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

Kumar N. Lalgudi
Marios C. Papaefthymiou

Department of Electrical Engineering
Yale University
New Haven, CT 06520

Miodrag Potkonjak

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

February 12, 1996

Abstract

Block-processing is a powerful and popular technique for increasing computation speed by simultaneously processing several samples of data. In comparison with conventional computations, block-processed computations have reduced hardware and power dissipation requirements and result in less expensive implementations on parallel and uniprocessor programmable platforms. 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 a new 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 restructure any given computation by relocating its delays so that for any given k, the resulting computation can process k data samples simultaneously in a fully regular manner. We formulate the k-delay problem as an integer linear program. We also give an efficient algorithm for the k-delay problem, whose running time is polynomial in the size of the computation dataflow graph. Specifically, for a given CDFG and a given positive integer k, our algorithm computes a retimed CDFG that can process k samples simultaneously, or determines that such a retiming is infeasible, in $O(V^3E + V^4 \lg V)$ time where V is the number of computation blocks and E is the number of interconnections in the CDFG. Our algorithm relies on a greedy procedure that operates on a constraints graph that is generated through a novel graph transformation.

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 the specification. Although the speed of hardware components has been increasing steadily, the throughput requirements of new applications have been increasing at an even faster pace. Recent studies show that while computational requirements per sample of state-of-the-art communication has been doubling every year, the processing power of hardware is doubling only every three years [3]. Furthermore, new applications requirements such as low power dissipation impose

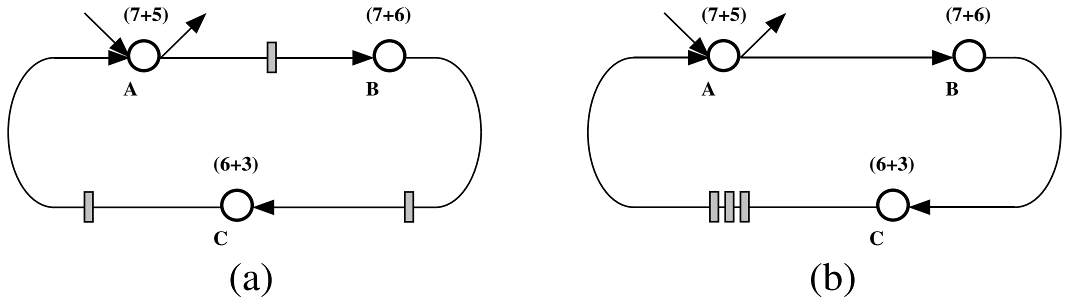


Figure 1: Improving block-processing by retiming. The block-processing factor of the original computation dataflow graph in Part (a) is 1. The block-processing factor of the retimed graph in Part (b) is 3.

additional design constraints which often further add to the gap between the speed of hardware primitives and the rate of incoming data.

In order to meet the increasing computational demands of today's communication applications, it is required to compute simultaneously on multiple samples of the incoming data stream. 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 which greatly facilitates efficient implementation on many hardware platforms [4, 7]. Enhanced regularity reduces the effort in software switching and address calculation, and improved locality improves the effectiveness of code-size reduction methods [10]. 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 [1]. 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. The typical cost measures include processing time, memory requirements, and energy dissipation per sample.

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 in a regular manner. For example, one can unfold the basic iteration and subsequently schedule computations so that computational blocks from different iterations are executed successively. This technique does not always result in full block-processing, however. Another transformation that can be used for increasing the block-processing factor is *retiming*. Contrary to other transformation techniques that have targeted high-level synthesis [6, 14], retiming has been used traditionally for clock period minimization [5, 8, 9] and for logic synthesis [11, 12]. The example in Figure 1 illustrates how retiming can be used to improve block-processing. The computation dataflow graph (CDFG) in this figure has three computation blocks A, B, and C and three delays. The pair $(x + y)$ above each block gives the initiation time x and the computation time y per data sample. The initiation time includes context-switching time such as the number of cycles required to bring from the background memory all data needed to start an iteration and the time required to reconfigure pipelines. The computation shown in Figure 1(a) needs (7+5)+(7+6)+(6+3)=34 cycles for a single iteration and $3 \times 34 = 102$ cycles for three iterations. The execution order for three iterations of the computation is $\langle A_1, B_1, C_1\rangle$, $\langle A_2, B_2, C_2\rangle$ and $\langle A_3, B_3, C_3\rangle$. The retimed CDFG in Figure 1(b) can complete all three iterations in only $(7+5+5+5)+(7+6+6+6)+(6+3+3+3) = 48$ cycles. The retimed CDFG is more than twice as fast as the original one, since the overhead to start one block is shared among three iterations. Since all delays are grouped on one edge, we can execute the three iterations in the order $\langle A1, A2, A3\rangle$, $\langle B1, B2, B3\rangle$ and $\langle C1, C2, C3\rangle$.

The retiming transformation preserves data dependencies and the two CDFG's in Figure 1 are functionally equivalent.

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. An important feature of our approach is that *all* nodes in the retimed CDFG achieve a block-processing factor of k and process data in an identical fashion. As a result, our retimed CDFG's operate faster and are less expensive to implement than generic block-processed CDFG's. Our first contribution is an *integer linear programming* (ILP) formulation of the k-delay problem. Our second contribution is a novel formulation of the k-delay problem as a constraints graph that relies on a novel graph transformation which is then solved by an efficient algorithm whose running-time is a polynomial in the size of the CDFG. Specifically, for a given CDFG and a given positive integer k, our algorithm computes a retimed CDFG that can process k samples simultaneously, or determines that such a retiming is infeasible. Our algorithm runs in $O(V^3E + V^4 \lg V)$ time, where V is the number of computation blocks and E is the number of interconnections in the CDFG.

