v) \leq kc < M - S(v, i)$, contradicting our choice of i . Thus we may place at most kf delays within any node v . \square

Returning to our example in Fig. 1(a), the iteration bound of the graph is $\frac{7}{2}$ but the size of node A dictates that $k = 2$. By this lemma, we know that at most 4 delays are placed inside any of the three nodes, and that any node is divided into at most 5 pieces by extended retiming.

7. Minimum Rate-Optimal Unfolding Factors

As we've said, an *iteration* of a data-flow graph is simply an execution of all nodes once. The average computation time of an iteration is called the *iteration period* of the DFG. If the DFG G contains a cycle, the average computation time of the cycle is the total computation time of the nodes divided by the number of delays in the cycle. This ratio must be smaller than the iteration period of the whole graph since the cycle constitutes a subgraph of G . If we compute the maximum time-to-delay ratio over all cycles of G we derive a lower bound on the iteration period of G . This maximum time-to-delay ratio is called the *iteration bound* [16] of G and is denoted $B(G)$.

If the iteration period of a graph's schedule equals the graph's iteration bound, the schedule is said to be *rate-optimal*. As we've said throughout this paper, our goal is to achieve rate-optimality via retiming and unfolding. If a data-flow graph can be unfolded f times and achieve rate-optimality (i.e. a clock period equal to $f \cdot B(G)$), we say that f is the *rate-optimal unfolding factor* for G . Obviously we wish to achieve rate-optimality while unfolding as little as possible. To this end we need to compute the minimum rate-optimal

unfolding factor for any graph. We begin by showing this link between a graph's clock period and iteration bound:

Lemma 7.1. *Let G be a data-flow graph without split nodes, c a cycle period and f an unfolding factor. Then there exists a legal extended retiming r on G such that $cl(G_{r,f}) \leq c$ if and only if $B(G) \leq \frac{c}{f}$.*

Proof: By Theorem 5.1, the existence of r implies the existence of an extended retiming r_f such that $cl((G_f)_{r_f}) \leq c$. Note that G_f is our unfolded graph; alone it has an unfolding factor of 1. Therefore, by Theorem 2.2 of [7], $B(G_f) \leq c$. However, by Property 6.2 of [11], $B(G_f) = f \cdot B(G)$ and so $B(G) \leq \frac{c}{f}$. The opposite implication is proved similarly. \square

With this in hand we can show:

Theorem 7.2. *Let G be a data-flow graph without split nodes. Let ℓ be a critical cycle of G , i.e. $B(G) = \frac{T(\ell)}{D(\ell)}$. Let g be the greatest common divisor of $T(\ell)$ and $D(\ell)$. Then $\frac{D(\ell)}{g}$ is the minimum rate-optimal unfolding factor for G .*

Proof: Let $\sigma = \frac{T(\ell)}{g}$ and $\rho = \frac{D(\ell)}{g}$. We need to show that ρ is a rate-optimal unfolding factor and that it is minimal.

1. By definition $B(G_{r,\rho}) \leq cl(G_{r,\rho})$ for any retiming r . On the other hand, since $B(G_r) = B(G)$ for any retiming r of G ,

$$\begin{aligned} B(G_{r,\rho}) &= \rho \cdot B(G_r) = \rho \cdot B(G) \\ &= \frac{D(\ell)}{g} \cdot \frac{T(\ell)}{D(\ell)} = \frac{T(\ell)}{g}, \end{aligned}$$

which is integral and is thus a legitimate choice of clock period for G . Since $\frac{B(G_{r,\rho})}{\rho} = B(G)$ for any retiming r of G , by Lemma 7.1 there exists a legal extended retiming r_0 such that $cl(G_{r_0,\rho}) \leq B(G_{r_0,\rho})$. Thus $cl(G_{r_0,\rho}) = B(G_{r_0,\rho})$ and ρ is a rate-optimal unfolding factor by definition.

2. Assume f is any other rate-optimal unfolding factor. Thus there exists an integer c such that $B(G) = \frac{c}{f}$. Therefore

$$\frac{c}{f} = \frac{\sigma}{\rho} = \frac{T(\ell)}{D(\ell)},$$

and so both fractions are reduced forms for $B(G)$. However, by definition of g , $\frac{\sigma}{\rho}$ must be the *most*

Table 2. Simulation results for common circuits.

Benchmark	Computation time			Iter. Bound	Min. optimal unf old factor		Iter. Pd. w/ Bold unf old factor	
	Add	Mult	Slow-down		Ext.	Trad.	Ext.	Trad.
Second order IIR filter	1	4	2	3	1	2	3	4
Second order IIR filter	1	10	6	2	1	6	2	10
2-Cascaded biquad filter	4	25	6	$\frac{11}{3}$	2	6	5.5	12.5
All-pole lattice filter	2	5	12	$\frac{3}{2}$	2	6	1.5	2.5
All-pole lattice filter	1	12	7	4	1	7	4	12
Fifth order elliptic filter	2	12	16	$\frac{7}{2}$	2	8	3.5	6
Fifth order elliptic filter	2	30	20	$\frac{11}{2}$	2	20	5.5	15

reduced form, and so $\rho \leq f$ for any other rate-optimal unfolding factor f . □

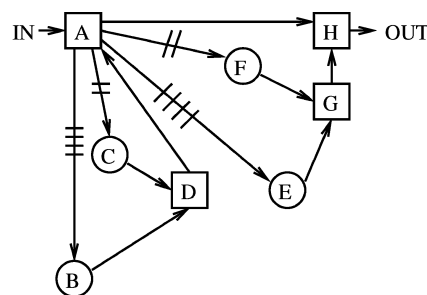
In short, to find the rate-optimal unfolding factor of a data-flow graph G , we compute $B(G)$ (a polynomial-time operation [19]) and reduce the resulting fraction to lowest terms. The denominator of this fraction is our desired unfolding factor.

