Proceedings of the IASTED International Conference
Parallel and Distributed Computing and Systems (PDCS '99)
November 3-6, 1999 – Cambridge, MA, USA

Rate-Optimal Graph Transformation via Extended Retiming and Unfolding*

Timothy W. O'Neil Edwin H.-M. Sha
Dept. of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556

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.

Index terms: Scheduling, Data-flow graphs, Retiming, Unfolding, Graph Transformation, Timing Optimization

1 Introduction

Because the most time-critical parts of real-time or computation-intensive 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)* [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). This model was originally used in circuitry [4] and has since become popular in other fields.

In our previous work [6,7], we proposed an efficient algorithm, *extended retiming*, which transforms a DFG into an equivalent graph with maximum parallelization and minimum schedule length. However, 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* [8]. 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 in 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 [4] to optimize the throughput of synchronous circuits, and has since been used extensively in many diverse areas. We have shown previously that neither unfolding [5] nor this traditional form of retiming [6] can produce optimal results when applied individually, but the combination will achieve optimality [2].

To illustrate these ideas, consider the example of Figure 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 is of our current iteration depends on data produced by task B two iterations ago. This representation of such a dependency is called a *delay* on the edge of the DFG.

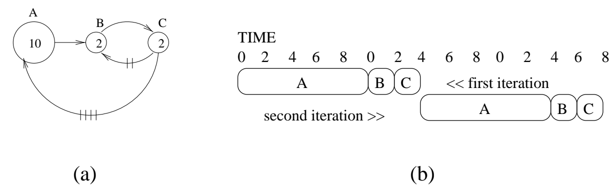

Figure 1: (a) A data-flow graph; (b) The schedule for the DAG part of Figure 1(a).

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 Figure 1(b), the iteration period of this graph is also 14. However, if we were to remove a delay from (C, 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. 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

*This work is partially supported by NSF grants MIP95-01006 and MIP97-04276, and by the A. J. Schmitt Foundation.

of retiming. The maximum of all such ratios, called the *iteration bound*, acts as a lower bound on the iteration period. In the case of Figure 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.

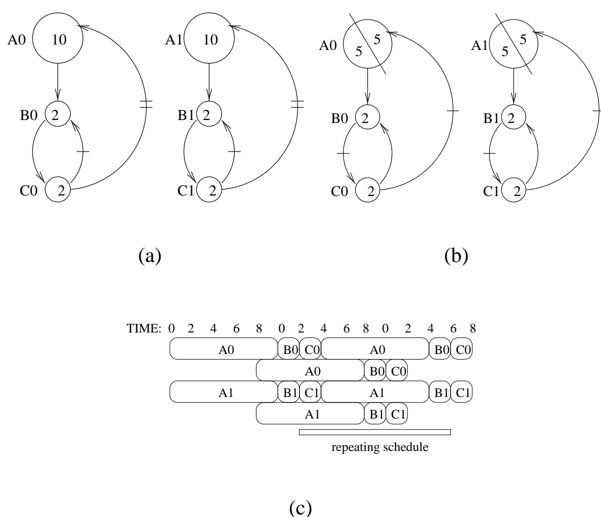

Figure 2: (a) The DFG of Figure 1(a) unfolded by a factor of 2; (b) Figure 2(a) retimed by extended retiming to be rate-optimal; (c) The optimal schedule for the retimed graph.

As an example, let's unfold the graph of Figure 1(a) by a factor of 2, as shown in Figure 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 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 Figure 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 Figure 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 the correct representation of the graph's schedule, especially when we assume a pipelined implementation.

An unfolding of 2 combined with extended retiming optimizes the graph of Figure 1(a). If we limit ourselves to traditional retiming, we must unfold the original DFG four times. After we retime 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. First, 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. Also, 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.

In this paper, 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. We modify our original definition of extended retiming to accommodate the possibility that multiple delays are placed inside a node. This permits us to combine extended retiming with unfolding. 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. 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 under a pipelined design. To the best of our knowledge, this is the first method that can do this.

2 Background & Extended Retiming

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 [1,8] heavily as we establish our new results.

Recall that 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.

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 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 general, an edge (u_i, v_j) having d delays in the unfolded graph represents a precedence relation between node u in the ith 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 [2].