Recently, retiming has been studied in the context of optimum vectorization for a class of DSP programs [13, 15]. Specifically, a technique for linear vectorization of DSP programs using retiming has been presented in [15]. 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.

The remainder of this paper is organized as follows. In Section 2 we describe the representation of computations as dataflow graphs. We also give background material on block-processing and retiming, and we give a precise mathematical formulation of the retiming problem for achieving a given block-processing factor k. In Section 3, we present an integer linear programming formulation of the k-delay problem. In Section 4, we describe a novel formulation of the k-delay problem and give an efficient algorithm that solves it in polynomial time in the size of the CDFG. We conclude in Section 5 by demonstrating the performance improvements that can be achieved by using our techniques to optimize systems for regular block-processing.

2 Preliminaries

In this section, we first describe the dataflow graph representation of computations. We subsequently provide some background on block-processing and give the conditions that must be satisfied for effective block-processing. We also provide background material on retiming, and we give a mathematical formulation of the k-delay problem.

2.1 Graph representation

The CDFG of a computation structure is represented by a directed graph $G = \langle V, E, w\rangle$. The nodes $v \in V$ model the computation blocks (subroutines, arithmetic or boolean operators) and the directed edges $e \in E$ model the interconnection (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 or registers associated with that interconnection. Figure 2(a) gives the graph representation of a sample CDFG.

A delay (state) in behavioral synthesis corresponds to an iteration boundary in software compilation and a register in gate-level description. All results in this paper can be translated from one domain to the other two in a straightforward manner. Translation of results from behavioral to logic synthesis involves only semantic interpretation.

2.2 Block-Processing

We now define some useful terminology related to block-processing of computation systems.

Definition 1 *A CDFG is* **computable** *if every directed cycle contains at least one delay.*

Definition 2 *A block-processing is* **legal** *if the CDFG is computable and the initial input/output relationship is preserved.*

Definition 3 *A* **block-processing factor** *k of a particular computational block is the largest number of samples which can be processed simultaneously or immediately after each other.*

Definition 4 *A block-processing is* **linear** *if all blocks have the same block processing factor.*

Definition 5 *A block-processing is* **regular** *if the computations from all iterations of any block can be processed in the same way.*

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

Lemma 6 *A block-processing of a CDFG $G = \langle V, E, w \rangle$ that processes k samples simultaneously is linear and regular if and only if for every edge $e \in E$, we have*

$$w(e) = \begin{cases} 0, & or \\ \geq k \ . & \end{cases} \tag{1}$$

Proof. ($\Rightarrow$) Let the block-processing be regular and linear and let us assume that Relation (1) is not satisfied. This implies that there exists an edge $u \xrightarrow{i} v$ such that $1 \leq w(i) \leq k-1$. Since the block-processing is linear, node v processes k samples per iteration. However, since i contains only $w(i)$ delays and therefore only $w(i)$ samples, the remaining $k - w(i)$ samples must be accessed from data in the previous iteration. This implies that v is processing in a way different from other nodes in G which contradicts our assumption that the block-processing is regular.

($\Leftarrow$) Straightforward. $\square$

From Definition 2 and Lemma 6 we can conclude that at least one edge with k delays is required on every cycle of G for a block-processing factor of k.

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 computable and 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, 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

$$F = \max_{u,v \in V} \{W(u,v) + W(v,u)\} \ , \tag{3}$$

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

$$\begin{aligned} w_r(e) &\leq F \quad \text{for every edge } e \in E \text{ , and} && (4) \\ W_r(u,v) &\leq F \quad \text{for every vertex pair } u, v \in V \ . && (5) \end{aligned}$$

2.4 The k-delay problem

Lemma 6 implies that a linear and regular block-processing factor k can be achieved only for CDFGs that have 0 or at least k delays on each edge. Since a given CDFG may not satisfy the condition in Relation (1), we retime it to redistribute its delays so that all nodes achieve the desired block-processing factor k. The problem of computing such a retiming, which we call the k-delay problem, can be expressed in precise mathematical terms as follows.

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

$$w_r(e) = \begin{cases} 0, & \text{or} \\ \geq k \ ; & \end{cases} \tag{6}$$

or determine that no such retiming exists. □

3 ILP formulation

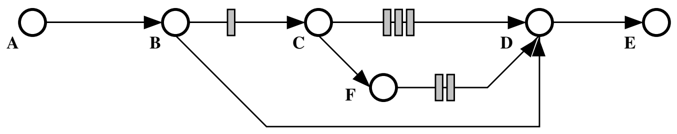
Problem KDP cannot be expressed directly in a linear programming form because of the disjunction (*or* requirement) in Relation (6). In this section we express Problem KDP as an Integer Linear Program (ILP) by utilizing the companion graph described in [8].

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

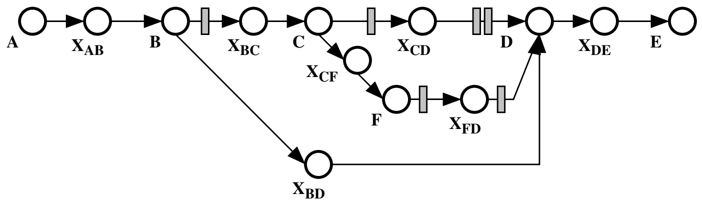
$$\begin{aligned} V' &= V \cup \left\{x_{uv} : u \xrightarrow{e} v \in E\right\} \ , \\ E' &= \left\{u \xrightarrow{e_1} x_{uv}, x_{uv} \xrightarrow{e_2} v : u \xrightarrow{e} v \in E\right\} \ , \end{aligned}$$

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

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



(a)



(b)

Figure 2: Constructing a companion graph. (a) Original CDFG $G = \langle V, E, w \rangle$. (b) Companion graph $G' = \langle V', E', w' \rangle$ generated by segmenting every edge in E into two edges by introducing a dummy vertex such that the first edge has a delay count of at most 1. Edge $C \overset{w=3}{\rightarrow} D$ in G has been segmented to generate edges $C \overset{w=1}{\rightarrow} X_{CD}$ and $X_{CD} \overset{w=2}{\rightarrow} D$ in G'.

