

Optimal Scheduling of Data-Flow Graphs Using Extended Retiming

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

Abstract

Many iterative or recursive applications commonly found in DSP and image processing applications can be represented by *data-flow graphs* (DFGs). 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 the schedule length. One of the most effective of these techniques is *retiming*. In this paper, we demonstrate that the traditional retiming technique does not always achieve optimal schedules (although it can be used in combination with other techniques to do so) and propose a new graph-transformation technique, *extended retiming*, which will.

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

1 Introduction

Many iterative or recursive applications, such as image processing, DSP and PDE simulations, can be represented by *data-flow graphs*, or DFGs [4]. The nodes of a DFG represent tasks, while edges between nodes represent data dependencies among the tasks, either within *iterations* (an execution of all tasks) or between iterations. To model repeated steps within an algorithm, a DFG may contain loops. To meet the desired throughput, it becomes necessary to use multiple processors or multiple functional units. Due to the expense of such units, it is important for us to minimize the number of processors we involve during execution, while maximizing the use of those processors that we do include. The process of assigning a starting time and processor to each event in the DFG, known as *scheduling*, becomes a vital step in this process.

There are two common approaches for system-level synthesis and scheduling of parallel systems:

1. We can explicitly schedule the DFG as-is.
2. We can first apply a graph transformation technique to the DFG in order to maximize the degree of parallelism, then schedule the acyclic (or *DAG*) part of the resulting graph.

There are many methods for doing scheduling [2,6,8]; hence the focus of our study will be the optimization of the DFG via graph transformation. We will later show that the second of these two methods is preferable to the first because the schedule it produces requires fewer resources.

The execution of all tasks of a DFG is called an *iteration*, with the length of time it takes to complete an iteration called the

schedule length of the DFG. While there are many graph transformation techniques available to us, it is possible to find graphs for which the current techniques will not produce a transformed DFG having minimum schedule length. We will demonstrate that in this paper, as well as propose a new transformation technique which does deliver optimal results. When compared with the traditional methods, our new technique quickly and easily produces a transformed graph without increasing the size of the DFG.

A great deal of research has been done attempting to optimize the schedule of tasks for an application after applying various graph transformation techniques to the application’s DFG. One of the more effective of these techniques is *retiming* [1,7], where delays are redistributed among the edges so that the application’s function remains the same, but the length of the longest zero-delay path, called the *clock period* of the DFG G and denoted $cl(G)$, is decreased. After applying simple DAG scheduling, the length of the schedule for graph G is given by $cl(G)$. Another popular technique is *unfolding*. Unfortunately, neither retiming nor unfolding produces optimal results when applied individually, although we may combine these techniques to achieve optimality [4]. Unfolding has an additional weakness, in that it greatly increases the size of the DFG that you wish to schedule. Therefore, we will focus on retiming in this paper.

To illustrate, consider the example of a DFG G_1 given in Figure 1(a). The numbers above the nodes of G_1 represent computation times for the individual tasks. The short bar-line cutting the edge from node B to node C , hereafter denoted by the ordered pair (B, C) , represents an inter-iteration dependency between these nodes. In other words, the line cutting (B, C) tells us that task C of our current iteration is dependent on data produced by task B of the previous iteration. This representation of such a dependency is called a *delay* on the edge of the DFG.

We can see that the DFG of Figure 1(a) has $cl(G_1) = 4$, obtained from the edge (A, B) . A delay can be removed from edge (C, A) and placed on edge (A, B) to create a retimed version of G_1 , denoted G_{1r} and pictured in Figure 1(b), with the property that $cl(G_{1r}) = 3$. The corresponding static schedule is found in Figure 2. We see that our repeating schedule calls for nodes B and C to begin execution together, with node A starting upon the conclusion of C . Because we have retimed node A once by moving a delay from its incoming edge to its outgoing edge, the first copy of A is not actually part of the repeating schedule; its purpose is merely to “prime the pump” so that the repeating part of the schedule may commence. We call such tasks the *prologue* of the static schedule. The polynomial-time algorithm of [7] tells us that 3 is the minimum clock period we can achieve by retiming G_1 .