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

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

Finally, a classic result from [4] 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 [2].

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

1. *$cl(G_f) = \max\{T(p) : p \in G$ is a path with $D(p) < f\}$.*
2. *$cl(G_f) \leq c$ iff $D(p) \geq f$ $\forall$ paths $p \in G$ with $T(p) > c$.*

An *extended (*or *f-extended) retiming* of a DFG $G = \langle V, E, d, t \rangle$ is a function $r : V \to \mathbf{Z} \times \mathbf{Q}^f$ where, for all $v \in V$, $r(v) = i + \left(\frac{r_1}{t(v)}, \frac{r_2}{t(v)}, ..., \frac{r_f}{t(v)} \right)$ for some integers $i, r_1, r_2, ..., r_f$ where $0 \leq r_k < t(v)$ for $k = 1, 2, ..., f$. 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 $\frac{1}{t(v)}(r_1, r_2, ..., r_f)$ or as a single fraction $\frac{r_1}{t(v)}$ 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 $\imath_r(v)$ to denote the value of this integer part, while $\Re_r(v)$ 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 Figure 3(a). A 3-extended retiming with $r(A) = 1 + \frac{1}{10}(0, 3, 7)$, $r(B) = 1$ and $r(C) = 0$ results in the retimed graph of Figure 3(b). For this retiming, $\imath_r(A) = \imath_r(B) = 1$ and $\imath_r(C) = 0$, while $\Re_r(A) = 2$ and $\Re_r(B) = \Re_r(C) = 0$.

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. Furthermore, we previously 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 $D(p)$ to count the delays both among the nodes and along the edges of p, we can easily obtain these properties:

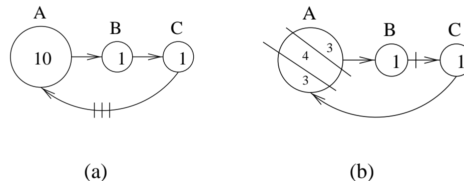

Figure 3: (a) Another sample DFG; (b) This DFG retimed to have clock period 4

Lem 2.1 *Let G be a DFG without split nodes and r an extended retiming. Then the retimed delay count on:*

1. *edge $e = (u, v)$ is $d(e) + \imath_r(u) - \imath_r(v) + \Re_r(u)$.*
2. *path $p : u \Rightarrow v$ is $D(p) + \imath_r(u) - \imath_r(v) + \Re_r(u)$.*
3. *cycle $\ell \in G$ is $D(\ell)$.*

Given an edge $e = (u, v)$, we use $d_r(u \to v)$ 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 Figure 3(b), let $e_1 = (A, B)$, $e_2 = (B, C)$ and $e_3 = (C, A)$. Then $d_r(A \to B)$ and $d_r(C \to A)$ are each 2 due to a split end-node, even though $d_r(e_1)$ and $d_r(e_3)$ are each zero. As with traditional retiming, an extended retiming is *legal* if $d_r(e) \geq 0$ for all edges $e \in E$ and *normalized* if $\min_v \imath_r(v) = 0$. Note two things. First, if r is an extended retiming, then $d_r(u \to v) = d_r(e) + \Re_r(u) + \Re_r(v)$ for all edges $e = (u, v) \in E$. Also, any extended retiming can be normalized by subtracting $\min_v \imath_r(v)$ from all values $\imath_r(v)$.

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 Figure 3(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 $\Re_r(u)$ delays, we can see that we are left with $\Re_r(u) + 1$ pieces which we can denote in order as u^0, u^1,..., $u^{\Re_r(u)}$. 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 $\Re_r(v) - k$ delays separating v^k, v^{k+1},..., $v^{\Re_r(v)}$. In short, $D_r(u^j \Rightarrow v^k) = D(p) + \imath_r(u) - \imath_r(v) + \Re_r(u) - \Re_r(v) + k - j$, 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 $v^{\Re_r(v)}$. Finally, note that, in a graph G with split nodes, $cl(G)$ is the maximum computation time among all zero-delay *subpaths* of G.