Figure 2 illustrates the construction of the companion graph.

The following lemma gives the necessary and sufficient conditions for a retiming to satisfy the requirements of Problem KDP.

Lemma 7 *Let $G = \langle V, E, w \rangle$ denote a computation dataflow graph 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 for G if and only if there exists a retiming function $r' : V' \rightarrow \mathbf{Z}$ such that for every edge $u \overset{e}{\rightarrow} v \in E'$, we have*

$$r'(v) + w'(e) - r'(u) \geq 0 \ , \tag{7}$$

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

$$\begin{aligned} w'_{r'}(e_1) &\leq 1 \ , &(8)\\ w'_{r'}(e_2) &\leq F \cdot w'_{r'}(e_1) \ , \ \textit{and} &(9)\\ w'_{r'}(e_2) &\geq (k-1) \cdot w'_{r'}(e_1) \ . &(10)\end{aligned}$$

Proof. Inequality (7) ensures that the retimed circuit is well-formed. Inequalities (8) and (9) ensure that the delay counts in $G'_{r'}$ satisfy the definition of a companion graph. Inequality (10) ensures that for every edge $u \overset{e}{\rightarrow} v$ in E, if edge $u \overset{e_1}{\rightarrow} x_{uv}$ has a delay after retiming, then edge $x_{uv} \overset{e_2}{\rightarrow} v$ has at least $k-1$ delays. In other words, we ensure that if edge $u \overset{e}{\rightarrow} v$ 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 $u \in V$. □

The following theorem expresses Problem KDP as a set of $O(E)$ integer linear programming constraints.

Theorem 8 *Let $G = \langle V, E, w \rangle$ denote a computation flow graph 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 if and only if there exists a retiming function $r' : V' \rightarrow \mathbf{Z}$ such that for every edge $u \overset{e}{\rightarrow} v \in E'$, we have*

$$r'(v) + w'(e) - r'(u) \geq 0 \ , \tag{11}$$

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

$$\begin{aligned} r'(x_{uv}) - r'(u) + w'(e_1) &\leq 1 \ , && (12) \\ r'(v) + F \cdot r'(u) - (F+1) \cdot r'(x_{uv}) + w'(e_2) - F \cdot w'(e_1) &\leq 0 \ , \ \textit{and} && (13) \\ k \cdot r'(x_{uv}) - (k-1) \cdot r'(u) - r'(v) + (k-1) \cdot w'(e_1) - w'(e_2) &\leq 0 \ . && (14) \end{aligned}$$

Proof. Follows directly from Relation (2) and Lemma 7. □

4 Polynomial-time algorithm

The constraints in the integer linear programming 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 and give a polynomial-time algorithm to compute a solution. We first give necessary conditions for Problem KDP to be feasible for a given computation flow graph. Subsequently, we describe a transformed graph G^T and we give a set of necessary and sufficient conditions for Problem KDP on G^T. We then describe an algorithm that computes a solution to Problem KDP using the transformed graph.

4.1 Necessary conditions

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

Lemma 9 *Problem KDP is feasible for a given CDFG $G = \langle V, E, w \rangle$ only if the following conditions are satisfied.*

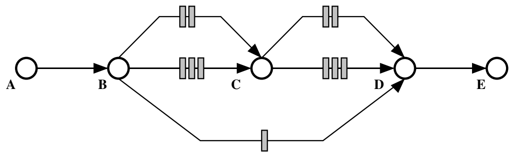
- *C1: For every vertex pair $u, v \in V$, we have $W(u,v) + W(v,u) \geq k$.*
- *C2: For every vertex pair $u, v \in V$ such that there exists a path $u \overset{p}{\rightsquigarrow} v$ in G with delay count $w(p)$ and $W(u,v) < w(p) < k$, there exists a retiming $r : V \rightarrow \mathbf{Z}$ such that in the retimed graph G_r, we have $W_r(u,v) \geq k$.*

Proof. First we prove Condition C1 by contradiction. During retiming, the delay count around a cycle remains unchanged. Moreover, $W(u,v) + W(v,u)$ equals the minimum delay count for any cycle through u and v. Thus, if $W(u,v) + W(v,u) < k$, then in the retimed graph G_r there exists at least one edge in the minimum cycle that has a delay count between 1 and $k-1$ which contradicts the fact that Problem KDP is feasible.

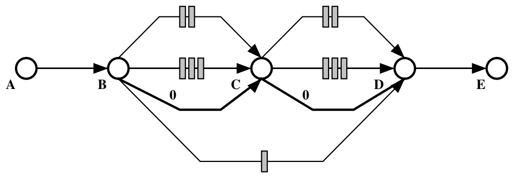
Let us assume that Condition C2 does not hold. Then for some vertex pair $u, v \in V$ and for every retiming $r : V \rightarrow \mathbf{Z}$, we have $W_r(u,v) \leq k-1$. If $W_r(u,v) = 0$, then we have $w_r(u \overset{p}{\rightsquigarrow} v) < k$. If $W_r(u,v) > 0$ then we have $W_r(u,v) \leq k-1$. Therfore, there exists a path $u \rightsquigarrow v$ in G_r that has less than k delays, and thus no edge on $u \rightsquigarrow v$ has more than $k-1$ delays which contradicts the fact that Problem KDP is feasible. □

For simplicity, we assume henceforth that we are given graphs G for which Condition C1 is already satisfied since Condition C1 can always be checked on a given graph G as a preprocessing step.

