Optimizing Computations for Effective Block-Processing

KUMAR N. LALGUDI
Intel Corporation
MARIOS C. PAPAEFTHYMIOU
University of Michigan
and
MIODRAG POTKONJAK
University of California

Block-processing can decrease the time and power required to perform any given computation by simultaneously processing multiple samples of input data. The effectiveness of block-processing can be severely limited, however, if the delays in the dataflow graph of the computation are placed suboptimally. In this paper we investigate the application of retiming for improving the effectiveness of block-processing in computations. In particular, we consider the *k-delay problem*: Given a computation dataflow graph and a positive integer k, we wish to compute a retimed computation graph in which the original delays have been relocated so that k data samples can be processed simultaneously and fully regularly. We give an exact integer linear programming formulation for the k-delay problem. We also describe an algorithm that solves the k-delay problem fast in practice by relying on a set of necessary conditions to prune the search space. Experimental results with synthetic and random benchmarks demonstrate the performance improvements achievable by block-processing and the efficiency of our algorithm.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Signal processing systems*; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Combinatorial optimization, computation dataflow graphs, embedded systems, high-level synthesis, integer linear programming, retiming, scheduling, vectorization

This research was supported in part by the National Science Foundation under grant MIP-9610108 and grant CCR 9796145.
Authors' addresses: K. N. Lalgudi, Intel Corporation, Hillsboro, OR 97124; email: klalgudi@ichips.intel.com; M. C. Papaefthymiou, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109; email: marios@eecs.umich.edu; M. Potkonjak, Department of Computer Science, University of California, Los Angeles, CA 90095; email: miodrag@cs.ucla.edu.
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 2000 ACM 1084-4309/00/0700–0604 $5.00

1. INTRODUCTION

In many application domains, computations are defined on very long streams of data whose arrival rate is dictated by the nature of the application. 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 the computational requirements per sample of state-of-the-art communication have been doubling every year, the processing power of hardware is doubling only every three years [Gibbs 1994]. Furthermore, new applications requirements such as low power dissipation impose additional design constraints, which 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 emerging communication applications, it is often necessary to simultaneously compute 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 greatly facilitates the efficient implementation of computations on various hardware platforms [Guerra et al. 1994; Kung 1988]. In particular, block-processing enhances regularity, thus reducing the effort in address calculation and software context switching. It also enhances locality in computations and consequently improves the effectiveness of code-size reduction techniques [Liao et al. 1995]. Furthermore, block-processing enables efficient utilization of pipelines and efficient implementation of vector-based algorithms such as FFT-based filtering and error-correction codes [Blahut 1985]. In general, block-processing is beneficial in all cases where the net cost of processing n samples simultaneously is lower than the net cost of processing these samples on an individual basis. 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. For example, one can unfold the basic iteration of a computation and schedule computation blocks from different iterations to execute successively. However, this technique may not uniformly increase the block-processing factor for all computational blocks.

Another transformation that can be used for increasing the block-processing factor is *retiming*. Unlike other architectural transformations that have targeted high-level synthesis [Ku and De Micheli 1992; Walker and Thomas 1989], retiming has been primarily explored at relatively low abstraction levels in the contexts of timing optimization [Ishii et al. 1992; Lalgudi and Papaefthymiou 1995; Leiserson and Saxe 1991; Lockyear and Ebeling 1993] and logic synthesis [Malik et al. 1990 ; De Micheli 1991].

Figure 1 illustrates the use of retiming for improving 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

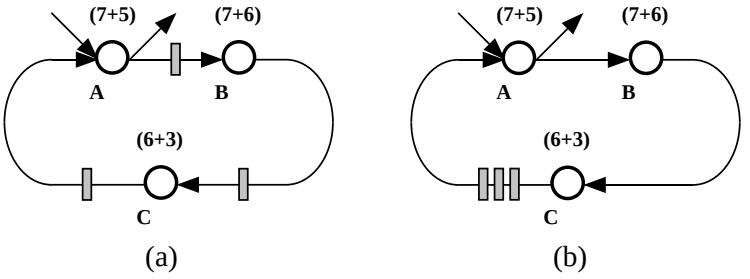


Fig. 1. Improving the effectiveness of block-processing by retiming. The block-processing factor of the original computation dataflow graph in part (a) is 1. The block-processing factor of the retimed graph in part (b) is 3.

block A, and an output stream is generated by A. Assuming that the computation is implemented by a uniprocessor system, the expression $(x + y)$ next to each block gives the *initiation time* x and the *computation time* y per block input. The initiation time includes the context-switch 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. On the other hand, the functionally equivalent CDFG in Figure 1(b), which is 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. 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 input samples.