3 Unfolding Then Retiming

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

To illustrate our idea, let's unfold the graph of Figure 3(a) two times to produce the graph of Figure 4(a). We now try to achieve rate optimality by using the method of [2] to schedule this graph with a cycle period of 8, as shown in Figure 4(c). Note that we've unfolded this graph as much as we want and will do no further unfolding. Thus, we can apply the result from [7], 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 $r(A0) = 1\frac{7}{10}$, $r(A1) = 1\frac{3}{10}$, $r(C1) = 0$, and $r(B0) = r(B1) = r(C0) = 1$, whose application to the graph in Figure 4(a) results in the graph in Figure 4(b). If G is the graph in Figure 3(a), then the graph in Figure 4(b) which has first been unfolded then retimed is denoted as $(G_f)_r$.

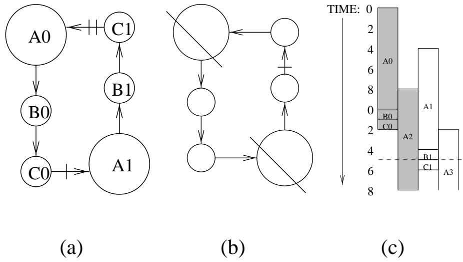

Figure 4: (a) The DFG of Figure 3(a) unfolded twice; (b) This DFG retimed; (c) This DFG's schedule with cycle period 8

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,

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

Thus, for our example, $r(A) = 1\frac{7}{10} \oplus 1\frac{3}{10} = 2 + \frac{1}{10}(7,3)$, $r(B) = 1 \oplus 1 = 2$ and $r(C) = 1 \oplus 0 = 1$. Normalizing this and reordering the 2-tuple into ascending order gives us a 2-extended retiming of $r(A) = 1 + \frac{1}{10}(3,7)$, $r(B) = 1$ and $r(C) = 0$, a retiming similar to the one that we found earlier to achieve the graph in Figure 3(b). (This graph, the version of G from Figure 3(a) which is retimed then unfolded to achieve optimality, is denoted as $G_{r,f}$.)

Now that we've established what we want to do, we can apply these ideas to the general case and prove the following theorem as in [5].

Thm 3.1 *Let G be a DFG 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 $cl\left((G_f)_{r_f}\right) \leq c$, $\exists$ a legal extended retiming r on the original G such that $cl(G_{r,f}) \leq c$.*

4 Retiming Then Unfolding

The first half of our desired result was fairly simple; this half will be much more complicated. We begin by showing our desired result for *traditional* retimings.

Thm 4.1 *Let G be a DFG without split nodes, c a cycle period and f an unfolding factor. For each legal retiming r on G such that $cl(G_{r,f}) \leq c$, $\exists$ a legal retiming r_f on the unfolded graph G_f such that $cl((G_f)_{r_f}) \leq c$.*

The proof of this theorem appears in [2], so we will only briefly discuss it. 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. 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 $e_{r,f}$ 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 2.1(1) for traditional retimings, our condition becomes

$$r_f(u_i) - r_f(v_j) = d_{r,f}(e_{r,f}) - d_f(e_f) \, \forall e_f = (u_i, v_j). \quad (2)$$

Since (2) constitutes a consistent linear system with integer solution r_f, out theorem is proved.

We now wish to prove the above theorem for extended retimings by reducing this case to the traditional one. 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 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 Figure 5(a) below. Recall that the 2-extended retiming with $r(A) = 1 + \frac{1}{10}(3,7)$, $r(B) = 1$ and $r(C) = 0$ applied to G achieves the optimal clock 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 3, 4 and 3, respectively, while the other nodes remain whole. So, if we split A as shown in Figure 5(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 nodes through node A^0, leaving one on each of the edges (A^0, A^1) and (A^1, A^2) while the lone remaining delay passes through A^2. Thus the traditional retiming ρ on $X^{G,r}$ defined as $\rho(A^0) = 3$, $\rho(A^1) = 2$, $\rho(A^2) = \rho(B) = 1$ and $\rho(C) = 0$ is equivalent to the extended retiming

r on G and results in a retimed extended graph with clock period 4, as shown in Figure 5(c). In keeping with our previous notation, the extended graph $X^{G,r}$ retimed by ρ will be named $X_{\rho}^{G,r}$.