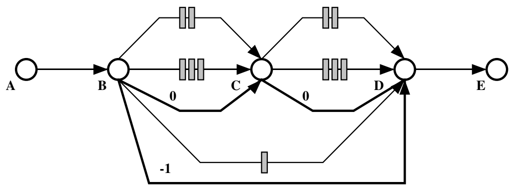
The main challenge in solving Problem KDP is to determine which edges need to contain delays and which do not. In the ILP formulation, we attempt to make this determination in an explicit manner which makes it computationally expensive. The technique described in this section, however, first determines vertex pairs whose paths need to contain delays



(a)



(b)



(c)

Figure 3: Illustration of explicit and implicit PD vertex pairs in CDFG's. 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 Part (b) are between explicit PD vertex pairs, and the weights on these edges indicate the delay excess associated with the corresponding vertex pairs. The bold edges in Part (c) denote both explicit and implicit PD vertex pairs. For example, (B, D) is an implicit PD vertex pair which becomes apparent only after PD vertex pairs (B, C) and (C, D) are made explicit.

and then redistributes delays on these paths in a greedy fashion to obtain the solution. The technique, therefore, determines which edges need to contain delays in an implicit manner.

Vertex pairs $u, v \in V$ for which the shortest path $u \overset{p}{\leadsto} v$ must necessarily contain delays in any retimed circuit whose edges satisfy Relation (6), are said to be *potentially deficient* in delays. In other words, there exists no retiming for which all the delays on $u \overset{p}{\leadsto} v$ can be pushed out of (u, v) without violating Relation (6) for an edge along some path between (u, v). We shall henceforth refer to such pairs as **potentially deficient** (PD) vertex pairs. It can be shown that the shortest path between a PD vertex pair in the retimed circuit must contain at least k delays, otherwise there would be a *deficient edge*, that is, an edge that violates Relation (6), along this path. For example vertex pairs $u, v \in V$ for which there exists a path $u \overset{p}{\leadsto} v$ with delay count $w(p)$ such that $W(u, v) < w(p) < k$, such as vertex pairs (B, C) and (C, D) in Figure 3, are potentially deficient. Therefore, Condition C2 must be imposed on them.

The following lemma shows that the necessary conditions of Lemma 9 are equivalent to the well-formedness condition of an appropriately constructed constraints graph.

Lemma 10 *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 $u \overset{p}{\leadsto} v$ with delay count $w(p)$ and $W(u, v) < w(p) < k$, add a new edge $u \xrightarrow{e} v$ with edge weight $w^*(e) = W(u, v) - k$. Then Problem KDP is feasible for G only if there exists a retiming $r : V \to \mathbf{Z}$ such that for every edge $u \xrightarrow{e} v \in E^*$ in the retimed graph G_r^**

we have

$$r(v) - r(u) + w^*(e) \geq 0 \ . \tag{15}$$

Proof. (By contradiction.) For a CDFG to be computable and well-formed, we know that every edge $e \in E$ must satisfy $w_r(e) \geq 0$. It remains to be shown that Inequality (15) must be satisfied by the additional edges in the set $E^* - E$. Consider a vertex pair $u, v \in V$ that is connected by an edge $e' \in E^* - E$. Let us assume that Problem KDP is feasible even though Inequality (15) is not satisfied by e'. From the construction of G^*, we have $w^*(e) = W(u,v) - k$. Since $r(v) - r(u) + w^*(e) < 0$ (as e' doesn't satisfy Inequality (15)), we have $r(v) - r(u) + W(u,v) - k < 0$, which implies that $W_r(u,v) < k$. Since Problem KDP is feasible, we know from Condition C2 Lemma 9 that $W_r(u,v) \geq k$, which leads us to a contradiction and the lemma follows. □

It is important to mention here that Condition C2 is necessary but not sufficient. The reason is that Condition C2 captures only *explicit* deficiencies and not *implicit* or hidden deficiencies. Let us assume that we wish to determine a solution for Problem KDP for the CDFG in Figure 3(a) with $k = 2$. Consider vertex pair (B, D) in Figure 3(a). Since the shortest and second shortest paths between $B \rightsquigarrow D$ have weights of 1 and 4 delays respectively and $k = 2$, from Lemma 9 (B, D) does not appear to be a PD vertex pair. We can, however, see that the shortest path between (B, D) must necessarily contain delays in any solution to Problem KDP since there is no way to retime the given CDFG and push the delay on edge $B \rightarrow D$ out of (B, D). This becomes obvious once we look at vertex pairs (B, C) and (C, D). Since (B, C) and (C, D) have to satisfy Condition C2, they need at least 2 delays on their shortest paths from $B \rightarrow C$ and $C \rightarrow D$. As a result, no delay along path $B \rightsquigarrow C \rightsquigarrow D$ can be moved outside (B, D). Since retiming changes the delay along paths between the same vertex pair in an identical manner, we infer that the the delay on edge $B \rightarrow D$ cannot be moved out of (B, D).

In order to expose implicit deficiencies, we construct a new graph G^T from G such that all PD vertex pairs become explicit. The graph $G^T = \langle V, E^T, w^T \rangle$ is constructed from $G = \langle V, E, w \rangle$ by iteratively adding a new edge $u \xrightarrow{i} v$ with weight $w^T(i) = W(u,v) - k$ between every PD vertex pair $u, v \in V$. Intuitively, $W(u,v) - k$ equals the *excess delay* of the pair $u, v \in V$ and indicates an upper bound on the number of delays that (u, v) can "contribute" to the rest of the graph. The delay deficiency could be positive or negative depending on $W(u,v)$ and k. As new edges are introduced, new vertex pairs can become PD as shown in Figure 3. For example, vertex pair (B, D) becomes PD only after the delay deficiencies of (B, C) and (C, D) become explicit.