Recently, retiming has been studied in the context of optimum vectorization for a class of DSP programs [Ritz et al. 1993 ; Zivojnovic et al. 1994]. In particular, a retiming-based technique for linear vectorization of DSP programs was presented in Zivojnovic et al. [1994]. This technique involves the redistribution of delays in the CDFG representation of a DSP program in a way that maximizes the concentration of delays on the edges. However, fully regular vectorization cannot be achieved using the linear vectorization approach in that paper. Moreover, the retiming problem for computing linear vectorizations is formulated as a nonlinear program, which can be computationally very expensive to solve.

In this paper we consider the problem of retiming computation dataflow graphs to achieve a given block-processing factor k. We call this the k-delay problem. We first give an exact *integer linear programming* (ILP) formulation of the k-delay problem. We then present a set of necessary conditions which we use to develop a branch-and-bound algorithm for solving the

k-delay problem efficiently in practice. 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 no such retiming exists. 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 are less expensive to implement than generic block-processed CDFGs. We provide extensive experimental results that demonstrate the effectiveness of our optimization and the efficiency of our algorithms.

The remainder of this paper is organized as follows. In Section 2 we describe the representation of computations as dataflow graphs and 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 give a set of necessary conditions that impose nonzero lower bounds on the delay counts of cycles and paths of any retimed CDFG that solves the k-delay problem. These bounds can be used to generate additional necessary conditions using an iterative procedure that we describe in Section 5. In Section 6 we identify paths that must have zero delay counts in every retiming of the original CDFG that solves the k-delay problem. In Section 7 we describe a practical branch-and-bound algorithm for the k-delay problem that relies on our necessary conditions. We present an experimental evaluation of our algorithm in Section 8, and conclude with directions for future research in Section 9.

2. PRELIMINARIES

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

2.1 Graph Representation

The computation dataflow graph (CDFG) of a computation structure is an edge-weighted 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 3(a) gives the graph representation of a sample CDFG.

A delay (state) in behavioral synthesis corresponds to an iteration boundary in software compilation or a register in gate-level description. All results in this paper can be straightforwardly translated from the behavioral domain to the other two.

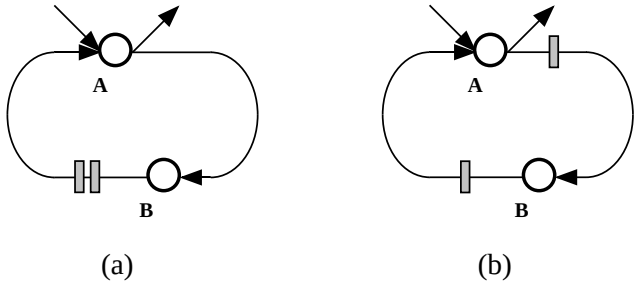


Fig. 2. Types of linear block-processing. (a) Regular and linear block-processing with a factor 2. (b) Irregular but linear block-processing with a factor 2.

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 $k \cdot |V|$ evaluations of a computation block 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 that are simultaneously processed by every computation block are accessed during the same block iteration. The retimed CDFG in Figure 2(a), for example, can be block-processed linearly and regularly with a block-processing factor of 2. The computation blocks for this CDFG execute in the order A_1, A_2, B_1, B_2, A_3, A_4, B_3, B_4, $\cdots$ and two input samples are consumed in each block iteration. Regular block-processing results in more efficient implementations of CDFGs because it reduces the costs of address calculation and software switching. For the first block iteration in the CDFG of Figure 2(a), the indices 1 and 2 computed for block A can also be used for block B. A linear block-processing is not necessarily regular. For example, the CDFG in Figure 2(b) can achieve a block-processing factor of 2 with the computation blocks executing in the order A_1, B_1, B_2, A_2, A_3, B_3, B_4, $\cdots$. This block-processing is *irregular*, however, because the computation blocks process different samples in a given block iteration. In the first block iteration, for example, block B processes samples 1 and 2, while block A processes samples 2 and 3.

