Optimizing Systems for Effective Block-Processing: The k-Delay Problem

Kumar N. Lalgudi Marios C. Papaefthymiou Miodrag Potkonjak

Department of Electrical Engineering
Yale University
New Haven, CT 06520

Department of Computer Science
University of California
Los Angeles, CA 90095

Abstract—Block-processing is a powerful and popular technique for increasing computation speed by simultaneously processing several samples of data. The effectiveness of block-processing is often reduced, however, due to suboptimal placement of delays in the dataflow graph of a computation.

In this paper we investigate an application of the retiming transformation for improving the effectiveness of block-processing in computation structures. Specifically, we consider the k-delay problem in which we wish to retime any given computation so that given an integer k the resulting computation can process k data samples simultaneously in a fully regular manner. Our main contribution is an $O(V^3E+V^4 \log V)$-time algorithm for the k-delay problem, where V is the number of computation blocks and E is the number of interconnections in the computation.

1 Introduction

In many application domains, computations are defined on semi-infinite or very long streams of data. The rate of the incoming data is dictated by the nature of the application and often cannot be satisfied by a straightforward implementation of a systems' specification. In order to meet the computational demands of several applications, multiple samples of the incoming data stream must be processed simultaneously. This approach, known as *block-processing* or *vectorization*, is widely used to satisfy throughput requirements through the use of parallelism and pipelining. Block-processing enhances both regularity and locality in computations, thus facilitating their efficient hardware implementation [1, 4]. Enhanced regularity reduces the effort in software switching and address calculation, and improved locality improves the effectiveness of code-size reduction methods [7]. Moreover, block-processing enables the efficient utilization of pipelines and efficient implementations of vector-based algorithms such as FFT-based filtering and error-correction codes. In general, block-processing is beneficial in all cases where the net cost of processing n samples individually is higher than the net cost of processing n samples simultaneously.

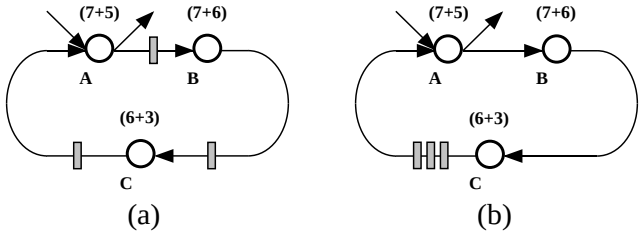

Figure 1: Improving the effectiveness of block-processing by retiming. The block-processing factor of the original computation dataflow graph in Part (a) is 1. The block-processing factor of the retimed graph in Part (b) is 3.

There are several ways to increase the *block-processing factor* of a computation, that is, the number of data samples that can be processed simultaneously. For example, one can unfold the basic iteration of a computation and schedule computational blocks from different iterations to execute successively. This technique, however, may not uniformly increase the block-processing factor for all computational blocks.

Another transformation that can increase the block-processing factor is *retiming*. Unlike other optimization techniques that have targeted high-level synthesis [3, 9], retiming has been used traditionally for clock period minimization [2, 5, 6]. Figure 1 illustrates the use of retiming to improve block-processing. The computation dataflow graph (CDFG) in this figure has three computation blocks A, B, and C and three delays. An input stream is coming into block A, and an output stream is generated by A. Assuming that the computation is implemented by a uniprocessor system, the pair $(x+y)$ above each block gives the initiation time x and the computation time y per block input. The initiation time includes context-switching overhead for fetching data and instructions from the background memory and the cost for reconfiguring pipelines. A single iteration of the computation in Figure 1(a) completes in (7+5)+(7+6)+(6+3)=34 cycles by executing the blocks in the order A_1, B_1, C_1. For three iterations, the computational blocks can be executed in the order $A_1, B_1, C_1, A_2, B_2, C_2, A_3, B_3, C_3$. In this case, a new input is consumed every 34 cycles, and the entire computation needs $3 \times 34 = 102$ cycles. The functionally equivalent CDFG in Figure 1(b) that has been obtained by retiming the original CDFG can complete all three iterations in a single *block iteration* that requires only $(7+5+5+5)+(7+6+6+6)+(6+3+3+3) = 62$ cycles, however. By grouping all three delays on one edge, the computations of the three iterations can be executed in the order $A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3$, thus amortizing the initiation time of *each* block over three inputs.