8. Examples

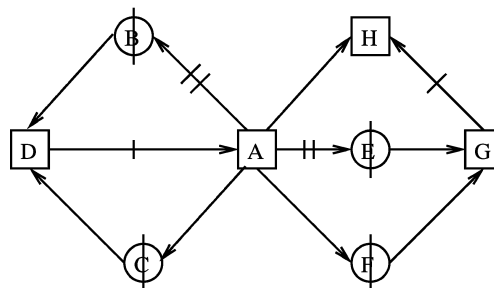
Let's consider the data-flow graph representation of a IIR filter. Assume that a multiplier (shown below as a circle) require four units of computation time, as opposed to one for an adder (shown as a square). Furthermore, to complicate our example, multiply the register count of each edge by 2, referred to in [3] as applying a *slowdown* of 2 to our original circuit. The result is pictured in Fig. 15(a).

The resulting circuit has an iteration bound of 3, and can be retimed via extended retiming to achieve this clock period as in Fig. 15(b) without unfolding. However, if we restrict ourselves to traditional retiming, the best clock period we can get is 4. The only way to obtain an optimal result is to unfold the graph by a factor of 2 and retime for a clock period of 6, as shown in Fig. 15(c).

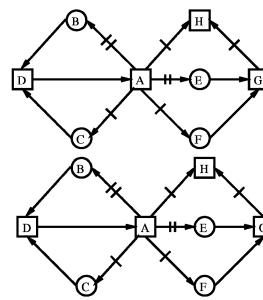
Repeating this exercise with other common filters yields Table 2. In all cases, we achieve better results by using extended retiming, getting an optimal clock period while requiring less unfolding. This improvement is illustrated by the last four columns of our table. Limiting ourselves to traditional retiming forces us to decide between two poor options:



(a)



(b)



(c)

Figure 15. (a) A 2-slow second order IIR filter; (b) This graph optimized by extended retiming; (c) This graph optimized by traditional retiming.

1. If we want an optimal clock period we must unfold by a larger factor, which is listed for each example in the second-to-last column of Table 2. This dramatically increases the size of our circuit, and thus the number of functional units we require and the production costs.
2. On the other hand, if we want to unfold by our extended unfolding factor (shown in boldface in the table), we will be forced to accept a larger iteration period (listed in the last column of the same table). The result is a smaller circuit running at less than optimal speed.

9. Conclusion

In this paper, we have improved our previous result by combining extended retiming with unfolding. We have shown that the order in which we retime and unfold is immaterial; this is demonstrated by the combination of Theorems 4.1 and 5.1. This result indicates that we should be able to find a retiming immediately without unfolding first, and we have constructed an $O(|V||E|)$ method to do this, based on our earlier simplified algorithm from [8]. Indeed, we have demonstrated that our work here is a generalization of our earlier work [7–10]. We have also proven an upper bound on the number of delays which may be embedded within a node as a result of extended retiming. Finally, we have developed a method for calculating the optimal unfolding factor for extended retiming: compute the iteration bound, reduce it to lowest terms, then use the denominator of the resulting fraction.

This work represents a theoretical ideal, yielding the best possible results while assuming a perfect world. Moving from this ideal to the real world of hardware implementations and resource constraints remains a significant, crucial and complicated next step.

In particular, we have developed these results while assuming the use of integral schedules. We also have the possibilities of fractional schedules, where operations may be scheduled at any time (not necessarily at integral points) [18]. Additionally, we have assumed the use of the most basic data-flow graph possible, although important variations have been described [20, 21]. We thus have additional models to explore as we discuss extended retiming.

Acknowledgments

This work was partially supported by NSF grants MIP-9501006 and MIP-9704276; and by the A.J. Schmitt Foundation while the authors were with the University of Notre Dame. It was also supported by the University of Akron, NSF grants ETA-0103709 and CCR-0309461, Texas ARP grant 009741-0028-2001 and the TI University program.