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

LEMMA 1. *Let $G = \langle V, E, w \rangle$ be a 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*

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

PROOF. ($\Rightarrow$) By contradiction. Suppose that Relation (1) is not satisfied for some CDFG that can be block-processed linearly and regularly with a factor 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

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

The retimed CDFG G_r is well-formed if and only if for all edges $e \in E$ we have $w_r(e) \geq 0$.

Several important properties of the retiming transformation stem directly from Eq. (2). One such property that we repeatedly use in the proofs of this paper is that for any given vertex pair u, v in V, a retiming r changes the original delay count of every path from u to v by the same amount. To verify this property, we express the postretiming delay count $w_r(p)$ along any such path p as the sum of the delay counts of its constituent edges:

$$w_r(p) = \sum_{e \in p} w_r(e)$$

$$= \sum_{e \in p} (w(e) + r(v) - r(u)) \tag{3}$$

$$= (r(v) - r(u)) + \sum_{e \in p} w(e), \tag{4}$$

since the sum in Eq. (3) telescopes. Thus, the change in the delay count of any path $u \overset{p}{\rightsquigarrow} v$ is $r(v) - r(u)$, which depends only on the endpoints of the path.

The following corollary follows immediately from Eq. (4) and is used extensively in the derivation of the necessary conditions in this paper.

Corollary 2. [Leiserson and Saxe 1991]. Let $G = \langle V, E, w \rangle$ be a given CDFG, and let r be a retiming function. Then for any path p in G, we have $w_r(p) = w(p)$.

Based on this corollary, it is straightforward to show that for any given edge $e \in E$, the maximum number of delays that any retiming can place on e cannot exceed

$$F = \max_{u \xrightarrow{e} v \in E} \{w(e) + W(v, u)\}, \tag{5}$$

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

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 exactly 0 or at least k delays on each edge. If a given CDFG does not satisfy the condition in Relation (1), we can redistribute its delays by retiming so that all nodes achieve the desired block-processing factor k. 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 $u \xrightarrow{e} v$ in E, we have

$$w_r(e) = 0 \text{ or } w_r(e) \geq k. \tag{6}$$

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 rely on the notion of the "companion graph" described in Lalgudi and Papaefthymiou [1995] to express problem KDP as an integer linear program (ILP).

The *companion graph* $G' = \langle V', E', w'\rangle$ 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, we have

$$V' = V \cup \{x_{uv} : u \xrightarrow{e} v \in E\},$$

$$E' = \{u \xrightarrow{e_1} x_{uv}, x_{uv} \xrightarrow{e_2} v : u \xrightarrow{e} v \in E\},$$

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

$$w'(e_1) = \min\{1, w(e)\},$$

$$w'(e_2) = w(e) - \min\{1, w(e)\}.$$

Figure 3 illustrates the construction of a CDFG's companion graph. Given the original graph G in Figure 3(a), the companion graph G' in