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

Recently, retiming has been studied in the context of optimum vectorization for a class of DSP programs [8, 10]. Specifically, a technique for linear vectorization of DSP programs using retiming has been presented in [10]. This technique involves the redistribution of delays in the CDFG representation of a DSP program in a way that results in maximum concentration of delays on the edges. Fully regular vectorization, however, cannot be achieved using the linear vectorization approach in that paper. Moreover, the non-linear integer programming formulation of the retiming problem for computing linear vectorizations presented in that paper can be computationally very expensive.

In this paper, we consider the problem of retiming computation dataflow graphs to achieve any given block-processing factor k. We call this the k-delay problem. We first present a straightforward *integer linear programming* (ILP) formulation of the k-delay problem. We then give an $O(V^3E + V^4 \lg V)$-time algorithm for the k-delay problem, where V is the number of computation blocks and E is the number of interconnections in the CDFG. This is the first polynomial-time algorithm ever presented for the k-delay problem. Given a CDFG and a positive integer k, our algorithm computes a retimed CDFG that achieves a block-processing factor of k or determines that such a retiming does not exist. An important feature of our approach is that *all* blocks in the retimed CDFG achieve the same block-processing factor k and the same execution order across iterations. As a result, our retimed CDFGs can operate faster and be less expensive to implement than generic block-processed CDFGs.

The remainder of this paper is organized as follows. In Section 2 we describe the representation of computations as dataflow graphs, and we give background material on block-processing and retiming. We also give a precise mathematical formulation of the k-delay problem. In Section 3, we present an integer linear programming formulation of the k-delay problem. In Section 4, we describe an asymptotically more efficient algorithm for the k-delay problem whose running time is polynomial time in the size of the CDFG. We conclude in Section 5 with an empirical evaluation of our proposed optimization technique.

2 Preliminaries

In this section, we give background material on computation dataflow graphs, block-processing, and retiming. We also give a mathematical formulation of the k-delay problem.

2.1 Computation dataflow graphs

A computation dataflow graph is an edge-weighted directed graph $G = \langle V, E, w \rangle$. The vertices $v \in V$ model the computation blocks of a computation (subroutines, arithmetic or boolean operations), and the directed edges $e \in E$ model interconnections (data and control dependencies) between the computation blocks. Each edge $e \in E$ is associated with a weight $w(e)$ that denotes the number of delays associated with that interconnection. A CDFG is *well-formed* if for every edge $e \in E$, we have $w(e) \geq 0$, and every directed cycle contains at least one delay.

2.2 Block-processing

Block-processing strives to maximize the throughput of a computation by simultaneously processing multiple samples of the incoming data. The maximum number of samples that can be processed simultaneously or immediately after each other by a block v is called the block-processing factor k_v of that block. A block-processing is *linear* if all blocks have the same block-processing factor k. Given a linear block-processing with factor k, the $k \cdot |V|$ computational block evaluations that generate k iterations of the computation constitute a block iteration. A linear block-processing with factor k is *regular* if the k data samples processed simultaneously by every computational block are accessed during the same block iteration. The retimed CDFG in Figure 1(b), for example, can be block-processed linearly and regularly with a block-processing factor of 3.

The following lemma gives necessary and sufficient conditions for achieving effective block-processing.

Lemma 1 *Let $G = \langle V, E, w \rangle$ be CDFG. We can achieve a linear and regular block-processing of G with factor k if and only if for every edge $e \in E$, we have*

$$w(e) = 0 \quad or \quad w(e) \geq k \ . \tag{1}$$

Proof. ($\Rightarrow$) If Relation (1) is not satisfied for some CDFG that can be block-processed linearly and regularly with a factor of k, then there exists an edge $u \xrightarrow{e} v$ such that $1 \leq w(e) \leq k-1$. Vertex v can process at most $w(e)$ samples per iteration. Since $w(e) < k$, the remaining $k - w(e)$ samples must be accessed from the previous block iteration, which contradicts regularity.

($\Leftarrow$) Straightforward. □

2.3 Retiming

A retiming of a CDFG $G = \langle V, E, w \rangle$ is an integer valued vertex-labeling $r : V \rightarrow \mathbf{Z}$. This integer value denotes the assignment of a lag to each vertex which transforms G into $G_r = \langle V, E, w_r \rangle$ where for each edge $u \xrightarrow{e} v$ in G, w_r is defined by the equation

$$w_r(e) = w(e) + r(v) - r(u) \ . \tag{2}$$

In order for the retimed CDFG G_r to be well-formed, the retiming r must satisfy the constraint $w_r(e) \geq 0$ for all edges $e \in E$.