The PD vertex pairs of a given circuit are determined by Algorithm AddEdges given in Figure 4 which transforms the given circuit G to generate the graph G^T. The algorithm operates in two steps. In the first step, explicit PD vertex 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 potentially deficient.) For every PD vertex pair (u, v), an edge $u \xrightarrow{e} v$ with weight $w(e) = W(u,v) - k$ is placed to ensure that $W_r(u,v)$ is at least k in the retimed circuit. In the second step, implicit PD vertex pairs are determined by comparing in each iteration, the delay counts of the shortest path in the transformed graph in the current iteration and the shortest path in the original graph, for every vertex pair. If the delay counts for these two shortest paths for a vertex pair (u, v) differ, then an edge $u \xrightarrow{e} v$ with weight $w(e) = W(u,v) - k$ is placed to ensure that $W_r(u,v)$ is at least k in the retimed circuit. The process is repeated till all

```
AddEdges(G, k)
for every vertex pair u, v ∈ V do
    m[u][v] ← FALSE        ▷ unmark all vertex pairs
Q ← ∅        ▷ Q is the set of PD vertex pairs
Run an all-pairs 2-shortest paths algorithm on G.
for every vertex pair u, v ∈ V with u ~p1~> v and u ~p2~> v being the two shortest paths between (u, v) do
    if w(p2) − w(p1) < k then        ▷ check if (u, v) is a PD vertex pair
        Q ← Q ∪ {(u, v)}
repeat
    for every PD vertex pair (u, v) ∈ Q do
        m[u][v] ← TRUE        ▷ mark vertex pair
        Introduce new edge u −e→ v with edge weight w^T(e) = W(u, v) − k
        E^T ← E^T ∪ {u −e→ v}
    G^T ← ⟨V, E^T, w^T⟩
    Q ← ∅
    Run an all-pairs shortest paths algorithm on G^T.
    for every vertex pair u, v ∈ V with W(u, v) > W^T(u, v) do
        if m[u][v] = FALSE then
            Q ← Q ∪ {(u, v)}
until Q = ∅
return G^T
```

Figure 4: Algorithm AddEdges transforms graph $G = \langle V, E, w\rangle$ by introducing edges with appropriate edge weights to generate G^T in $O(V^3E + V^4 \lg V)$ time. In the new graph G^T, all PD vertex pairs of G are made explicit.

the PD vertex pairs are 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 they differ by greater than k, then the condition $W_r(u, v) > k$ is taken care of automatically.)

The following lemma proves the correctness of the encoding of necessary conditions as additional edges of the transformed graph, generated in each iteration of Algorithm AddEdges.

Lemma 11 *Let $G = \langle V, E, w\rangle$ be a given CDFG, and let $G^i = \langle V, E^i, w^i\rangle$ be the transformed graph generated by the* **repeat** *loop of Algorithm* AddEdges *after iteration i. Let $G^{i+1} = \langle V, E^{i+1}, w^{i+1}\rangle$ be the graph constructed from G^i after iteration $i+1$ as follows. For every vertex pair $u, v \in V$ with $W(u, v) > W^i(u, v)$, add edge $u \xrightarrow{e} v$ with edge weight $w^{i+1}(e) = W(u, v) - k$. Let Problem KDP be feasible for graph G and let $r : V \to \mathbf{Z}$ be a solution. If for every edge $u \xrightarrow{e} v \in E^i$ of G^i, r satisfies*

$$r(v) - r(u) + w^i(e) \geq 0 \ ,$$

then for every edge $u \xrightarrow{e} v \in E^{i+1}$ of G^{i+1}, r satisfies

$$r(v) - r(u) + w^{i+1}(e) \geq 0 \ . \tag{16}$$

Proof. Consider a vertex pair $u, v \in V$ such that $W(u, v) > W^i(u, v)$. Since retiming changes the delay count on paths between a given vertex pair by the same amount, we have $W_r(u, v) > W_r^i(u, v)$. Since r satisfies the feasibility conditions for G^i, for every edge $u \xrightarrow{e} v \in E^i$ we have $w_r^i(e) \geq 0$. By adding edge-weights along the shortest path from

u to v in G_r^i, we infer that $W_r^i(u,v) \geq 0$. Since $W_r(u,v) > W_r^i(u,v)$, we conclude that $W_r(u,v) > 0$. Now, if $W_r(u,v) < k$ then there exists some edge along the shortest path in G_r that contains less than k delays which violates our assumption that r is a solution to Problem KDP. Therefore, we conclude that the condition $W_r(u,v) \geq k$ must hold. Now, for the additional edge $u \xrightarrow{e} v \in E^{i+1}$, we have $w_r^{i+1}(e) = W_r(u,v) - k$. Since $W_r(u,v) \geq k$, we conclude that $w_r^{i+1}(e) \geq 0$ must hold which implies that r satisfies Inequality (16). This concludes the proof of our lemma. □

The necessary conditions in Lemmas 9 and 11 for all the PD vertex pairs are enforced by Algorithm AddEdges which introduces an edge $u \xrightarrow{e} v$ with weight $w(e) = W(u,v) - k$ in graph $G = \langle V, E, w\rangle$ to generate $G^T = \langle V, E^T, w^T\rangle$. The following lemma gives the necessary conditions for the feasibility of Problem KDP on a given CDFG G in terms of the transformed graph G^T.

Lemma 12 *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 Problem KDP be feasible for graph G and let $r : V \to \mathbf{Z}$ be a solution. Then for every edge $u \xrightarrow{e} v \in E^T$, r satisfies*

$$r(v) - r(u) + w^T(e) \geq 0 \ . \tag{17}$$

Proof. From Algorithm AddEdges we know that for every edge $u \xrightarrow{e} v \in E^T - E$, we have $w^T(e) = W(u,v) - k$. Since r is a solution to Problem KDP, from Lemmas 10 and 11 we infer that $W_r(u,v) \geq k$. Since retiming changes the delay count for all paths between a given vertex pair by the same amount, we have $w_r^T(e) = W_r(u,v) - k$. Since $W_r(u,v) \geq k$, we conclude that $w_r^T(e) \geq 0$ which implies that r satisfies Inequality (17). □

It is important to note here that the edge-weight $w^T(e)$ for an additional edge introduced by Algorithm AddEdges has the same weight of $W(u,v) - k$ in the transformed graph G^T, independent of the iteration it is introduced in.