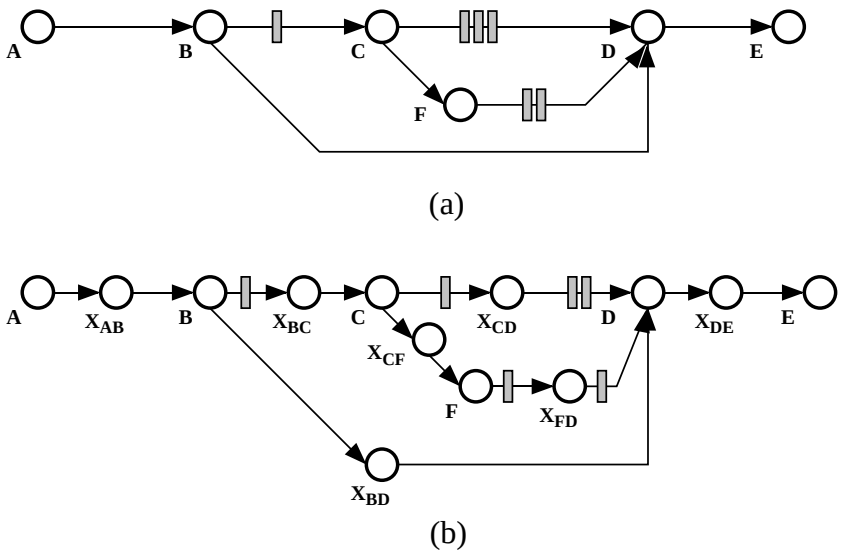


Fig. 3. Generation of companion graph. (a) Original CDFG $G = \langle V, E, w \rangle$; (b) corresponding companion graph $G' = \langle V', E', w' \rangle$.

Figure 3(b) is generated by replacing each edge e in G by two edges e_1 and e_2 connected in series by a dummy vertex. If the delay count of the original edge is nonzero, then the delay count of e_1 equals 1, and the delay count of e_2 equals $w(e)-1$. If the delay count $w(e)$ is zero, the delay counts of e_1 and e_2 are also zero. For example, the edge $C \rightarrow D$ in G is segmented into the edges $C \rightarrow X_{CD}$ and $X_{CD} \rightarrow D$ with delay counts 1 and 2, respectively.

The following lemma gives the necessary and sufficient conditions that hold for any retiming that solves problem KDP.

LEMMA 3. *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 $u \xrightarrow{e} v$ in E', we have*

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

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

$$w'_{r'}(u \rightarrow x_{uv}) \leq 1, \tag{8}$$

$$w'_{r'}(x_{uv} \rightarrow v) \leq F \cdot w'_{r'}(u \rightarrow x_{uv}), \tag{9}$$

$$w'_{r'}(x_{uv} \rightarrow v) \geq (k-1) \cdot w'_{r'}(u \rightarrow x_{uv}). \tag{10}$$

PROOF. ($\Rightarrow$) Let r be a retiming function that solves problem KDP and thus satisfies Relation (6). In this case, it is straightforward to verify that the function

$$r'(x) = \begin{cases} r(x) & x \in V, \\ r(u) - w'(u \to x) + \min\{1, w(e) + r(v) - r(u)\} & x \in V' - V, u \to x \in E', u \xrightarrow{e} v \in E, \end{cases}$$

satisfies Inequalities (7)–(10).

($\Leftarrow$) Let r' be a solution for Inequalities (7)–(10), and let $r(u) = r'(u)$ for all $u \in V$. By the definitions of the function r and the companion graph G', it follows that

$$\begin{aligned} w_r(u \xrightarrow{e} v) &= r(v) + w(e) - r(u) \\ &= r'(v) + w(e) - r'(u) \\ &= r'(v) + (w(e) - \min\{1, w(e)\}) + \min\{1, w(e)\} - r'(u) \\ &= r'(v) + (w(e) - \min\{1, w(e)\}) - r'(x_{uv}) + r'(x_{uv}) + \min\{1, w(e)\} - r'(u) \\ &= r'(v) + w'(e_2) - r'(x_{uv}) + r'(x_{uv}) + w'(e_1) - r'(u) \\ &= w'_{r'}(x_{uv} \to v) + w'_{r'}(u \to x_{uv}). \end{aligned} \tag{11}$$

Now, a simple case analysis shows that r is a solution to problem KDP. If $w'_{r'}(u \to x_{uv}) = 0$, then Inequality (10) implies that $w'_{r'}(x_{uv} \to v) = 0$. From Eq. (11) it follows that $w_r(u \to v) = 0$, and thus Relation (6) holds. If $w'_{r'}(u \to x_{uv}) = 1$, then from Inequality (2) and Eq. (2) it follows that $w_r(u \to v) \geq k$, and thus Relation (6) holds. □