An important characteristic of the graph G in the context of retiming that is defined for each pair of vertices u and v is the parameter

$$W(u, v) = \min \left\{ w(p) : u \overset{p}{\rightsquigarrow} v \right\} \ ,$$

where $w(p) = \sum_{e \in p} w(e)$ denotes the delay count of a path p.

Another useful parameter defined for a CDFG is

$$F = \max_{u \xrightarrow{e} v \in E} \{ w(e) + W(v, u) \} \ , \tag{3}$$

which gives an upper bound on the number of delays that can be placed on any edge. It can be shown that for any any retiming r, we have

$$w_r(e) \leq F \quad \text{for every edge } e \in E \ . \tag{4}$$

2.4 The k-delay problem

According to Lemma 1, a linear and regular block-processing with factor k can be achieved only for CDFGs that have either 0 or at least k delays on each edge. If a given CDFG does not satisfy Relation (1), we can redistribute its delays by retiming. We call the problem of computing such a retiming the *k-delay problem*:

Problem KDP *(The k-delay problem) Given a CDFG $G = \langle V, E, w \rangle$ and a positive integer k, compute a retiming function $r : V \rightarrow \mathbf{Z}$ such that for every edge $u \stackrel{e}{\rightarrow} v$ in E, we have*

$$w_r(e) = 0 \quad or \quad w_r(e) \geq k \ , \tag{5}$$

or determine that no such retiming exists.

3 ILP formulation

Problem KDP cannot be expressed directly in a linear programming form because of the disjunction in Relation (5). In this section we rely on the notion of the *companion graph* that was described in [5] to express Problem KDP as an Integer Linear Program (ILP).

The companion graph G' of a CDFG G is constructed by segmenting every edge $u \stackrel{e}{\rightarrow} v \in E$ into two edges $u \stackrel{e_1}{\rightarrow} x_{uv}$ and $x_{uv} \stackrel{e_2}{\rightarrow} v$, where x_{uv} is a dummy vertex. Thus, the companion graph $G' = \langle V', E', w' \rangle$ is defined as

$$\begin{aligned} V' &= V \cup \left\{ x_{uv} : u \stackrel{e}{\rightarrow} v \in E \right\} \ , \\ E' &= \left\{ u \stackrel{e_1}{\rightarrow} x_{uv}, x_{uv} \stackrel{e_2}{\rightarrow} v : u \stackrel{e}{\rightarrow} v \in E \right\} \ , \end{aligned}$$

and for each edge $u \stackrel{e}{\rightarrow} v \in E$, we have

$$\begin{aligned} w'(e_1) &= \min\{1, w(e)\} \ , \text{ and} \\ w'(e_2) &= w(e) - \min\{1, w(e)\} \ . \end{aligned}$$

The following lemma gives necessary and sufficient conditions for any retiming that solves Problem KDP.

Lemma 2 *Let $G = \langle V, E, w \rangle$ be a CDFG, and let $G' = \langle V', E', w' \rangle$ be its companion graph. Then there exists a retiming function $r : V \rightarrow \mathbf{Z}$ that solves Problem KDP on G if and only if there exists a retiming function $r' : V' \rightarrow \mathbf{Z}$ such that for every edge $u \stackrel{e}{\rightarrow} v \in E'$, we have*

$$r'(v) + w'(e) - r'(u) \geq 0 \ , \tag{6}$$

and for every edge $u \stackrel{e}{\rightarrow} v$ in E, we have

$$w'_{r'}(e_1) \leq 1 \ , \tag{7}$$

$$w'_{r'}(e_2) \leq F \cdot w'_{r'}(e_1) \ , \ and \tag{8}$$

$$w'_{r'}(e_2) \geq (k-1) \cdot w'_{r'}(e_1) \ . \tag{9}$$

Proof. Inequality (6) ensures that the retimed CDFG is well-formed. Inequalities (7) and (8) ensure that the delay counts in $G'_{r'}$ satisfy the definition of a companion graph. Inequality (9) ensures that for every edge $u \stackrel{e}{\rightarrow} v$ in E, if edge $u \stackrel{e_1}{\rightarrow} x_{uv}$ has a delay after retiming, then edge $x_{uv} \stackrel{e_2}{\rightarrow} v$ has at least $k-1$ delays. In other words, we ensure that if edge $u \stackrel{e}{\rightarrow} v$ in E has any delays, then it has at least k delays after retiming. By construction, a solution r for Problem KDP on G can be derived from r' by simply setting $r(u) = r'(u)$ for all $u \in V$. □