References

1. L.-F. Chao and E.H.-M. Sha, "Scheduling Data-Flow Graphs via Retiming and Unfolding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, 1997, pp. 1259–1267.
2. A. Zaky and P. Sadayappan, "Optimal Static Scheduling of Sequential Loops on Multiprocessors," in *Proceedings of the International Conference on Parallel Processing*, 1992, pp. III 130–137.
3. C.E. Leiserson and J.B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, vol. 6, 1991, pp. 5–35.
4. S.Y. Kung, J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing*, Prentice Hall, 1985.
5. L.-F. Chao and E.H.-M. Sha, "Retiming and Unfolding Data-Flow Graphs," in *Proceedings of the International Conference on Parallel Processing*, 1992, pp. II 33–40.
6. M. Lam, "Software Pipelining: An Effective Scheduling Technique for VLIW Machines," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1988, pp. 318–328.
7. T.W. O'Neil, S. Tongsima, and E.H.-M. Sha, "Extended Retiming: Optimal Retiming via a Graph-Theoretical Approach," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 4, 1999, pp. 2001–2004.
8. T.W. O'Neil, S. Tongsima, and E.H.-M. Sha, "Optimal Scheduling of Data-Flow Graphs Using Extended Retiming," in *Proceedings of the ISCA 12th International Conference on Parallel and Distributed Computing Systems*, 1999, pp. 292–297.
9. T.W. O'Neil and E.H.-M. Sha, "Rate-Optimal Graph Transformation via Extended Retiming and Unfolding," in *Proceedings of the IASTED 11th International Conference on Parallel and Distributed Computing and Systems*, vol. 10, 1999, pp. 764–769.
10. T.W. O'Neil and E.H.-M. Sha, "Optimal Graph Transformation using Extended Retiming with Minimal Unfolding," in *Proceedings of the IASTED 12th International Conference on Parallel and Distributed Computing and Systems*, 2000, pp. 128–133.
11. K.K. Parhi and D.G. Messerschmitt, "Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimum Unfolding," *IEEE Transactions on Computers*, vol. 40, 1991, pp. 178–195.
12. P.-Y. Calland, A. Darte, and Y. Robert, "Circuit Retiming Applied to Decomposed Software Pipelining," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, 1998, pp. 24–35.
13. F. Sanchez and J. Cortadella, "Reducing Register Pressure in Software Pipelining," *Journal of Information Science and Engineering*, vol. 14, 1998, pp. 265–279.

14. K.S. Chatha and R. Vemuri, "RECOD: A Retiming Heuristic to Optimize Resource and Memory Utilization in HW/SW Codesigns," in *Proceedings of the IEEE International Workshop on Hardware/Software Codesign*, 1998, pp. 139–143.
15. M. Sheliga, N.L. Passos, and E.H.-M. Sha, "Fully Parallel Hardware/Software Codesign for Multi-Dimensional DSP Applications," in *Proceedings of the IEEE International Workshop on Hardware/Software Codesign*, 1996, pp. 18–25.
16. M. Renfors and Y. Neuvo, "The Maximum Sampling Rate of Digital Filters Under Hardware Speed," *Transactions on Circuits and Sampling*, vol. CAS-28, 1981, pp. 196–202.
17. L.-F. Chao, "Scheduling and Behavioral Transformations for Parallel Systems," PhD thesis, Dept. of Computer Science, Princeton University, 1993.
18. L.-F. Chao and E. H.-M. Sha, "Static Scheduling for Synthesis of DSP Algorithms on Various Models," *Journal of VLSI Signal Processing*, vol. 10, 1995, pp. 207–223.
19. A. Dasdan and R.K. Gupta, "Faster Maximum and Minimum Mean Cycle Algorithms for System-Performance Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, 1998, pp. 889–899.
20. E.A. Lee and D.G. Messerschmitt, "Static Scheduling of Synchronous Data-Flow Programs for Digital Signal Processing," *IEEE Transactions on Computers*, vol. 36, 1987, pp. 24–35.
21. G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-Static Dataflow," *IEEE Transactions on Signal Processing*, vol. 44, 1996, pp. 397–408.



Timothy O'Neil received his Ph.D. in Computer Science and Engineering from the University of Notre Dame in 2002, where he was

awarded the Arthur J. Schmitt Fellowship. He also received master's degrees in mathematics (1991) and computer and information sciences (1993) from The Ohio State University in Columbus, Ohio. He is presently an Assistant Professor in the Computer Science Department at the University of Akron. His current research interests include loop transformations and data scheduling.



Edwin Hsing-Mean Sha received the B.S.E. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1986; he received the M.A. and Ph.D. degree from the Department of Computer Science, Princeton University, Princeton, NJ, in 1991 and 1992, respectively. From August 1992 to August 2000, he was with the Department of Computer Science and Engineering at University of Notre Dame, Notre Dame, IN. He served as Associate Chairman for Graduate Studies from 1995 to 2000. Since 2000, he has been a tenured full professor in the Department of Computer Science at the University of Texas at Dallas.

He has published more than 170 research papers in referred conferences and journals. He has been serving as an editor for several journals such as *IEEE Transactions on Signal Processing* and *Journal of VLSI Signal Processing*. He also served as program committee members in numerous conferences. He received Oak Ridge Association Junior Faculty Enhancement Award in 1994, and NSF CAREER Award. He was a guest editor for the special issue on Low Power Design of *IEEE Transactions on VLSI Systems* in 1997. He also served as the program chairs for the International Conference on Parallel and Distributed Computing Systems (PDCS), 2000 and PDCS 2001. He received Teaching award in 1998.