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

THEOREM 4. *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 $r : V \to \mathbf{Z}$ that solves problem KDP if and only if there exists a retiming function $r' : V' \to \mathbf{Z}$ such that for every edge $u \xrightarrow{e} v \in E'$, we have*

$$r'(v) + w'(e) - r'(u) \geq 0, \tag{12}$$

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

$$r'(x_{uv}) - r'(u) + w'(e_1) \leq 1, \tag{13}$$

$$r'(v) + F \cdot r'(u) - (F + 1) \cdot r'(x_{uv}) + w'(e_2) - F \cdot w'(e_1) \leq 0, \tag{14}$$

$$k \cdot r'(x_{uv}) - (k - 1) \cdot r'(u) - r'(v) + (k - 1) \cdot w'(e_1) - w'(e_2) \leq 0. \tag{15}$$

PROOF. Follows directly from Equation (2) and Inequalities (7) and (10) from Lemma 3. □

4. NONZERO LOWER BOUNDS

The main challenge in solving problem KDP is to determine which edges should have delays in the retimed CDFG. In the ILP formulation of the problem, these edges are identified explicitly through Inequalities (13), (14), and (15). Unfortunately, the resulting constraints do not appear to have any special structure and can only be solved using general ILP solvers. For large CDFGs, this option is impractical. In this section we give two sets of necessary conditions that impose nonzero lower bounds on the delay counts of cycles and paths in the retimed CDFG. In conjunction with the necessary conditions we give in Sections 5 and 6, these conditions can be used to prune the solution space of the k-delay problem and to design algorithms that solve it efficiently in practice.

The first set of necessary conditions ensures that there exist sufficiently many delays around the directed cycles of a CDFG to achieve a given block-processing factor k. Intuitively, the redistribution of the delays around the cycles of the CDFG is possible only if every cycle in the graph has at least k delays. This statement is formalized in the following lemma.

LEMMA 5. *Let $G = \langle V, E, w \rangle$ be a CDFG. Problem KDP is feasible only if for every vertex pair u, v in V, we have*

$$W(u, v) + W(v, u) \geq k. \tag{16}$$

PROOF. By contradiction. Suppose problem KDP is feasible and there exists a vertex pair u, v for which Inequality (16) does not hold. Let $u \overset{p}{\rightsquigarrow} v$ and $v \overset{q}{\rightsquigarrow} u$ be directed paths in G such that $w(p) = W(u, v)$ and $w(q) = W(v, u)$. Then, for the directed cycle $c = u \overset{p}{\rightsquigarrow} v \overset{q}{\rightsquigarrow} u$, we have $w(c) \leq k - 1$. From Corollary 2, it follows that for any retiming r, the cycle c must include an edge e such that $w_r(e) = w(c) \leq k - 1$, thus contradicting the feasibility of problem KDP. □

For every vertex pair u, v in V, the quantities $W(u, v)$ can be computed efficiently in $O(VE + V^2 \lg V)$ steps by an all-pairs shortest-paths computation [Cormen et al. 1990]. This computation can be performed as a preprocessing step that quickly rejects the CDFGs with insufficient delays.

The second set of necessary conditions imposes a lower bound on the delay count of selected paths in the retimed CDFG. This lower bound is a direct consequence of Corollary 2. Specifically, consider any pair of vertices u, v in V that are connected in the original CDFG by two paths whose delay counts differ by less than k. If one of these paths is the shortest one from u to v, then for every retiming r that solves problem KDP, the delay counts of all paths $u \rightsquigarrow v$ must be greater than or equal to k, since

retiming changes the delay counts of all these paths by the same amount. The following theorem gives a formal proof of this statement.

LEMMA 6. *Let $G = \langle V, E, w \rangle$ be a given CDFG, and let $r : V \rightarrow \mathbf{Z}$ be a solution to problem KDP. Then for every vertex pair $u, v \in V$ such that there exists a path $u \overset{p}{\rightsquigarrow} v$ in G with $0 < w(p) - W(u, v) < k$, we have*

$$W_r(u, v) \geq k, \tag{17}$$

where $W_r(u, v) = W(u, v) + r(v) - r(u)$.