Theorem 3 *Let $G = \langle V, E, w \rangle$ be a computation flow graph and let $G' = \langle V', E', w' \rangle$ be its companion graph. Then Problem KDP on G can be expressed as an integer linear program on the companion graph G'.*

Proof. Follows directly from the linearity of Relation (2) and the form of the inequalities in Lemma 2. □

4 Polynomial-time algorithm

The constraints in the ILP formulation of Problem KDP do not appear to have any special structure. We thus need to resort to general integer linear programming solvers to compute a solution. In this section we show that Problem KDP can be solved efficiently by giving a polynomial-time algorithm for it. We first give a set of necessary conditions for the feasibility of Problem KDP on a given CDFG. Subsequently, we describe the construction of a transformed graph G^T, and we give a set of necessary and sufficient conditions for the feasibility of Problem KDP on G^T. We then present a polynomial-time algorithm that uses G^T to solve Problem KDP.

4.1 Necessary conditions

The following lemma gives necessary conditions for the feasibility of Problem KDP.

Lemma 4 *Let $G = \langle V, E, w \rangle$ be a CDFG. Problem KDP is feasible on G only if the following conditions hold.*

C1: For every vertex pair $u, v \in V$, we have $W(u,v) + W(v,u) \geq k$.

C2: For every vertex pair $u, v \in V$ such that there exists a path $u \stackrel{p}{\rightsquigarrow} v$ in G with delay count $w(p)$ and $W(u,v) < w(p) < k$, there exists a retiming $r : V \rightarrow \mathbf{Z}$ such that in the retimed graph G_r, we have $W_r(u,v) \geq k$.

Proof. Omitted. □

The challenge in solving Problem KDP is to determine which edges should have nonzero delay count. In the ILP formulation, we determine these edges explicitly. The asymptotically efficient technique we describe in this section determines these edges implicitly. It first identifies vertex pairs $u, v \in V$ such that for every path $u \stackrel{p}{\rightsquigarrow} v \in G_r$, we have $w_r(p) > 0$. It subsequently obtains a solution to Problem KDP by greedily redistributing the delays of each path.

The main idea of our technique is to identify vertex pairs $u, v \in V$ such that the shortest path $u \stackrel{q}{\rightsquigarrow} v$ in *every* retimed CDFG that satisfies Relation (5) must necessarily contain delays. We call these vertex pairs *delay essential* (DE). For example, every vertex pair $u, v \in V$ for which there exists a path $u \stackrel{p}{\rightsquigarrow} v$ such that $W(u,v) < w(p) < k$ is delay essential, because $W_r(u,v) = 0$ implies that $w_r(p) = w(p) - W(u,v) < k$

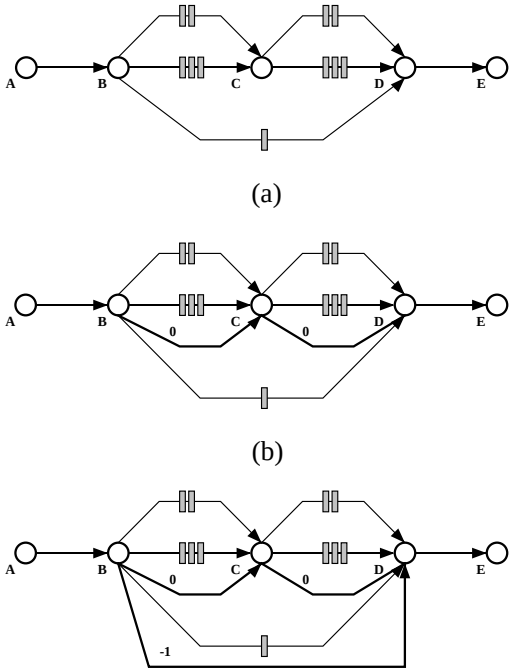

Figure 2: Illustration of explicit and implicit delay essential (DE) vertex pairs. The CDFG in Part (a) has been transformed by Algorithm AddEdges to generate the CDFG in Part (b) and then finally the CDFG in Part (c). The bold edges in Part (b) are between explicit DE pairs. The weights on these edges indicate the excess delay associated with the corresponding vertex pairs. The bold edges in Part (c) denote both explicit and implicit DE pairs. For example, the pair B, D is implicitly DE and becomes apparent only after the DE vertex pairs B, C and C, D are made explicit.

which violates Relation (5). Condition C2 must hold for every such pair, that is, the delay count of the shortest path $u \overset{p}{\rightsquigarrow} v$ in G_r must be at least k, otherwise Relation (5) will be violated for some edge along this path.