As we said earlier, there is a second method that can be used

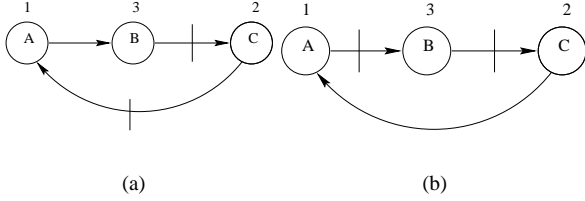


Figure 1: (a) The data-flow graph G_1 ; (b) G_1 retimed to have $cl(G_{1_r}) = 3$

to minimize the length of a schedule. Chao and Sha [3] presented a unified algorithm which is referred to as *DFG scheduling* in this paper. We use this method to construct a schedule in Figure 2 for the DFG of Figure 1(a) which satisfies all data dependencies among the nodes of the graph. Since the repeating part of the schedule of tasks executes every 3 clock cycles, we say that this is a static schedule with *cycle period* 3, denoted $cy(G_1) = 3$. Note that each task's starting time must be repeated every $cy(G_1)$ time steps. Thus, in this case, the length of the schedule is given by $cy(G_1)$. Furthermore, each task is implemented at the beginning of a time step and not between time steps; hence we say that this schedule is *integral*. Finally, see that consecutive instances of a task do not overlap, making this a *non-pipelined implementation* of the static schedule. This paper assumes the use of this integral/non-pipelined model.

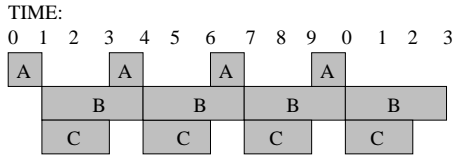


Figure 2: The schedule for G_1 with $cy(G_1) = cl(G_{1_r}) = 3$

Although DFG scheduling is a polynomial-time algorithm which gives an optimal cycle period for systems, the schedule it yields typically requires more resources than the one we derive by retiming first. Therefore, in this paper, we present a graph transformation technique which yields the same optimal results as DFG scheduling. The graph that results from our approach can then be directly used as the input to the next stage of the resource-constrained scheduling.

We note that we have discovered a retiming and static schedule such that $cl(G_{1_r}) = cy(G_1)$ for the graph of Figure 1(a). We could ask ourselves whether we can always find such a retiming. In this paper, we propose a new algorithm which will always yield a schedule whose minimal length is given by the clock or cycle period.

This paper demonstrates the following:

1. We show that traditional retiming and static scheduling are not equivalent. Indeed, DFG scheduling is a more powerful technique than retiming for minimizing the clock or cycle period of a DFG.
2. We propose a new graph transformation technique, *extended retiming*, which we will show produces the same clock

period as does optimal DFG scheduling while consuming fewer resources when combined with DAG scheduling.

3. Because of this equivalence, we are able to design a constant-time algorithm for verifying the existence of an extended retiming r which will make $cl(G_r) \leq c$ for a given integer c .
4. We demonstrate two algorithms, one based on graphs and the other on static schedules, for finding extended retimings.

2 Definitions and Basic Properties

A *data-flow graph* (DFG) is a finite, directed, weighted graph $G = \langle V, E, d, t \rangle$ where V is the vertex set of computation nodes; $E \subseteq V \times V$ is the edge set, representing precedence relations among the nodes; $d : E \rightarrow \mathbf{Z}$ is a function with $d(e)$ the delay count (i.e., number of delays) on edge e ; and $t : V \rightarrow \mathbf{Z}$ is a function with $t(v)$ the computation time of node v . 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. As we said, the *clock period* $cl(G)$ of a DFG G is the computation time of the longest zero-delay path; in other words $cl(G) = \max\{T(p) : p \in G \text{ is a path with } D(p) = 0\}$.

A *retiming* r (sometimes referred to in this paper as a *traditional retiming*) of a DFG G is a function from v to the integers which specifies a transformation of G . It labels each vertex and results in a new *retimed graph* $G_r = \langle V, E, d_r, t \rangle$ where $d_r(e) = d(e) + r(u) - r(v)$ for each edge $e = (u, v)$. For example, the function with $r(A) = 1$ and $r(B) = r(C) = 0$ retimes the graph in Figure 1(a) to the graph in Figure 1(b). If, in addition, we have $\min_v r(v) = 0$, we say that the retiming is *normalized*.

In our method, we first construct the *scheduling graph* $G^s = \langle V, E, w, t \rangle$ by reweighting each edge $e = (u, v)$ according to the formula $w(e) = d(e) - \frac{t(u)}{c}$. Figure 3 shows the scheduling graph of our example when $c = 3$.