PROOF. By contradiction. Suppose that r solves problem KDP and that $W_r(u, v) \leq k - 1$ for some path $u \overset{p}{\rightsquigarrow} v$ which satisfies the condition in the lemma. We show by a case analysis that there exists an edge $e \in E$ for which Relation (6) does not hold.

If $W_r(u, v) > 0$, then for the path $u \overset{q}{\rightsquigarrow} v$ with $w_r(q) = W_r(u, v)$, we have $0 < w_r(q) \leq k - 1$. Therefore, for some edge e in q, we have $0 < w_r(e) \leq k - 1$, thus violating Relation (6).

If $W_r(u, v) = 0$, consider the path $u \overset{p}{\rightsquigarrow} v$ in the statement of the lemma. For the retimed delay count $w_r(p)$, we have

$$\begin{aligned} w_r(p) &= w(p) + r(v) - r(u) \\ &< (k + W(u, v)) + r(v) - r(u) \\ &= k + (W(u, v) + r(v) - r(u)) \\ &= k + W_r(u, v) \\ &= k, \end{aligned} \tag{18}$$

since $W_r(u, v) = 0$. Furthermore, we have

$$\begin{aligned} w_r(p) &= w(p) + r(v) - r(u) \\ &> W(u, v) + r(v) - r(u) \\ &= W_r(u, v) \\ &= 0. \end{aligned} \tag{19}$$

From Inequalities (18) and (19), it follows that for some edge e in p, we have $0 < w_r(e) \leq w_r(p) < k$, and therefore Relation (6) does not hold. □

```
LB(G, k)
 1 Q ← ∅
 2 for each u, v in V
 3     compute shortest path u ⇝(p1) v
 4 for each u, v in V
 5     do if W(u, v) + W(v, u) < k
 6           then return FAIL
 7 for each u, v in V
 8     compute strictly-second shortest path u ⇝(p2) v
 9 for each u, v in V
10     do if w(p2) − w(p1) < k
11           then Q ← Q ∪ {(u, v)}
12 return Q
```

Fig. 4. Pseudocode for Algorithm LB, which checks the cycle bounds in Lemma 5 and computes the path bounds in Lemma 6. If some cycle bound is violated, the algorithm returns FAIL. Otherwise, it returns a set of vertex pairs u, v with $W_r(u, v) \geq k$.

The $O(V^2)$ vertex pairs u, v in Lemma 6 can be computed efficiently using Algorithm LB in Figure 4. During the execution of the algorithm, these pairs are stored in the variable $\mathcal{Q}$. Algorithm LB proceeds by computing the shortest path $u \overset{p_1}{\rightsquigarrow} v$ for each vertex pair u, v in V (lines 2–3). It then verifies the condition $w(c) \geq k$ for all directed cycles c in G (lines 4–6). In lines 7–8, it computes the strictly-second shortest path $u \overset{p_2}{\rightsquigarrow} v$ for each vertex pair u, v in V. If $w(p_2) - w(p_1) < k$, then the vertex pair u, v is included in $\mathcal{Q}$ (lines 9–11).

In the following theorem we show that Algorithm LB is correct and runs in polynomial time.

THEOREM 7. *In $O(V^2E)$ time, Algorithm LB correctly checks all cycle bounds in Lemma 5 and identifies all vertex pairs in the statement of Lemma 6.*

PROOF. The all-pairs shortest-paths computation in lines 2–3 can be performed in $O(VE + V^2\lg V)$ time using Johnson's algorithm [Cormen et al. 1990]. The $O(V^2)$ inequalities in lines 4–6 can be checked in $O(V^2)$ time. The all-pairs strictly-second shortest-paths computation in lines 7–8 can be performed in $O(V^2E)$ time using the algorithm from Lalgudi and Papaefthymiou [1997]. The $O(V^2)$ inequalities in lines 9–11 can be checked in $O(V^2)$ time. Therefore, the overall running time of Algorithm LB is $O(VE + V^2\lg V) + O(V^2) + O(V^2E) + O(V^2) = O(V^2E)$.