The following lemma casts the necessary conditions of Lemma 4 as a retiming problem on an appropriately constructed constraints graph.

Lemma 5 *Let $G = \langle V, E, w \rangle$ be a given CDFG and let $G^* = \langle V, E^*, w^* \rangle$ be the constraints graph that is constructed from G as follows: For every vertex pair $u, v \in V$ such that there exists a path $u \overset{p}{\rightsquigarrow} v$ with delay count $w(p)$ and $W(u,v) < w(p) < k$, add a new edge $u \overset{e}{\rightarrow} v$ with edge weight $w^*(e) = W(u,v) - k$. Then Problem KDP is feasible for G only if there exists a retiming $r : V \rightarrow \mathbf{Z}$ such that for every edge $u \overset{e}{\rightarrow} v \in E^*$ in the retimed graph G_r^* we have*

$$r(v) - r(u) + w^*(e) \geq 0 \ . \tag{10}$$

Proof. Follows from the construction of G^* and Condition C2 in Lemma 4. □

Observe that Condition C2 is necessary but not sufficient, since it captures only *explicit* delay requirements and not *implicit* or hidden requirements. Let us assume, for example, that we wish to solve Problem KDP for the CDFG in Figure 2(a) with $k = 2$. Since the shortest and the second shortest paths between vertices B and D have 1 and 4 delays, respectively, Condition C2 does not apply, and the pair B, D does not appear to be delay essential. We can verify, however, that the shortest path between B and D must necessarily contain delays in any solution of Problem KDP, since it is impossible to retime the given CDFG and zero out the delay count of $B \rightarrow D$. Since the vertex pairs B, C and C, D must satisfy Condition C2, they need at least 2 delays on their shortest paths $B \rightarrow C$ and $C \rightarrow D$. Thus, no delay along the path $B \rightsquigarrow C \rightsquigarrow D$ can be moved outside $B \rightsquigarrow D$. Since retiming changes the delay along paths between the same vertex pair in an identical manner, the delay on edge $B \rightarrow D$ cannot be moved out of $B \rightsquigarrow D$.

In order to expose implicit delay requirements, we construct a new graph $G^T = \langle V, E^T, w^T \rangle$ from G such that all delay essential vertex pairs become explicit. The graph G^T is constructed from G by introducing an edge $u \overset{i}{\rightarrow} v$ with weight $w^T(i) = W(u,v) - k$ between every delay essential vertex pair $u, v \in V$. Intuitively, $W(u,v) - k$ equals the *excess delay* of the pair $u, v \in V$ and gives an upper bound on the number of delays that can be "contributed" by that pair to the rest of the graph. As new edges are introduced, new vertex pairs can become delay essential, as shown in Figure 2. For example, the pair B, D becomes delay essential only after the delay requirements of the pairs B, C and C, D become explicit.

Algorithm AddEdges in Figure 3 transforms the given graph G into G^T. The algorithm operates in two steps. In the first step, explicit delay essential pairs are determined by comparing the delay counts on the shortest and second shortest paths for every vertex pair and checking for Condition C2. (It can be shown that it is sufficient to compare the delay count of the two shortest paths between a vertex pair $u, v \in V$ to determine if it is delay essential.) For every delay essential vertex pair u, v, an edge $u \overset{e}{\rightarrow} v$ with weight $w(e) = W(u,v) - k$ is introduced to ensure that $W_r(u,v) \geq k$. In the second step, implicit delay essential pairs are determined by comparing for every vertex pair the delay counts of the shortest path in the transformed graph of the current iteration and the shortest path in the original graph. If for a vertex pair (u, v) the delay counts of these two paths differ, then an edge $u \overset{e}{\rightarrow} v$ with weight $w(e) = W(u,v) - k$ is introduced to ensure that $W_r(u,v) \geq k$. The process is repeated until every delay essential vertex pair is made explicit by these additional edges. (It can be shown that is is sufficient to place the additional edge only if the delay counts for the two shortest paths differ by less than k. If their difference exceeds k, then the condition $W_r(u,v) \geq k$ is taken care of automatically.)

The following lemma expresses the necessary conditions for the feasibility of Problem KDP on a given G in terms of the transformed graph G^T.

