EXTENDED RETIMING: OPTIMAL SCHEDULING VIA A GRAPH-THEORETICAL APPROACH

Timothy W. O'Neil   *Sissades Tongsima*   *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). This graph is then used to perform *DFG scheduling*, where the starting times for executing the application's individual tasks are determined. The minimum length of time required to execute all tasks once is called the *schedule length* of the DFG. 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 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 and propose a new graph-transformation technique, *extended retiming*, which will. We will also present an algorithm for finding an extended retiming which transforms a DFG into one with minimal schedule length. Finally, we will demonstrate a constant-time algorithm which verifies the existence of a retimed DFG with the minimum schedule length.

1. INTRODUCTION

Many iterative or recursive applications can be represented by *data-flow graphs*, or DFGs [3, 8]. The nodes of a DFG represent tasks, while edges between nodes represent data dependencies among the tasks. In order to improve the performance of the system running these applications, one could apply graph-transformation techniques to the DFG in an attempt to increase the degree of parallelism. However, for certain graphs, these techniques may not be able to produce a transformed DFG with minimum schedule length. We will demonstrate this 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.

Static scheduling, the process of determining the starting time for each node of a DFG, is one of the most important stages in architectural synthesis. The execution of all tasks is referred to as an *iteration*, with the minimum length of time it takes to complete an iteration called the *schedule length* of the DFG. We seek to optimize our schedule by minimizing its length. If we assume that we have an adequate number of processors available to obtain the optimal schedule, there are two ways to accomplish this: we can explicitly schedule the DFG as-is, or we can first apply graph transformation techniques to the DFG and schedule the acyclic (*DAG*) part of the resulting graph.

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 [4, 7]. One of the more effective of these techniques is *retiming* [1, 5], 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 as-early-as-possible scheduling (also referred to in this paper as *DAG scheduling*), the length of the schedule for graph G is given by $cl(G)$.

To illustrate, consider the example of a DFG $G1$ given in Figure 1. We can see that the DFG of Figure 1 has $cl(G1) = 4$. A delay can be removed from edge (C,A) and placed on edge (A,B) to create a retimed version of $G1$, denoted $G1_r$ and pictured in Figure 2, with the property that $cl(G1_r) = 3$. The corresponding static schedule is found in Figure 3. 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. The first copy of A is not actually part of the repeating schedule; it's 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 [5] tells us that 3 is the minimum clock period we can achieve by retiming $G1$.

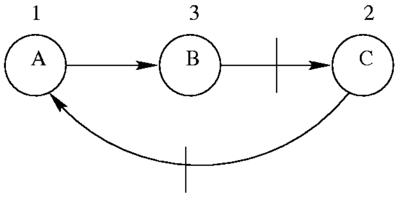

Figure 1: The simple data-flow graph G1

Chao and Sha [2] presented a unified algorithm which is referred to as *DFG scheduling* in this paper. We use this method to construct a schedule in Figure 3 for the DFG of Figure 1 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 $cy(G1) = 3$. Note that each task's starting time must be repeated every $cy(G1)$ time steps. Thus, in this case, the length of the schedule is given by $cy(G1)$. Furthermore, each task is implemented at the beginning of a time step and not between time

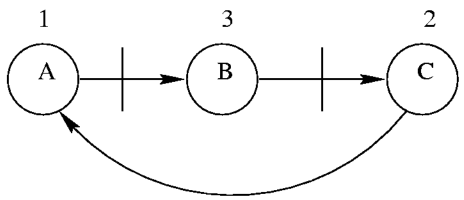

Figure 2: G1 retimed to have cl(G1$_r$)=3

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.