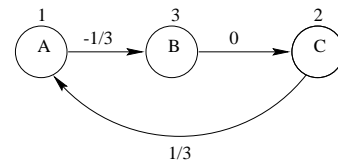


Figure 3: The scheduling graph of G_1 with $c = 3$

It can be shown that, if c is a feasible clock period, then the scheduling graph contains no negative-weight cycles. Thus, we further alter G^s by adding a node v_0 and zero-weight directed edges from v_0 to every other node in G . 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 Figure 3, we note that $sh(A) = 0$ and $sh(B) = sh(C) = -\frac{1}{3}$. It takes $O(|V||E|)$ time to compute $sh(v)$ for every node v [3].

As we've stated, an *iteration* is simply an execution of all nodes of a DFG once. The average computation time of an iteration is called the *iteration period* of the DFG. If a DFG G contains a loop, then the iteration period is bounded from below by the *iteration bound* [9] of G , which is denoted by $B(G)$ and defined to

be the maximum time-to-delay ratio of all cycles in G . For example, the graph of Figure 1(a) contains only one loop, which has 2 delays and a total computation time 6; thus $B(G1) = 3$ for this graph. The schedule for this graph which is displayed in Figure 2 has an iteration period of 3. In this situation, when the iteration period of a static schedule equals the iteration bound of the DFG, we say that the schedule is *rate-optimal*. According to Lemma 3.1 of [3], $B(G) \leq c$ if and only if the scheduling graph G^s contains no cycles having negative weight, where c is a clock period for the DFG G .

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 Figure 2 is $S(v, i) = 3(i - sh(v))$ for all nodes v and iterations i , where the values for $sh(v)$ are derived from the graph of Figure 3.

A legal schedule is a *repeating schedule for cycle period c* if $s(v, i + 1) = s(v, i) + c$ for all nodes v and iterations i . It's easy to see that $S(v, i)$ 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*.

We can now prove the following:

Theorem 2.1 *Let G be a DFG and c a positive integer. Then there exists a legal, integral, repeating, static schedule under the non-pipelined implementation with cycle period c if and only if $B(G) \leq c$ and $t(v) \leq c$ for all nodes v .*

Proof: Theorem 2.3 of [3] states that any legal static schedule with cycle period c can be implemented under a non-pipelined design if and only if $t(v) \leq c$ for all nodes v . Theorem 3.5 of the same paper shows that $S(v, i) = c \cdot (i - sh(v))$ is a legal integral schedule with cycle period c if and only if $B(G) \leq c$. \square

We can produce the static DFG schedule in Figure 2 by constructing the scheduling graph (Figure 3) and then computing $sh(v)$ for each of the nodes. Following this, we apply the formula from the above proof to create the schedule; for this example $S(A, 0) = 0$ and $S(B, 0) = S(C, 0) = 3 \cdot \frac{1}{3} = 1$. We will soon see that our proposed extended retiming technique can achieve a clock period equal to the cycle period of optimal DFG scheduling.

3 Extended Retiming

We now demonstrate that traditional retiming will not necessarily result in an optimal schedule and devise a form of retiming that is equivalent to DFG scheduling.

Consider the DFG $G2$ in Figure 4(a). The minimum clock period of this graph is 5. Indeed, it's easy to see that, no matter how we position the delays, we have one zero-delay edge with computation time at least 5. However, the rate-optimal schedule on $G2$ shown in Figure 4(b) has cycle period 4.

In general, the minimum value of $cy(G)$ is less than or equal to the minimum value of $cl(G_r)$. The technique of DFG scheduling is more powerful than the traditional technique of retiming in

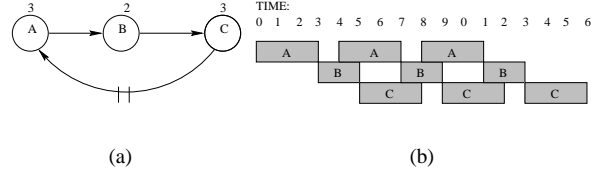


Figure 4: (a) The DFG $G2$; (b) The schedule of $G2$ with $cy(G2) = 4$

some sense. However, the advantages of using graph transformation techniques are clearly visualized and easily understood from working with graph models. Therefore, we now develop a new graph transformation technique.

Suppose that we're allowed to split a node into two pieces. For example, we could split node B in half to get the DFG of Figure 5. We see that we can now move a delay from edge (C, A) to the space between the halves of B and achieve a retimed graph with clock period 4. Our new concept will be called *extended retiming*.