Lemma 6 *Let $G = \langle V, E, w \rangle$ be a CDFG, and let $G^T = \langle V, E^T, w^T \rangle$ be the transformed graph generated by Algorithm* AddEdges. *Let $r : V \rightarrow \mathbf{Z}$ be a solution for Problem KDP on G. Then for every edge $u \overset{e}{\rightarrow} v \in E^T$, r satisfies*

$$r(v) - r(u) + w^T(e) \geq 0 \ . \tag{11}$$

Proof. Omitted. □

Lemma 7 *Algorithm* AddEdges *transforms a given CDFG $G = \langle V, E, w \rangle$ into G^T or determines that such a transformation is not possible in $O(V^3 E + V^4 \lg V)$ steps.*

```
ADDEDGES(G, k)
1  for every vertex pair u, v ∈ V
2       do m[u][v] ← FALSE
3  Q ← ∅
4  Run an all-pairs 2-shortest paths algorithm on G.
5  for every vertex pair u, v ∈ V with u ⇝(p1) v and u ⇝(p2) v
         being the two shortest paths between u, v
6       do if w(p2) − w(p1) < k
7             then Q ← Q ∪ {(u, v)}
8  repeat
9       for every delay essential vertex pair u, v ∈ Q
10           do m[u][v] ← TRUE
11              Introduce u →(e) v with w^T(e) = W(u, v) − k
12              E^T ← E^T ∪ {u →(e) v}
13      G^T ← ⟨V, E^T, w^T⟩
14      Q ← ∅
15      Run an all-pairs shortest paths algorithm on G^T
16      for every pair u, v ∈ V with W(u, v) > W^T(u, v)
17           do if m[u][v] = FALSE
18                 then Q ← Q ∪ {(u, v)}
19 until Q = ∅
20 return G^T
```

Figure 3: Algorithm ADDEDGES transforms $G = \langle V, E, w\rangle$ into G^T in $O(V^3E + V^4 \lg V)$ steps. In the new graph G^T, all delay essential vertex pairs of G are explicit.

Proof. The **repeat** loop in Step 8 executes $O(V^2)$ times, since each iteration adds at least one edge. The loop body runs in $O(VE + V^2 \lg V)$ time. □

4.2 Sufficient conditions

In this section we show that the necessary conditions presented in Subsection 4.1 are also sufficient for Problem KDP. The following lemma proves an important property of G_r^T which is used in Lemma 9 to prove the sufficiency of these conditions. Intuitively, this lemma shows that if an edge $u \xrightarrow{e} v$ has positive but fewer than k delays in a graph G_r that satisfies the conditions of Lemma 6, then its delay count is the minimum delay count for the vertex pair $u, v \in V$. Consequently, we can zero out the delay count of such an edge by retiming.

Lemma 8 *Let $G = \langle V, E, w\rangle$ be a CDFG, and let $G^T = \langle V, E^T, w^T\rangle$ be the transformed graph generated by Algorithm* ADDEDGES. *Let $r : V \to \mathbf{Z}$ be a retiming such that for every edge $e \in E^T$, we have*

$$r(v) - r(u) + w^T(e) \geq 0 \ . \tag{12}$$

Then for every deficient edge $u \xrightarrow{e} v$ in E, such that $0 < w_r^T(e) < k$, we have

$$w_r^T(e) = W_r^T(u, v) \ . \tag{13}$$

Proof. Omitted. □

A *supervertex* $\mathcal{U}$ of a vertex u is the set of vertices x such that

$$\mathcal{U} = \{x \in V : \ x \overset{p}{\rightsquigarrow} u \text{ in } G \text{ with } w(p) = 0 \quad \text{or} \quad u \overset{q}{\rightsquigarrow} x \text{ in } G \text{ with } w(q) = 0\} \ . \tag{14}$$

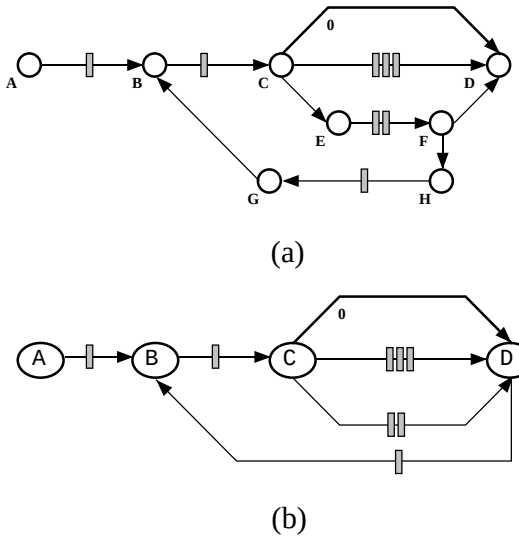

Figure 4: Generating the supergraph of a transformed CDFG. The supervertices $\mathcal{A} = \{A\}$, $\mathcal{B} = \{B, G\}$, $\mathcal{C} = \{C, E\}$, and $\mathcal{D} = \{D, F, H\}$ of the CDFG in Part (b) are obtained by collapsing vertices connected by zero-weight edges of the CDFG in Part (a). Vertices connected by bold edges are not collapsed to form supervertices even though the bold edges may have zero weight.