The cycle bounds from Lemma 5 are checked in lines 4–6, and their verification is correct by construction. It remains to show that Algorithm LB identifies exactly all vertex pairs u, v in V for which there exists a path $u \overset{p}{\rightsquigarrow} v$ such that $0 < w(p) - W(u, v) < k$. For each path $u \overset{p_1}{\rightsquigarrow} v$ computed in lines 2–3, we have $w(p_1) = W(u, v)$ by construction. Since each

path $u \overset{p_2}{\rightsquigarrow} v$ in lines 7–8 is *strictly* shorter than p_1, we have $w(p_1) < w(p_2)$, and therefore $0 < w(p) - W(u, v)$ holds for $p = p_2$. Moreover, since each path $u \overset{p_2}{\rightsquigarrow} v$ is the *second* shortest one between u and v, it follows that for every path $u \overset{p}{\rightsquigarrow} v$ with $w(p) \neq w(p_1)$ and $w(p) \neq w(p_2)$, we have $w(p) \geq w(p_2)$. Therefore, if $w(p_2) - w(p_1) < k$, then $w(p) - W(u, v) < k$ holds for $p = p_2$ and the pair u, v is correctly included in $\mathcal{Q}$. Conversely, if $w(p_2) - w(p_1) \geq k$, then $w(p) - W(u, v) \geq w(p_2) - w(p_1) \geq k$ for all paths $u \overset{p}{\rightsquigarrow} v$ with $w(p) > W(u, v)$, and the pair u, v is correctly *not* included in the set $\mathcal{Q}$. □

A key point for the asymptotically efficient operation of Algorithm LB is that the strictly-second shortest paths computed in lines 7–8 are not required to be simple. The problem of computing the strictly-second shortest simple path between a given vertex pair is NP-complete. Fortunately, the corresponding problem without the simple path requirement can be solved in polynomial time [Lalgudi and Papaefthymiou 1997]. For graphs that satisfy Inequality (16) in Lemma 5, if a strictly-second shortest path $u \overset{p_2}{\rightsquigarrow} v$ is nonsimple, then $w(p_2) - W(u, v) \geq k$ and p_2 includes a single directed cycle c. (Otherwise, since $w(c) \geq k$ for every directed cycle c in G, a simple path q with $w(q) < W(u, v)$ can be found by eliminating the directed cycle in p_2.) It follows that $w(p_2) - W(u, v) < k$ only if the strictly-second shortest path is simple.

A retiming that satisfies the necessary conditions in Lemmas 5 and 6 can be computed by solving a single-source shortest-paths problem on an auxiliary graph G_a. This graph is generated by adding to the original graph G an edge $u \rightsquigarrow v$ with weight $w(e) = W(u, v) - k$ for all vertex pairs in the set $\mathcal{Q}$ returned by Algorithm LB.

THEOREM 8. *Let $G = \langle V, E, w \rangle$ be a given CDFG, and let k be a given positive integer. Moreover, let $G_a = \langle V_a, E_a, w_a \rangle$ be an auxiliary graph, which is defined as follows:*

$$V_a = V,$$

$$E_a = E \cup \left\{ u \overset{e}{\rightarrow} v : u \overset{p}{\rightsquigarrow} v \in G \text{ and } 0 < w(p) - W(u, v) < k \right\},$$

$$w_a(e) = \begin{cases} w(e) & e \in E, \\ W(u, v) - k & e \in E_a - E. \end{cases}$$

A retiming function $r : V \rightarrow \mathbf{Z}$ solves problem KDP only if for every edge $u \overset{e}{\rightarrow} v \in E_a$, we have

$$r(v) - r(u) + w_a(e) \geq 0. \tag{20}$$

```
AUX1(G, k)
 1 Q ← LB(G, k)
 2 V_a ← V
 3 E_a ← E
 4 for each vertex pair u, v in Q
 5     do E_a ← E_a ∪ {u → v}
 6 for each edge u -e-> v in E_a
 7     do if e ∈ E
 8           then w_a(e) ← w(e)
 9           else w_a(e) ← W(u, v) − k
10 return G_a
```

Fig. 5. Algorithm AUX1 for computing the auxiliary graph G_a.