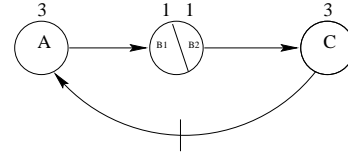


Figure 5: Graph $G2$ with node B split so that $cl(G2_r) = 4$

An *extended retiming* of a DFG $G = \langle V, E, d, t \rangle$ is a function $r: V \rightarrow \mathbf{Q}$ such that $t(v) \cdot r(v) \in \mathbf{Z}$ for all $v \in V$. From this definition, $r(v)$ can be viewed as consisting of an integer part and a fractional part. The integer part $\lceil r(v) \rceil$ is the number of delays pushed to each *outgoing* edge of v , while the fractional part conveys the position of a delay within a split node. Therefore, the value $\lceil r(v) \rceil$ is the number of delays drawn *from* each *incoming* edge of v , and if $\lceil r(v) \rceil \neq \lfloor r(v) \rfloor$, a delay is left inside node v which splits it into two subnodes. For example, an extended retiming with $r(A) = 1$, $r(B) = \frac{1}{2}$ and $r(C) = 0$ for the graph of Figure 4(a) achieves the retimed graph described for Figure 5.

As with standard retiming, we will denote the DFG retimed by r as $G_r = \langle V, E, d_r, t \rangle$, where $d_r(u \rightarrow v) = d(e) + \lceil r(u) \rceil - \lfloor r(v) \rfloor$ is the retimed delay count of edge $e \in E$. Note that this definition includes those delays inside the end nodes; we will continue to use the notation $d_r(e)$ to refer to the number of delays on e *not including* delays contained within split nodes. It is easy to show that $d_r(e) = d(e) + \lceil r(u) \rceil - \lfloor r(v) \rfloor$. An extended retiming is *legal* if $d_r(e) \geq 0$ for all edges $e \in E$ and *normalized* if $0 \leq \min_v r(v) < 1$. In order to normalize an extended retiming, we must subtract $\min_v \lfloor r(v) \rfloor$ from all values $r(v)$. From the definition of d_r , we can easily see that the retimed delay count on the path $p: u \Rightarrow v$ is $D_r(p) = D(p) + \lceil r(u) \rceil - \lfloor r(v) \rfloor$, and that the retimed delay count on the cycle $\ell \in G$ is $D_r(\ell) = D(\ell)$.

The proof of the major theorem in this section will require a short study of the properties of $sh(v)$ under certain conditions.

Lemma 3.1 *Let $G = \langle V, E, d, t \rangle$ be a DFG and c a positive integer. Let $X = \min_v sh(v)$. For all nodes v define $R(v) = c \cdot ((sh(v) - X) - \lfloor sh(v) - X \rfloor)$. Then for every path $p: u \Rightarrow v$:*

1. $\lfloor sh(v) - X \rfloor \leq \lfloor sh(u) - X \rfloor + D(p)$.
2. If $R(u) < T(p) - t(v)$, then $\lfloor sh(v) - X \rfloor \leq \lfloor sh(u) - X \rfloor + D(p) - 1$.
3. Assume that $T(p) > c$. If $R(v) \geq t(v)$ then $\lfloor sh(v) - X \rfloor \leq \lfloor sh(u) - X \rfloor + D(p) - 1$.
4. Assume that $R(u) = 0$ and $T(p) > c$. If $R(v) \geq t(v)$ then $\lfloor sh(v) - X \rfloor \leq \lfloor sh(u) - X \rfloor + D(p) - 2$.

Proof: For (1), choose $s : v_0 \Rightarrow u$ and $q : v_0 \Rightarrow v$ to be the paths in the modified scheduling graph G^s such that $sh(u) = W(s)$ and $sh(v) = W(q)$. It is easy to see by the Triangle Inequality that

$$sh(v) \leq sh(u) + W(p) \leq sh(u) + D(p) - \frac{T(p) - t(v)}{c}.$$

Since $T(p) - t(v) > 0$, $sh(v) - X < sh(u) - X + D(p)$ for any path $p : u \Rightarrow v$ and any constant X . Taking the floor of both sides now yields our desired result. The proofs of the remaining parts are similar. \square

If we substitute edges for paths, we would deduce analogous results.

The equivalence of a traditional retiming and the absence of negative cycles in the scheduling graph was established for unit-time DFGs in [7]. We will now prove a similar relationship for *general-time* DFGs. In the process we will also develop our first algorithm for finding an extended retiming for a DFG.