We can transform any given G into its supergraph $\mathbf{sup}(G)$ by collapsing all vertices connected by zero-weight edges into a single supervertex. This transformation is illustrated in Figure 4. Note that when we compute the supergraph of a transformed graph $\mathbf{sup}(G^T)$, we only collapse those vertices connected by zero-weight edges that belong to the original graph G. In Figure 4, for example, even though the additional edge $\mathcal{C} \to \mathcal{D}$ has no delays, the vertices $\mathcal{C}$ and $\mathcal{D}$ are not collapsed.

In the following lemma, we show that a retiming that minimizes the number of *edges with delays* in E subject to Inequality (11) is a solution to Problem KDP. We prove this result by contradiction. We consider a retiming that satisfies Inequality (11) and minimizes the number of *edges $\in E$ with delays* without solving Problem KDP. We then show how to to further minimize the number of edges with delays in E, thus contradicts our assumption that our initial retiming was optimal.

Lemma 9 *Let $G = \langle V, E, w\rangle$ be a given CDFG and let $G^T = \langle V, E^T, w^T\rangle$ be its transformed graph generated by Algorithm* ADDEDGES. *Let $M = \{e \in E : w(e) \geq 1\}$ denote the set of edges in G with delays on them. If r is a retiming that minimizes $|M|$ under the constraint*

$$r(v) - r(u) + w^T(e) \geq 0 \quad \text{for every } u \xrightarrow{e} v \in E^T \ , \tag{15}$$

then r is a solution to Problem KDP.

Proof. Let r minimize the number of edges with delays under Inequality (15). If r does not solve Problem KDP, then there exist edges $i \in E$ in G_r^T such that $1 \leq w_r^T(i) \leq k - 1$. Let $u \xrightarrow{j} v \in E$ be such an edge with $w_r^T(j) = m$, where $1 \leq m \leq k - 1$. Let $\mathbf{sup}(G_r^T)$ be the supergraph of G_r^T, and let $\mathcal{U}$ and $\mathcal{V}$ be the supervertices of u and v, respectively.

We now compute a new retiming r' that maintains the necessary conditions for the feasibility of Problem KDP

on G. Consider an assignment $r'(\mathcal{X}) = m - W_r^T(\mathcal{U}, \mathcal{X})$ to all supervertices $\mathcal{X}$ in $\mathbf{sup}(G_r^T)$. By the definition of shortest paths, for every supervertex pair $(\mathcal{X}_1, \mathcal{X}_2)$ in $\mathbf{sup}(G_r^T)$ we have

$$W_r^T(\mathcal{U}, \mathcal{X}_2) \leq W_r^T(\mathcal{U}, \mathcal{X}_1) + W_r^T(\mathcal{X}_1, \mathcal{X}_2) \ ,$$

which can be rewritten as

$$r'(\mathcal{X}_2) - r'(\mathcal{X}_1) + W_r^T(\mathcal{X}_1, \mathcal{X}_2) \geq 0 \ .$$

Consequently, the supergraph $\mathbf{sup}(G_r^T)_{r'}$ obtained by setting $r(u)+r'(u)$ equal to the retiming of $u \in V$ satisfies Inequality (15).

For the new retiming r', the edge $u \xrightarrow{j} v$ contains no more delays. Note that we have, $r'(\mathcal{U}) = m - W_r^T(\mathcal{U},\mathcal{U}) = m - 0 = m$. Since $0 < w_r^T(u \xrightarrow{j} v) = m < k$, we infer from Lemma 8 that $W_r^T(\mathcal{U}, \mathcal{V}) = W_r^T(u, v) = w_r^T(j) = m$, which implies that $r'(\mathcal{V}) = m - W_r^T(\mathcal{U}, \mathcal{V}) = m - m = 0$. Since $r'(\mathcal{U}) = m$ and $r'(\mathcal{V}) = 0$, it follows that $u \xrightarrow{j} v$ has a zero delay count after the retiming r'.