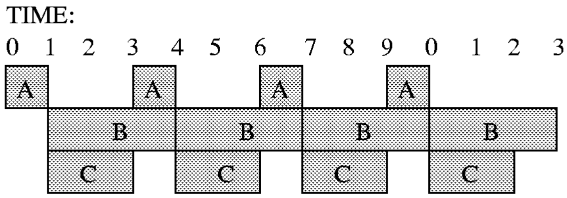

Figure 3: The schedule for G1 with cy(G1)=c1(G1$_r$)=3

We note that we have discovered a retiming and static schedule such that $cl(G1_r) = cy(G1)$ for the graph of Figure 1. 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. We want to deal with these questions:

1. What is the relationship between retiming and DFG scheduling?
2. Does the existence of a static schedule with $cy(G) = c$ imply the existence of a legal retiming such that $cl(G_r) = c$?
3. If (2) is not true, is there any way to map the cycle period of a legal static schedule into the graph model?
4. Is there an efficient graph transformation algorithm based on the properties from DFG scheduling which produces the same optimal result?

This paper answers these questions in the next section by demonstrating the following:

1. We show that 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 to be equivalent to DFG scheduling.
3. We demonstrate an algorithm based on DFGs for finding extended retimings.
4. We design a constant-time algorithm for verifying the existence of an extended retiming r which makes $cl(G_r) \leq c$ for a given integer c.

Finally, we conclude with a summary of our work and a list of further questions for exploration.

2. 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.

2.1. The Relationship Between Traditional Retiming and DFG Scheduling

We began by asserting that, given a DFG G and cycle period $cy(G)$, we could always find a retiming r such that $cl(G_r) = cy(G)$. Unfortunately, this doesn't happen. Consider the DFG $G2$ in Figure 4. 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 5 has cycle period 4.

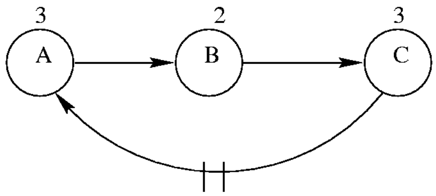

Figure 4: The sample DFG G2

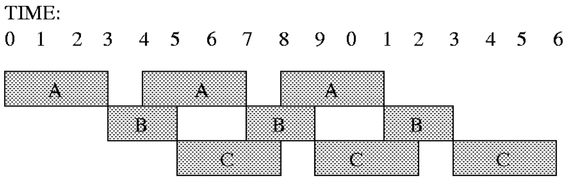

Figure 5: The schedule of G2 with cy(G2)=4

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 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 6. 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. We now will augment the traditional retiming technique by permitting the insertion of a delay between pieces of a split node. Our new concept will be called *extended retiming*.

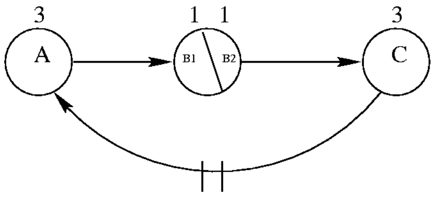

Figure 6: Graph G2 with node B split so that cl(G2$_r$)=4

2.2. The Concept of Extended Retiming

An *extended retiming* of a DFG $G = \langle V, E, d, t\rangle$ is a function *r:V→Q* such that $t(v) \cdot r(v) \in Z$ for all *v*∈*V*. From this definition, $r(v)$ can be viewed as consisting of an integer part and a fractional part. The integer part $\lfloor r(v) \rfloor$ 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 $\lceil r(v) \rceil - \lfloor r(v) \rfloor = 1$, a delay is left inside node *v* which splits it into two subnodes with computation times $t(v) \cdot (r(v) - \lfloor r(v) \rfloor)$ and $t(v) \cdot (\lceil r(v) \rceil - r(v))$. For example, an extended retiming with $r(A) = 1$, $r(B) = \frac{1}{2}$ and $r(C) = 0$ for the graph of Figure 4 achieves the retimed graph described for Figure 6.

As with standard retiming, we will denote the DFG retimed by *r* as $G_r = \langle V, E, d_r, t\rangle$, where $d_r(e) = d(e) + \lceil r(u) \rceil - \lfloor r(v) \rfloor$ is the retimed delay count of edge *e*∈*E*. An extended retiming is *legal* if $d_r(e) \geq 0$ for all edges *e*∈*E* and *normalized* if 0≤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)*. We can easily obtain these properties of D_r from the definition of d_r:

Lemma 2.1 *Let* G *be a DFG and* r *an extended retiming.*

1. *The retimed delay count on the path* $p : u \Rightarrow v$ *is* $D_r(p) = D(p) + \lceil r(u) \rceil - \lfloor r(v) \rfloor$.
2. *The retimed delay count on the cycle* $\ell \in G$ *is* $D_r(\ell) = D(\ell)$.