Theorem 3.1 *Let $G = \langle V, E, d, t \rangle$ be a (general-time) DFG. Let c be a positive integer with $t(v) \leq c$ for all v . Then there is a legal extended retiming r on G such that $cl(G_r) \leq c$ if and only if the scheduling graph G^s contains no negative-weight cycle.*

Proof: Assume $cl(G_r) \leq c$. $B(G_r) \leq cl(G_r)$, and since retiming does not affect the iteration bound, $B(G) = B(G_r) \leq c$. This implies that $\frac{T(\ell)}{D(\ell)} \leq c$ for all cycles ℓ in G , which implies that $D(\ell) - \frac{T(\ell)}{c} \geq 0$. Since a cycle in G is also one in G^s , we derive $W(\ell) \geq 0$ for all cycles ℓ in G^s . Therefore, G^s has no negative weight cycles.

On the other hand assume that G^s contains no negative-weight cycle. Define X and $r(v)$ as in Lemma 3.1. We now define

$$r(v) = \lfloor sh(v) - X \rfloor + \min \left\{ 1, \frac{R(v)}{t(v)} \right\}. \quad (1)$$

We want to show that r is a legal extended retiming and that $cl(G_r) \leq c$.

- Choose any edge $e = (u, v)$ in G . Then $sh(v) \leq sh(u) + d(e) - \frac{t(u)}{c}$, and so $d(e) + sh(u) - sh(v) \geq \frac{t(u)}{c} > 0$. There are now two cases.

1. Let $R(u) \geq t(u)$. By definition we know that $d_r(e) = d(e) + \lceil r(u) \rceil - \lceil r(v) \rceil$, and so

$$d_r(e) \geq d(e) + (\lfloor sh(u) - X \rfloor + 1) - (\lfloor sh(v) - X \rfloor + 1) \geq 0.$$
2. If $R(u) < t(u)$, then by the analogue of Lemma 3.1(2) for edges,

$$\lceil r(u) \rceil \geq \lfloor sh(u) - X \rfloor \geq \lfloor sh(v) - X \rfloor - d(e) + 1.$$

So in this case

$$\begin{aligned} d_r(e) &\geq d(e) + (\lfloor sh(v) - X \rfloor - d(e) + 1) \\ &\quad - (\lfloor sh(v) - X \rfloor + 1) = 0. \end{aligned}$$

In any case, $d_r(e) \geq 0$ and r is a legal extended retiming.

- Let $p : u \Rightarrow v$ be a path in G with $T(p) > c$. By Lemma 4 of [7], it suffices to show that $D_r(p) \geq 1$. We know that $sh(v) \leq sh(u) + D(p) - \frac{T(p) - t(v)}{c}$ and so $D(p) + sh(u) - sh(v) \geq \frac{T(p) - t(v)}{c} > 0$. We have four cases to consider.

1. If $sh(u) - X$ and $sh(v) - X$ are both integers, then

$$\begin{aligned} D_r(p) &= D(p) + \lceil r(u) \rceil - \lceil r(v) \rceil \\ &= D(p) + sh(u) - sh(v). \end{aligned}$$

This quantity is an integer which is greater than or equal to a *strictly* positive fraction; hence we conclude that $D_r(p) \geq 1$ in this case.

2. Assume that $sh(u) - X$ is integral but $sh(v) - X$ is not. Hence $\lceil r(u) \rceil = r(u) = \lfloor sh(u) - X \rfloor$ in this case. We know that $R(u) = 0$ but have two possibilities for $R(v)$.

- (a) If $R(v) < t(v)$ then $r(v) = \lfloor sh(v) - X \rfloor \leq \lfloor sh(u) - X \rfloor + D(p) - 1$ by Lemma 3.1(2); hence

$$\begin{aligned} D_r(p) &\geq D(p) + \lfloor sh(u) - X \rfloor \\ &\quad - (\lfloor sh(u) - X \rfloor + D(p) - 1). \end{aligned}$$

- (b) If $R(v) \geq t(v)$, then by Lemma 3.1(4),

$$\lceil r(v) \rceil \leq \lfloor sh(u) - X \rfloor + d(e) - 1.$$

In any case $D_r(p) \geq 1$.