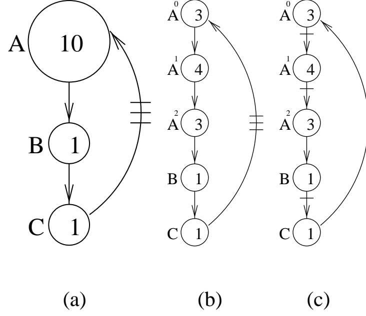

Figure 5: (a) Our sample DFG; (b) The resulting extended graph; (c) This extended graph retimed to have clock period 4

When we unfold $X_{\rho}^{G,r}$ twice, as shown in Figure 6(a), the resulting graph has a clock period of 2×4 or 8 and unfolding factor $f = 2$. According to our theorem above, we must be able to construct a retiming ρ_f for the twice-unfolded extended graph (shown in Figure 6(b)) which results in a clock period of 8. We do so by matching each edge $e_f = (u_i, v_j)$ of $X_f^{G,r}$ (Figure 6(b)) with an edge $e_{\rho,f} = \left(u_{(i-r(u)) \bmod f}, v_{(j-r(v)) \bmod f}\right)$ from $X_{\rho,f}^{G,r}$ (Figure 6(a)) as in [2], then using this matching to construct the linear system of equations (2) 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 with $\rho_f(A_0^0) = 2$, $\rho_f(C_0) = \rho_f(A_1^2) = \rho_f(B_1) = \rho_f(C_1) = 0$ and $\rho_f(A_0^1) = \rho_f(A_0^2) = \rho_f(B_0) = \rho_f(A_1^0) = \rho_f(A_1^1) = 1$, which leads to the retimed graph $(X_f^{G,r})_{\rho_f}$ shown in Figure 7(a).

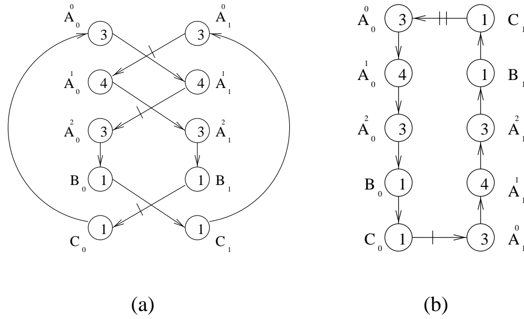

Figure 6: The extended graph unfolded twice: (a) after retiming first; (b) with no retiming

We now use this retiming to construct a retiming r_f for our original unfolded graph. Note that, in Figure 7(a), we have placed a delay on the edge $\left(A_0^0, A_0^1\right)$. 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 three time units, we define our extended retiming so that the fractional part of $r_f(A_0)$ is $\frac{3}{10}$. We also have $\rho_f(A_0^2) = 1$, which calls for us to push a delay through the last piece of A_0 onto the outgoing edge, and so the integer part

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

Table 1: Table of matching edges and linear equations from Figures 6(a) and 6(b).

of $r_f(A_0)$ must be 1. Similarly, the delay on $\left(A_1^1, A_1^2\right)$ separates the last piece of A_1 from the initial part of the node. Since this first part has a total computation time of 7, and since we do not push a delay through A_1^2, we need to have $r_f(A_1) = \frac{7}{10}$. 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 $r_f(A_0) = 1\frac{3}{10}$, $r_f(B_0) = 1$, $r_f(C_0) = 0$, $r_f(A_1) = \frac{7}{10}$ and $r_f(B_1) = r_f(C_1) = 0$. Applying this retiming to our original unfolded graph, shown in Figure 7(b), results in the graph of Figure 7(c), which indeed has a clock period of 8.

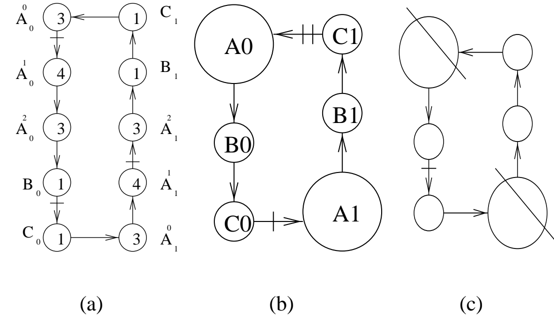

Figure 7: (a) The unfolded extended graph of Figure 6(b) now retimed by ρ_f; (b) Our original graph unfolded twice; (c) This graph retimed

Using the ideas that we've just run through, we can now formally prove our desired result, as in [5].