2.3. Scheduling Graphs and Iteration Bounds

Before proceeding to our main result, let us briefly review two concepts from the literature. Given a DFG *G*, we construct the *scheduling graph* $G^s = \langle V, E, w, t\rangle$ by reweighting each edge *e*=(u,v) according to the formula $w(e) = d(e) - \frac{t(u)}{c}$. Figure 7 shows the scheduling graph of our example when *c*=3.

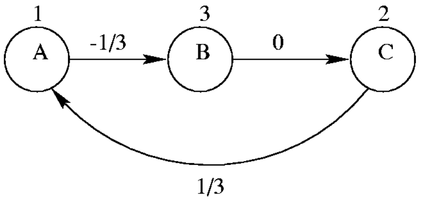

Figure 7: The scheduling graph of G1 with c=3

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 7, we note that $sh(A) = 0$ and $sh(B) = sh(C) =$-$\frac{1}{3}$.

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* [6] of *G*, which is denoted *B(G)* and defined to be the maximum time-to-delay ratio of all cycles in *G*. For example, the graph in Figure 1 contains only one loop, which has two delays and a total computation time of 6; thus $B(G) = 3$ for this graph. The schedule for this graph displayed in Figure 3 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.*

Since the iteration bound for the graph of Figure 1 is 3, and all nodes of this graph are 3 or smaller, Theorems 2.3 and 3.5 of [2] tell us that the minimum achievable cycle period for this graph is 3. We can then produce the static DFG schedule in Figure 3 by constructing the scheduling graph and then computing $sh(v)$ for each of the nodes. We can then use this information to create the schedule of Figure 3.

2.4. The Equivalence of Extended Retiming and DFG Scheduling

The equivalence of a traditional retiming and the absence of negative cycles in the scheduling graph was established for unit-time DFGs in [5]. We will now outline the proof (due to space limitations) of a similar relationship for *general-time* DFGs:

Theorem 2.1 *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:

By Theorem 11 of [5], $cl(G_r) \leq c$ implies the absence of negative weight cycles in the scheduling graph. On the other hand assume that G^s contains no negative-weight cycle. For every node *v* define

$$r(v) = \lfloor sh(v) \rfloor + \frac{c}{t(v)} \left(sh(v) - \lfloor sh(v) \rfloor \right). \qquad (1)$$

It can be proven that $\lfloor sh(v) \rfloor \leq r(v) < \lfloor sh(v) \rfloor + 1$ for all *v* under these circumstances. Furthermore, by the method of Theorem 11 of [5], we can show that *r* is a legal extended retiming of *G*. It remains to be shown that $cl(G_r) \leq c$.

Let $p : u \Rightarrow v$ be a path with $T(p) > c$. Let $s : v_0 \Rightarrow u$ and $q : v_0 \Rightarrow v$ be the paths in the modified G^s such that $sh(u) = W(s)$ and $sh(v) = W(q)$. Since $W(p) = D(p) - \frac{T(p)-t(v)}{c}$ by definition, we see that

$$D(p) + sh(u) - sh(v) \geq \frac{T(p) - t(v)}{c} > 0.$$

There are now three cases:

1. If $sh(u)$ and $sh(v)$ are both integers, then $r(u) = sh(u)$ and $r(v) = sh(v)$. In this case, $D_r(p) = D(p) + sh(u) - sh(v)$, which is integral and greater than or equal to a strictly positive fraction. We can thus conclude $D_r(p) \geq 1$ in this case.
2. If $sh(u)$ is not an integer, then

$$\begin{aligned} D_r(p) &= D(p) + \lceil r(u) \rceil - \lfloor r(v) \rfloor \\ &= D(p) + \lfloor sh(u) \rfloor - \lfloor sh(v) \rfloor + 1 \\ &\geq \lfloor D(p) + sh(u) - sh(v) \rfloor + 1 \\ &\geq \left\lfloor \frac{T(p)-t(v)}{c} \right\rfloor + 1 \geq 1. \end{aligned}$$