3. If $sh(u) - X$ is not integral but $sh(v) - X$ is, then $r(u) = \lfloor sh(u) - X \rfloor + 1$ and $r(v) = \lfloor sh(v) - X \rfloor$; hence

$$D_r(p) = D(p) + \lfloor sh(u) - X \rfloor + 1 - \lfloor sh(v) - X \rfloor.$$

Therefore $D_r(p) \geq \lfloor D(p) + sh(u) - sh(v) \rfloor + 1 \geq 1$.

4. Finally, if neither $sh(u) - X$ nor $sh(v) - X$ is integral, we have two subcases.

- (a) If $R(v) < t(v)$ then $\lceil r(v) \rceil = \lfloor sh(v) - X \rfloor$; thus

$$\begin{aligned} D_r(p) &\geq D(p) + \lfloor sh(u) - X \rfloor + 1 \\ &\quad - \lfloor sh(u) - X \rfloor. \end{aligned}$$

- (b) If $R(v) \geq t(v)$ then $\lceil r(v) \rceil = \lfloor sh(v) - X \rfloor + 1$. Since $\lceil r(u) \rceil = \lfloor sh(u) - X \rfloor + 1$ we obtain

$$D_r(p) = D(p) + \lfloor sh(u) - X \rfloor - \lfloor sh(v) - X \rfloor.$$

Thus, by Lemma 3.1(3),

$$\begin{aligned} D_r(p) &\geq D(p) + \lfloor sh(u) - X \rfloor \\ &\quad - (\lfloor sh(u) - X \rfloor + D(p) - 1). \end{aligned}$$

In any case $D_r(p) \geq 1$, and by Lemma 4 of [7], $cl(G_r) \leq c$. \square

For example, when applied to the graph of Figure 4(a) with $c = 4$, the retiming described in this theorem yields $r(A) = \frac{4}{3}$, $r(B) = 1$ and $r(C) = 0$. Since the most time-consuming step in this process is the computation of $sh(v)$ for all nodes v , it takes $O(|V||E|)$ time to calculate an extended retiming this way. Let's now summarize what we've proven so far:

Theorem 3.2 *Let G be a DFG and c an integer. The following are equivalent:*

1. There is a legal extended retiming r on G s.t. $cl(G_r) \leq c$.
2. There exists a legal, repeating, static schedule for G under the integral/non-pipelined model with cycle period c .
3. G^s contains no cycle having negative delay count and $t(v) \leq c$ for all nodes v of G .
4. $B(G) \leq c$ and $t(v) \leq c$ for all nodes v of G .

See that we have also developed a constant-time algorithm for checking the legality of a clock period. If we know the values of $B(G)$ and the maximum $t(v)$ in advance, we simply verify that they are all smaller than our chosen c and can know almost immediately if there's an extended retiming which gives us $cl(G_r) \leq c$. (If we don't we can compute $B(G)$ in polynomial time [5].) This theorem also shows that extended retiming can produce a clock period as low as the cycle period obtained by DFG scheduling. Therefore, both techniques are equivalent.

4 Finding an Extended Retiming from a Static Schedule

We now wish to present an alternate approach to finding an extended retiming, this one based on a legal schedule (not necessarily obtained from DFG scheduling) and not on the scheduling graph. This approach will be easier to visualize and, hopefully, more intuitive.

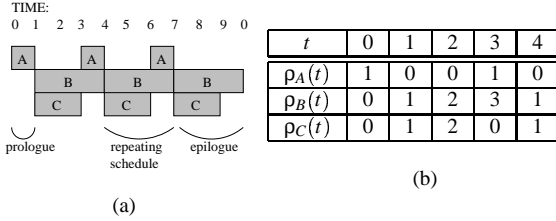


Figure 6: (a) The schedule for $G1$ with $cy(G1) = 3$; (b) The values of ρ_v for $G1$ with $c = 3$

When we construct a repeating schedule for the graph of Figure 1(a) with $c = 3$, we observe that the prologue in the resulting schedule (shown in Figure 6(a)) consists of one copy of node A and no copies of B or C . This corresponds to the retiming function $r(A) = 1$, $r(B) = r(C) = 0$ that we found to transform the graph of Figure 1(a) to the graph of Figure 1(b) with clock period 3.

In general, when a delay is pushed through node v , every copy of node v is shifted up one iteration, bumping the first copy of v into the prologue. Hence, if r is a traditional normalized retiming, the number of copies of node v appearing in the prologue is $r(v)$. Furthermore, a static schedule contains one copy of each node. Since there will always be some node not in the prologue, we can distinguish easily where the prologue ends and the repeating part of the schedule begins; we simply look for the last node to enter the static schedule. These results are important to us as we attempt to devise retimings merely by examining static schedules.