The following lemma gives the running time of Algorithm AddEdges.

Lemma 13 *In $O(V^3E + V^4 \lg V)$ steps, Algorithm* AddEdges *transforms a given* CDFG *$G = \langle V, E, w\rangle$ into G^T or determines that such a transformation is not possible.*

Proof. Steps 1-2 take $O(V^2)$ time. Since, we can show that the all-pairs 2-shortest paths can be computed in $O(VE + V^2 \lg V) + O(V^2E)$ time, Step 4 takes $O(V^2E)$ time while Steps 5-7 take $O(V^2)$ time to complete. The **repeat** loop in Step 8 can execute $O(V^2)$ times in the worst case when in each iteration only one additional edge between a PD vertex pair gets added to the circuit. The **for** loop in Steps 9-11 takes $O(V^2)$ steps since we have at most $O(V^2)$ vertex pairs. Since Johnson's algorithm can be used to compute the all-pairs shortest paths, Step 14 takes $O(VE + V^2 \lg V)$ time while the **for** loop in Step 15 takes $O(V^2)$ time. Thus the body of the **repeat** loop takes $O(VE + V^2 \lg V)$ time to complete and AddEdges terminates in $O(V^2) \cdot O(VE + V^2 \lg V) = O(V^3E + V^4 \lg V)$ steps. □

4.2 Sufficient conditions

In Section 4.1, we have shown how to obtain the necessary conditions for Problem KDP to be feasible for a given graph G. In this section, we will show that these conditions are also sufficient.

The following lemma proves an important property of G_r^T which is used in Lemma 15 to prove the sufficiency of the necessary conditions given in Lemmas 9 and 11. Intuitively, the lemma shows that if in a retimed circuit that satisfies the necessary conditions in Lemma 12 there are deficient edges $u \xrightarrow{e} v$ with less than k delays, then the delay count on such edges is the minimum delay count for the vertex pair $u, v \in V$. Consequently, delays on such edges can be retimed out without violating well-formedness.

Lemma 14 *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*

$$r(v) - r(u) + w^T(e) \geq 0 \ . \tag{18}$$

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

$$w_r^T(e) = W_r^T(u, v) \ . \tag{19}$$

Proof. (By contradiction.) Let r be a retiming such that for every edge $u \xrightarrow{e} v \in E^T$, we have $r(v) - r(u) + w^T(e) \geq 0$. Let $e' \in E$ satisfy $0 < w_r^T(e) < k$ and let $w_r^T(e') > W_r^T(u, v)$. Then by the definition of $W_r^T(u, v)$ there exists a shortest path $u \overset{p}{\rightsquigarrow} v$ such that $w_r^T(p) = W_r^T(u, v) < w_r^T(e')$. Since Inequality (17) is satisfied for every edge $u \xrightarrow{e} v \in E^T$, we infer that $w_r^T(p) \geq 0$. Since $0 \leq w_r^T(p) < w_r^T(e') < k$, we infer that $w_r^T(e') - w_r^T(p) < k$. Since retiming changes the delay count of all paths between a given vertex pair by the same amount, we infer that $w^T(e') - w^T(p) < k$. Since $e' \in E$ we have $w^T(e') = w(e')$, from which we infer that $w(e') - w^T(p) < k$. Now, if the path $u \overset{p}{\rightsquigarrow} v$ is in G, then we have $w^T(p) = w(p)$ from which we infer that $w(e') - w(p) < k$. This implies that there exist two paths in G between vertex pair $u, v \in V$ whose delay counts differ by less than k which violates Condition C2 of Lemma 9. Thus $u \overset{p}{\rightsquigarrow} v$ is a path that exists only in G^T from which we infer that the shortest path in G between $u, v \in V$ has delay count greater than $w^T(p)$ or $W(u, v) > w^T(p) = W^T(u, v)$. From the proof of Lemma 12 we know that for every vertex pair with $W(u, v) > W^T(u, v)$, we have $W_r(u, v) \geq k$. From the definition of $W_r(u, v)$, we know that $w_r(e') \geq W_r(u, v)$. Since $w_r^T(e) = w_r(e')$ and $W_r(u, v) \geq k$, we infer that $w_r^T(e) \geq k$ which contradicts our assumption that $w_r^T(e) < k$. $\square$

We now introduce some useful terminology for CDFGs. The *supervertex* $\mathcal{U}$ of a vertex u is the set of vertices x such that

$$\mathcal{U} = \left\{ x \in V : x \overset{p}{\rightsquigarrow} u \text{ in } G \text{ with } w(p) = 0 \quad \text{or} \quad u \overset{q}{\rightsquigarrow} x \text{ in } G \text{ with } w(q) = 0 \right\} \ . \tag{20}$$

We can transform any given graph G into its supergraph $\mathbf{sup}(G)$ by collapsing all vertices that are connected by zero-weight edges into a single supervertex. This transformation is illustrated in Figure 5. 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. For example in Figure 5, even though the additional edge $\mathcal{C} \to \mathcal{D}$ has no delays, the vertices $\mathcal{C}$ and $\mathcal{D}$ are not collapsed into one single vertex.

In the following lemma, we show that a retiming that minimizes the number of *edges with delays* in E subject to Inequality (17) is a solution to Problem KDP. We prove this result by contradiction. We begin with the assumption that a retiming that satisfies Inequality (17) and minimizes the number of *edges* $\in E$ *with delays* is not a solution. We then describe a way to further minimize the number of edges with delays in E. This contradicts our assumption that our original retiming had minimized the number of edges with delays in E.