3. If $sh(u)$ is an integer but $sh(v)$ isn't, then

$$\lfloor sh(u) \rfloor - \lfloor sh(v) \rfloor > \lfloor sh(u) - sh(v) \rfloor$$

and we derive in this case

$$D_r(p) > \lfloor D(p) + sh(u) - sh(v) \rfloor \geq 0.$$

Hence $D_r(p)$ is an integer strictly greater than zero in this case.

In any case $D_r(p) \geq 1$, and by Lemma 4 of [5], $cl(G_r) \leq c$.

□

For example, when applied to the graph of Figure 4 with c=4, the retiming described in this theorem yields $r(A) = 0$, $r(B) =$-$\frac{1}{2}$ and $r(C) =$-1, precisely the retiming which produces the graph of Figure 6. Normalized, this becomes $r(A) = 1$, $r(B) = \frac{1}{2}$ and $r(C) = 0$. Let's now summarize what we've proven so far:

Theorem 2.2 *Let* G *be a DFG and* c *an integer. The following are equivalent:*

1. *There is a legal extended retiming* r *on* G *such that* $cl(G_r) \leq$ *c.*
2. *There exists a legal, integral, repeating, static schedule for* G *under the non-pipelined implementation with cycle period* c.
3. *The scheduling graph (for* G *that's associated with* c*)* G^s *contains no cycle having negative delay count and* $t(v) \leq c$ *for all nodes* v *of* G.
4. *The iteration bound* $B(G) \leq c$ *and* $t(v) \leq c$ *for all nodes* v *of* G.

Proof:

The equivalence of (1) and (3) is established in Theorem 2.1. The equivalence of (3) and (4) is proven in Lemma 3.1 of [2]. Finally, the equivalence of (4) and (2) is shown in Theorems 2.3 and 3.5 of the same paper.

□

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 smaller than our chosen *c* and can know almost immediately if there's an extended retiming which gives us $cl(G_r) \leq c$. Traditionally, this check has taken O($|V|\,|E|$) time. 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.

3. 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 also 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 done this without considering the effect of unfolding. Many good results have come from combining the traditional retiming and unfolding techniques when optimizing data-flow graphs [3]. It is our future work to study the combination of extended retiming and unfolding.

Throughout this paper, we have assumed the use of integral schedules under a non-pipelined implementation. We also have the possibilities of *fractional* schedules, where operations may be scheduled at any time (not necessarily at integral points), and pipelined implementations [2]. Combinations of these four parameters (integral or fractional DFG scheduling, pipelined or non-pipelined implementation) give us three additional models to explore as we discuss extended retiming.

4. ACKNOWLEDGEMENT

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

5. REFERENCES

[1] P.-Y. Calland, A. Darte, and Y. Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Transactions on Parallel and Distributed Systems*, 9:24–35, 1998.

[2] 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.

[3] L.-F. Chao and E. H.-M. Sha. Scheduling data-flow graphs via retiming and unfolding. *IEEE Transactions on Parallel and Distributed Systems*, 8:1259–1267, 1997.

[4] A. Darte, G.-A. Silber, and F. Vivien. Combining retiming and scheduling techniques for loop parallelization and loop tiling. *Parallel Processing Letters*, 7:379–392, 1997.

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

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

[7] C.-Y. Wang and K.K. Parhi. Resource-constrained loop list scheduler for DSP algorithms. *Journal of VLSI Signal Processing*, 11:75–96, 1995.

[8] A. Zaky and P. Sadayappan. Optimal static scheduling of sequential loops on multiprocessors. In *Proceedings of the International Conference on Parallel Processing*, pages III 130–137, 1992.