Thm 4.2 *Let $G = \langle V, E, d, t\rangle$ be a DFG without split nodes, c a cycle period and f an unfolding factor. For every legal extended retiming r on G such that $cl(G_{r,f}) \leq c$, $\exists$ a legal extended retiming r_f on the unfolded graph G_f such that $cl((G_f)_{r_f}) \leq c$.*

5 Min. 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* [9] 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 stating this link between a graph's clock period and iteration bound which is proven in [5]:

Lem 5.1 *Let G be a DFG without split nodes, c a cycle period and f an unfolding factor. Then $\exists$ a legal extended retiming r on G such that $cl(G_{r,f}) \leq c$ iff $B(G) \leq \frac{c}{f}$.*

We this in hand we can now show, as in [5]:

Thm 5.1 *Let G be a DFG without split nodes. Let ℓ be a critical cycle of G, i.e. $B(G) = \frac{T(\ell)}{D(\ell)}$. Let $g = \gcd\{T(\ell), D(\ell)\}$. Then $\frac{D(\ell)}{g}$ is the minimum rate-optimal unfolding factor for G.*

In short, to find the rate-optimal unfolding factor of a data-flow graph G, we compute $B(G)$ (a polynomial-time operation [3]) and reduce the resulting fraction to lowest terms. The denominator of this fraction is our desired unfolding factor.

6 Experimental Results

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 [4] as applying a *slowdown* of 2 to our original circuit. The result is pictured in Figure 8(a).

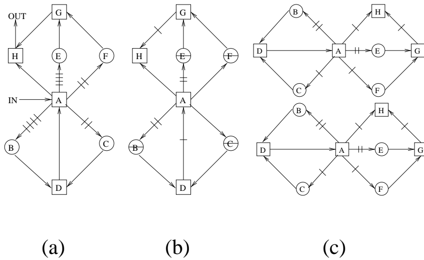

Figure 8: (a) A 2-slow second order IIR filter; (b) This graph optimized by extended retiming; (c) This graph optimized by traditional retiming

The resulting circuit has an iteration bound of 3, and can be retimed via extended retiming to achieve this clock period as in Figure 8(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 Figure 8(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: 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. 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.

Benchmark	Comp. Time		Slow-down	Iter. Bound	Min. Optimal Unf. Factor		Iter. Pd. w/ **Bold** U. F.	
	$+$	$\times$			**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	$\frac{11}{2}$	**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

Table 2: Experimental results for common circuits

7 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 3.1 and 4.2. We have also 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. Our result in this article indicates that we should be able to find a retiming immediately without unfolding first via a similar method, but we have yet to develop such a procedure.

References

[1] L.-F. Chao. *Scheduling and Behavioral Transformations for Parallel Systems*. PhD thesis, Dept. of Comput. Sci., Princeton Univ., 1993.

[2] L.-F. Chao and E. H.-M. Sha. Scheduling data-flow graphs via retiming and unfolding. *IEEE Trans. Parallel & Distributed Syst.*, 8:1259–1267, 1997.

[3] A. Dasdan and R. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Trans. CAD of Integrated Circuits & Syst.*, 17:889–899, 1998.

[4] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[5] T. O'Neil and E. H.-M. Sha. Combining extended retiming and unfolding for rate-optimal graph transformation. Tech. Rept. 99-12, Univ. of Notre Dame, 1999. Available online at *www.cse.nd.edu/tech_reports/*.

[6] T. O'Neil, S. Tongsima, and E. H.-M. Sha. Extended retiming: Optimal retiming via a graph-theoretical approach. In *Proc. ICASSP-99*, vol. 4, pp. 2001–2004, 1999.

[7] T. O'Neil, S. Tongsima, and E. H.-M. Sha. Optimal scheduling of data-flow graphs using extended retiming. In *Proc. ISCA 12th Int. Conf. Parallel & Distributed Computing Syst.*, pp. 292–297, 1999.

[8] K. Parhi and D. Messerschmitt. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Trans. Comput.*, 40:178–195, 1991.

[9] M. Renfors and Y. Neuvo. The maximum sampling rate of digital filters under hardware speed. *Trans. Circuits & Sampling*, CAS-28:196–202, 1981.