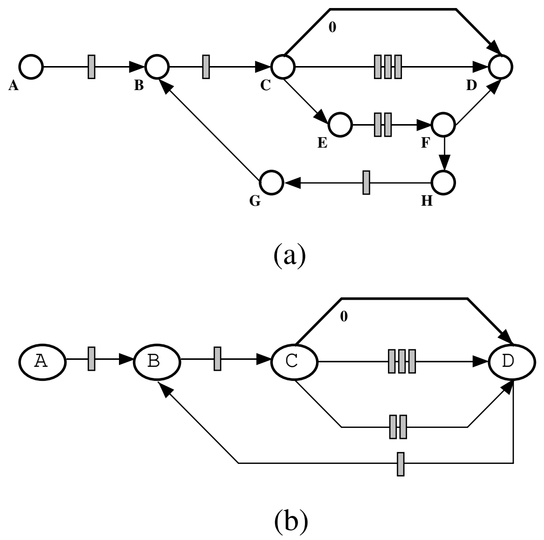


Figure 5: Generating the supergraph of a transformed CDFG. The supervertices $\mathcal{A} = \{A\}$, $\mathcal{B} = \{B, G\}$, $\mathcal{C} = \{C, E\}$, and $\mathcal{D} = \{D, F, H\}$ of the CDFG in Part (b) are obtained by collapsing vertices connected by zero-weight edges of the CDFG in Part (a). Note that vertices connected by the bold edges are not collapsed to form supervertices even though the bold edges may have zero-weight.

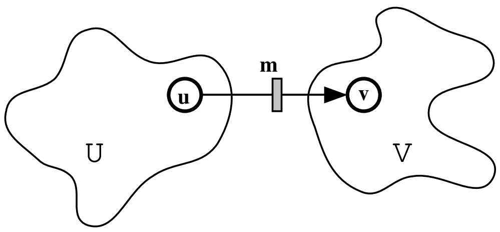


Figure 6: Deficient edge $u \to v$ between two supervertices with delay-count $m = 1$ in $\mathbf{sup}(G_r^T)$ for Problem KDP with $k = 2$.

Lemma 15 *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*

$$r(v) - r(u) + w^T(e) \geq 0 \qquad \textit{for every edge } u \xrightarrow{e} v \in E^T \ , \tag{21}$$

then r is a solution to Problem KDP.

Proof. (By contradiction.) Let r minimize the number of edges with delays under Inequality (21) and let r not be a solution. This implies that in the retimed circuit G_r^T, there are still some deficient edges $i \in E$ such that $1 \leq w_r(i) \leq k - 1$. (Note that we only care about delay counts on the edges in E and not the additional edges that are introduced when we transform G.) Let $u \xrightarrow{j} v \in E$ be such an edge with $w_r^T(j) = m$ where $1 \leq m \leq k - 1$ as shown in Figure 6. 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.

(Note that supervertices $\mathcal{U}$ and $\mathcal{V}$ have to be distinct, otherwise no additional retiming of $\mathbf{sup}(G_r^T)$ can remove all the delays from $u \xrightarrow{j} v$. The proof that $\mathcal{U}$ and $\mathcal{V}$ are distinct supervertices proceeds as follows. Let us assume that $\mathcal{U}$ and $\mathcal{V}$ are not distinct. This implies that there exist two vertices $a \in \mathcal{U}$ and $b \in \mathcal{V}$ with paths $a \overset{p}{\rightsquigarrow} b$ or $b \overset{q}{\rightsquigarrow} a$ such that $w(p) = 0$

or $w(q) = 0$. If $w(q) = 0$, then it implies the existence of a cycle $a \rightsquigarrow u \xrightarrow{j} v \rightsquigarrow b \overset{q}{\rightsquigarrow} a$ with weight $m < k$ which violates Condition C1 of Lemma 9. If $w(p) = 0$, then there exist two paths $a \rightsquigarrow u \xrightarrow{j} v \rightsquigarrow b$ and $a \overset{p}{\rightsquigarrow} b$ whose weights differ by $m < k$ and with $W_r(a, b) = 0$ which violates Condition C2 of Lemma 9. We conclude from this contradiction that $\mathcal{U}$ and $\mathcal{V}$ are distinct.)

We now show how to compute a new retiming r' that maintains the necessary conditions for Problem KDP to be feasible for G. Consider a retiming assignment of $r'(\mathcal{X}) = m - W_r^T(\mathcal{U}, \mathcal{X})$ to all supervertices $\mathcal{X}$ in $\mathbf{sup}(G_r^T)$. (Note that an assignment $r(\mathcal{X})$ is equivalent to the assignment $r(x) = r(\mathcal{X})$ for all vertices $x \in \mathcal{X}$.) From the definition of shortest paths, for every supervertex pair $(\mathcal{X}_1, \mathcal{X}_2)$ in $\mathbf{sup}(G_r^T)$ we have

$$
\begin{aligned}
W_r^T(\mathcal{U}, \mathcal{X}_2) &\leq W_r^T(\mathcal{U}, \mathcal{X}_1) + W_r^T(\mathcal{X}_1, \mathcal{X}_2) \\
-W_r^T(\mathcal{U}, \mathcal{X}_2) &\geq -W_r^T(\mathcal{U}, \mathcal{X}_1) - W_r^T(\mathcal{X}_1, \mathcal{X}_2) \\
m - W_r^T(\mathcal{U}, \mathcal{X}_2) &\geq m - W_r^T(\mathcal{U}, \mathcal{X}_1) - W_r^T(\mathcal{X}_1, \mathcal{X}_2) \\
r'(\mathcal{X}_2) &\geq r'(\mathcal{X}_1) - W_r^T(\mathcal{X}_1, \mathcal{X}_2) \\
r'(\mathcal{X}_2) - r'(\mathcal{X}_1) + W_r^T(\mathcal{X}_1, \mathcal{X}_2) &\geq 0 \, .
\end{aligned}
$$

Consequently, the resulting supergraph $\mathbf{sup}(G_r^T)_{r'}$ which is obtained by setting $r(u) + r'(u)$ equal to the retiming of $u \in V$ for G^T, satisfies Inequality (21).