We now wish to introduce a different notation which better represents static schedules when we write programs to work

with them. As usual we let \mathbf{Z}_n be the set of integers from 0 to $n - 1$ inclusive. For each node v , we can now define a function $\rho_v : \mathbf{Z}^+ \rightarrow \mathbf{Z}_{t(v)+1}$ by first initializing $\rho_v(x)$ to zero for all positive integers x , then setting $\rho_v(S(v, i)) = 1$, $\rho_v(S(v, i) + 1) = 2, \dots, \rho_v(S(v, i) + t(v) - 1) = t(v)$ for all positive integers i . Since the values of $S(v, i)$ are dependent on the value of the cycle period c , the function depends on what our cycle period is. Figure 6(b) shows the values of the functions ρ_v for the graph of Figure 1(a) with $c = 3$. In light of what we've demonstrated so far, it is easy to see that ρ_v , restricted to the set $\{x \geq S(v, 0)\}$ is a periodic function of period $cl(G)$ for each node v .

We can now define the *state* of graph G at time-step t , denoted $St_G(t)$, to be the $|V|$ -tuple $(\rho_{v_1}(t), \dots, \rho_{v_{|V|}}(t))$. As the composition of functions which are eventually periodic, St_G restricted to the set $\{x \geq \max_v S(v, 0)\}$ is itself periodic of period $cl(G)$. Table 1 shows the state at each time-step of the graph in Figure 1(a). Here the first coordinate of the state reflects the condition of node A , the second that of B , and the third that of C . In order for this notation to make sense, there has to be some ordering of the nodes.

t	$St_{G1}(t), c = 3$	t	$St_{G1}(t), c = 3$
0	(1,0,0)	4	(0,1,1)
1	(0,1,1)	5	(0,2,2)
2	(0,2,2)	6	(1,3,0)
3	(1,3,0)	7	(0,1,1)

Table 1: The state table for $G1$

We now propose an algorithm which uses this schedule to quickly find a retiming for G which makes $cl(G) \leq c$. We do this by working an example. Consider the graph of Figure 1(a) with desired clock period 3. See that $S(A, 0) = 0$, $S(B, 0) = 1$ and $S(C, 0) = 1$ under these circumstances, where S is the schedule of Theorem 2.1. As we noted above, the state function St_{G1} is periodic of period 3 for $t \geq 1$. So, let's take Table 1 and cut it between rows 0 and 1. When we compare this table to the block-diagram of the schedule in Figure 6(a), we see that everything above the cut corresponds to the prologue. We've noted that we can find a legal retiming for a node merely by counting the number of occurrences of that node in the prologue of the static schedule. Hence, it makes sense that we can also find our retiming by counting the number of occurrences of a node in the states above the cut in our table.

We have developed enough insight to propose that a legal extended retiming for a graph can be found merely by cutting the state table above the row numbered by $\max_v S(v, 0)$ and counting the number of occurrences of each node above the cut. Formally, we must show the following:

Theorem 4.1 *Given a DFG G and clock period c with $B(G) \leq c$ and $t(v) \leq c$ for all nodes v . If $S(v, i) = c \cdot (i - sh(v))$, set M to be the maximum of the values $S(v, 0)$ for all nodes v . Then $r(v) = \frac{1}{t(v)} |\{x < M : \rho_v(x) > 0\}|$ is a legal extended retiming such that $cl(G_r) \leq c$.*

Proof: See that our function r is merely counting the total amount of time an occurrence of a node exists in the prologue and dividing by the computation time of the node, i.e. the amount of time one occurrence of a node takes. This produces the number of occurrences of a node in the prologue. Let's use another method to

count the total amount of time an occurrence of a node u exists in the prologue of our static schedule, then divide by $t(u)$ to produce a different function which is equivalent to our proposed retiming.