We now show that the number of edge with delays can be reduced further. By construction, a supergraph has no zero-weight edges. Therefore, retiming supervertices ensures that no delays are introduced on zero-weight edges. Since all delays are removed from at least one edge, namely $u \xrightarrow{j} v \in E$, the resulting supergraph $\mathbf{sup}(G_r^T)_{r'}$ has a smaller number of edges in E with nonzero delays than G_r^T, thus contradicting our assumption that r minimizes the number of edges with delays in E. We conclude from this contradiction that r is a solution to Problem KDP. □

4.3 The algorithm

The proof of Lemma 9 captures our two-step strategy for solving Problem KDP. The first step generates the transformed G^T and computes a retiming that satisfies Inequality (11), thus ensuring that all paths in G_r have enough delays. The second step places delays onto individual edges using a greedy procedure that computes an incremental retiming for every edge with fewer than k delays. In each iteration, the incremental retiming removes all the delays on the deficient edge and redistributes them on other edges *with delays* without violating any of the necessary conditions. This process is repeated until there are no more deficient edges left and all edges satisfy Relation (5). The final retiming is the sum of the retiming in the first step and of all the incremental retimings in the second step. Algorithm SolveKDP described in theFigure 5 implements this greedy procedure.

Theorem 10 *Algorithm* SolveKDP *computes a retiming that correctly solves Problem KDP, or determines that the problem is infeasible in* $O(V^3E + V^4 \lg V)$ *steps.*

```
SolveKDP (G, k)
1   G^T ← AddEdges (G, k)
2   Compute r that satisfies (11) for every edge in G^T
3   if no feasible r exists
4       then return INFEASIBLE
5   while ∃ u --j--> v ∈ E of G_r^T with 0 < w_r^T(j) < k
6       do Construct sup(G_r^T)
7          Compute W_r^T(U, X) using single-source
               shortest-paths from U to all X ∈ sup(G_r^T)
8          for every X ∈ sup(G_r^T)
9              do r'(X) ← w_r^T(j) − W_r^T(U, X)
10         r ← r + r'
11  return r
```

Figure 5: Algorithm SolveKDP for solving Problem KDP.

5 Conclusion

It is straightforward to calculate the potential benefit of block-processing and k-delay retiming. If we denote the sum of the all initiation times by O and the sum of all computation times by C, the total reduction in execution time with regular block-processing is given by the factor $k \cdot (O + C)/(O + k \cdot C)$. We applied our techniques to an adaptive voice echo canceler, an adaptive video coder, and two examples from [10]. Our results indicate that it is possible to achieve performance improvements between 30% and 45% using our k-delay optimization.

References

[1] L. M. Guerra, M. Potkonjak, and J. Rabaey. System-level design guidance using algorithm properties. In *Proc. of the VLSI Signal Processing Workshop*, pages 62–73, October 1994.

[2] A. T. Ishii, C. E. Leiserson, and M. C. Papaefthymiou. Optimizing two-phase, level-clocked circuitry. In *Advanced Research in VLSI and Parallel Systems: Proc. of the 1992 Brown/MIT Conference*. MIT Press, March 1992.

[3] D. C. Ku and G. De Micheli. Relative scheduling under timing constraints: Algorithms for high-level synthesis of digital circuits. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 11(6):696–717, 1992.

[4] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, 1988.

[5] K. N. Lalgudi and M. C. Papaefthymiou. DelaY: An efficient tool for retiming with realistic delay modeling. In *Proceedings of the 32th ACM/IEEE Design Automation Conference*, pages 304–309, June 1995.

[6] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1), 1991.

[7] S. Liao, S. Devadas, K. Keutzer, S. Tjiang, and A. Wang. Storage assignment to decrease code size. In *Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 186–195, 1995.

[8] S. Ritz, M. Pankert, V. Zivojnovic, and H. Meyr. Optimum vectorization of scalable synchronous dataflow graphs. In *Proc. of the International Conference on Application-Specific Array Processors*, pages 285–296, October 1993.

[9] R. A. Walker and D. E. Thomas. Behavioral transformation for algorithmic level ic design. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 8(10):1115–1127, 1989.

[10] V. Zivojnovic, S. Ritz, and H. Meyr. Retiming of DSP programs for optimum vectorization. In *Proc. of the International Conference on Acoustic, Speech, and Signal Processing*, 1994.