We now show that 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 14 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, from which we infer that 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 15 captures our two-step strategy for obtaining a solution to Problem KDP. The first step involves transforming the graph and computing a retiming that satisfies Inequality (17) to ensure that there are enough delays on all paths of our computation graph. The second step involves bringing delays onto individual edges by employing a greedy procedure which computes an incremental retiming for every deficient 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 till there are no more deficient edges left and all edges satisfy Relation (6). The final retiming is the sum of the first step retiming and the sum of all the subsequent incremental retimings in the second step. Algorithm SolveKDP described in the Figure 7 implements this greedy procedure. This greedy procedure does not guarantee that the retimed graph has minimum number of edges

```
SolveKDP (G, k)
1   G^T ← AddEdges (G, k)
2   Compute r that satisfies Inequality (17) for every edge in G^T.
3   if no feasible r then
4       return INFEASIBLE.
5   while there exists edge u -j-> v ∈ E of G_r^T with 0 < w_r^T(j) = m < k do
6       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) do
9           r'(X) ← m − W_r^T(U, X)
10      r ← r + r'
11  return r.
```

Figure 7: Algorithm SolveKDP for solving Problem KDP.

with delays. Yet, it ensures that for every edge $u \xrightarrow{e} v \in V$, we have $w_r(e) \geq k$ whenever $w_r(e) \neq 0$.

The following lemma gives the running time of Algorithm SolveKDP.

Theorem 16 *In $O(V^3E + V^4 \lg V)$ time Algorithm* SolveKDP *computes a retiming that correctly solves Problem KDP, or determines that the problem is infeasible.*

Proof. From Lemmas 12 and 15, we know that Inequality (17) for G^T represents the necessary and sufficient conditions for Problem KDP to be feasible for a given CDFG G. Thus Algorithm SolveKDP determines that Problem KDP is infeasible in Step 3-4 when Inequality (17) cannot be satisfied. Also, we know from Lemma 15 there always exists a solution for Problem KDP when Inequality (17) is satisfied. This solution is computed iteratively in Steps 5-11. The **while** loop executes till there are no more deficient edges. In each iteration of the **while** loop, the delays on the deficient edge $u \xrightarrow{e} v$ are completely removed and distributed among other edges. The vertices connected by this edge are subsequently collapsed when $\mathbf{sup}(G_r^T)$ is constructed in Step 6. Since we have at most V vertices in $\mathbf{sup}(G_r^T)$, and in each iteration we merge two vertices into one, r eventually converges to the solution.

From Lemma 13 we know that Step 1 takes $O(V^3E + V^4 \lg V)$ time. Since there are $O(V^2)$ edges in G^T, we solve $O(V^2)$ difference constraints in $O(V)$ variables in Step 2 which takes $O(V^3)$ time using the Bellman-Ford algorithm. The body of the **while** loop in Step 5 iterates $O(V)$ times since in each iteration, we merge two vertices by eliminating the deficient edge connecting them. Step 6 takes $O(E)$ time, Step 7 takes $O(E + V \lg V)$ time, and the **for** loop in Step 8 takes $O(V)$ time. Thus Algorithm SolveKDP computes a solution to Problem KDP in $O(V^3E + V^4 \lg V) + O(V^3) + O(V) \cdot O(E + V \lg V) = O(V^3E + V^4 \lg V)$ time. □

5 Results

We applied the k-delay optimization to four examples. Our experimental results are shown in the table of Figure 8. The examples are adaptive voice echo canceler, adaptive video coder, and two examples from [15]. The results shown are from uniprocessor implementations with initiation and computation times obtained using measurements on typical DSP

Design	# cycles with original CDFG	k	# cycles with optimized CDFG	Improvement (%)
Zivojnovic 1	960	4	600	37.5
Zivojnovic 2	1200	4	660	45
Echo Canceler	1215	3	840	31
Video Coder	22500	2	13500	40

Figure 8: Experimental results for uniprocessor implementations. All times are measured as the number of cycles.

general purpose processors, such as TMS32020 and Motorola 56000. The improvement for each CDFG is given by the fraction 1-(# cycles with optimized CDFG)/(# cycles with original CDFG).

It is interesting to note that it is easy to calculate and predict the potential benefit of block-processing and k-delay retiming. If we denote the sum of the context-switching times of all nodes by O, and the sum of all computation times by C, it is straightforward to verify that after k-delay retiming the total reduction in execution time is given by the factor $k \cdot (O+C)/(O+k \cdot C)$.

References

[1] R. E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley Publishing Company, 1985.

[2] D. Eppstein. Finding the k shortest paths. Technical Report 94-26, Department of Information and Computer Science, University of California, Irvine, May 1994.

[3] W. W. Gibbs. Software's chronic crisis. *Scientific American*, pages 86–95, 1994.

[4] 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.

[5] 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*, pages 245–264. MIT Press, March 1992.

[6] 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.

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

[8] 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.

[9] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):1–27, 1991.

[10] 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.

[11] S. Malik, E. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. In *Proc. of the Hawaii International Conference on System Sciences*, June 1990.

[12] G. De Micheli. Synchronous logic synthesis: Algorithms for cycle-time minimization. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 10(1):63–73, 1991.

[13] 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.

[14] 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.

[15] 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.