Let $X = \min_v sh(v)$. There is some node that is the last to make its initial appearance in our schedule; call it w and note that $S(w, 0) = -c \cdot X$. As we've noted, the entrance of w marks the end of the prologue and the beginning of the repeating schedule. Hence, the difference in time between the start of the first occurrence of u and the end of the prologue is $c \cdot (sh(u) - X)$. Since c is the clock period of G , a new occurrence of u begins every c time-steps; hence the number of occurrences of u in the prologue is $sh(u) - X$. There are two components to this value:

1. The number of complete copies of u is represented by the integral part $\lfloor sh(u) - X \rfloor$. We multiply this quantity by $t(u)$ to get the total amount of time complete occurrences of u exists in the prologue.
2. What remains represents the number of partial occurrences of u in the prologue. Since this is a fraction over c , multiply this quantity by c to get the total amount of time taken up by partial instances of u . We must also be careful to insure that this value cannot exceed the total computation time of u .

Thus, we get $t(u) \cdot \lfloor sh(u) - X \rfloor + \min\{t(u), c \cdot ((sh(u) - X) - \lfloor sh(u) - X \rfloor)\}$ time steps when node u is part of the prologue. Now divide by $t(u)$ to get a retiming of:

$$r(u) = \lfloor sh(u) - X \rfloor + \min\left\{1, \frac{c}{t(u)} \cdot ((sh(u) - X) - \lfloor sh(u) - X \rfloor)\right\}.$$

This function $r(v)$ is the same as that of Theorem 3.1. Furthermore, we have shown that $r(v)$ is a function that accurately counts the number of occurrences of a node in the prologue. \square

5 Examples

To demonstrate the effectiveness of our method, let's apply it to several common filters. In each case, we will assume that multiplication and addition each take a certain number of time units to execute, listed in Table 2 under "Cost". To further complicate our examples, we will also multiply the register count of each filter by the quantity listed in parentheses in the first column, referred to in [7] as applying a *slowdown* to the original circuit. We can see from the results, listed in Table 2, that extended retiming gives a smaller clock period in all cases than does traditional retiming.

Benchmark (Slowdown)	Cost		Min. Clock Pd.	
	Add	Mult	Trad. Ret.	Ext. Ret.
All-Pole Lattice Filter (4)	2	2	4	3
All-Pole Lattice Filter (4)	3	4	6	5
Second Order IIR Filter (2)	2	2	4	3
2-Casc. Biquad Filter (2)	3	4	6	5
5th Order Elliptic Filter (2)	2	2	14	13
5th Order Elliptic Filter (2)	3	4	22	21

Table 2: Minimal achievable clock periods for common circuits

6 Conclusion

We have seen that our new method of extended retiming permits us to transform any data-flow graph to one whose clock period matches the cycle period of any of its legal schedules. We have demonstrated that an integer is a legal choice for either the clock or cycle period of the retimed graph provided that it is an upper bound on the computation times of all nodes of the graph and on the iteration bound of the graph. We have shown a very simple method for finding an extended retiming which yields a desired clock period; namely, cut the static schedule at the point where the last node to be scheduled enters and count the number of occurrences of each node to the right of the cut.

We have done these things without considering the effect of unfolding. As we said in the Introduction, many good results have come from combining the traditional retiming and unfolding techniques when optimizing data-flow graphs. It is our future work to study the combination of extended retiming and unfolding.

7 Acknowledgements

This work is partially supported by NSF grants MIP-95-01006 and MIP-97-04276, and by the A.J. Schmitt Foundation.

References

- [1] P.-Y. Calland, A. Darte, and Y. Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Trans. on Parallel and Distributed Systems*, 9:24–35, 1998.
- [2] L.-F. Chao, A. LaPaugh, and E. H.-M. Sha. Rotation scheduling: a loop pipelining algorithm. In *Proc. ACM/IEEE Design Automation Conf.*, pages 566–572, 1993.
- [3] L.-F. Chao and E. H.-M. Sha. Static scheduling for synthesis of DSP algorithms on various models. *Journal of VLSI Signal Processing*, 10:207–223, 1995.
- [4] L.-F. Chao and E. H.-M. Sha. Scheduling data-flow graphs via retiming and unfolding. *IEEE Trans. on Parallel and Distributed Systems*, 8:1259–1267, 1997.
- [5] A. Dasdan and R. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17:889–899, 1998.
- [6] G. DeMicheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.
- [7] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.
- [8] N. Passos, E. H.-M. Sha, and S. Bass. Loop pipelining for scheduling multi-dimensional systems via rotation. In *Proc. ACM/IEEE Design Automation Conf.*, pages 485–490, 1994.
- [9] M. Renfors and Y. Neuvo. The maximum sampling rate of digital filters under hardware speed. *Trans. on Circuits and Sampling*, CAS-28:196–202, 1981.