PROOF. From Eq. (2) and the definition of well-formedness, we have that Inequality (20) holds for every edge $e \in E$. It remains to show that Inequality (20) holds for every edge in the set $E_a - E$. Consider an edge $u \xrightarrow{e}$ in $e \in E_a - E$ with $w_a(e) = W(u, v) - k$. For the purpose of contradiction, suppose that problem KDP is feasible and that $r(v) - r(u) + w_a(e) < 0$. We then have

$$\begin{aligned} W_r(u, v) &= W(u, v) + r(v) - r(u) \\ &= (w_a(e) + k) + r(v) - r(u) \\ &= (w_a(e) + r(v) - r(u)) + k \\ &< k. \end{aligned} \tag{21}$$

Since r solves problem KDP, Lemma 6 implies that $W_r(u, v) \geq k$, which contradicts Inequality (22). □

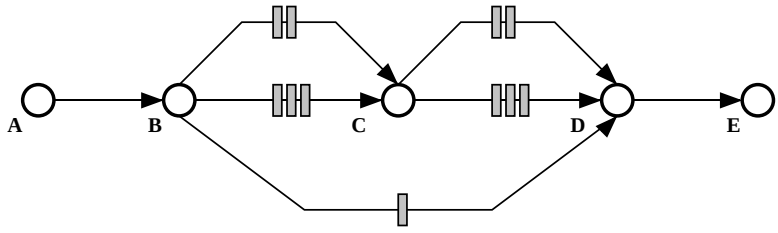
The auxiliary graph G_a can be computed in $O(V^2E)$ time using Algorithm AUX1 in Figure 5.

THEOREM 9. *In $O(V^2E)$ time, Algorithm AUX1 correctly computes an auxiliary graph $G_a = \langle V_a, E_a, w_a \rangle$ such that a retiming function $r : V \rightarrow \mathbf{Z}$ solves problem KDP only if for every edge $u \xrightarrow{e} v \in E_a$, we have*

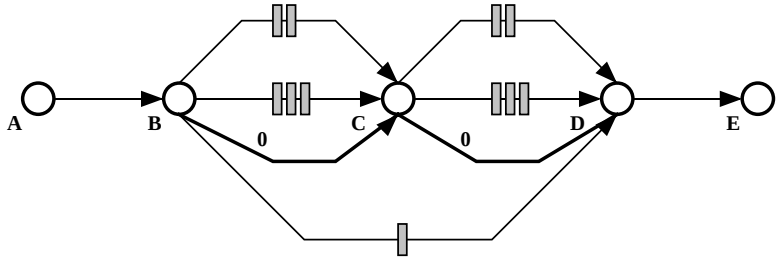
$$r(v) - r(u) + w_a(e) \geq 0. \tag{22}$$

PROOF. The running time of Algorithm AUX1 follows from Theorem 7. The correctness follows from Theorem 8. □

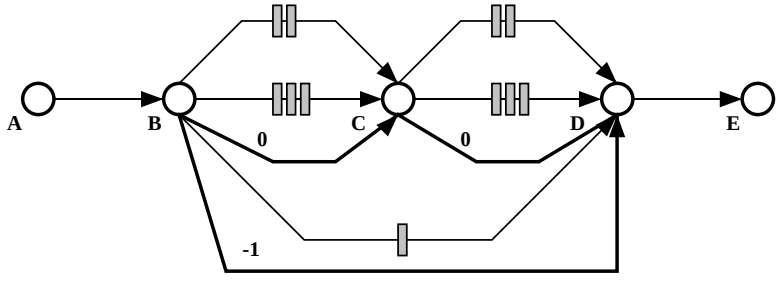
A retiming function r that satisfies Inequality (20) can be computed in $O(V^3)$ time by running a Bellman-Ford single-source shortest-paths algorithm on the graph G_a with $O(V^2)$ edges. Therefore, including the computation of G_a, the total running time required for computing a retiming function r that satisfies the necessary conditions in Lemmas 5 and 6 is $O(V^2E) + O(V^3) = O(V^2E)$ steps.



(a)



(b)



(c)

Fig. 6. Illustration of lower bounds on path delay counts. (a) Original dataflow graph G; (b) auxiliary graph G_a. The two bold edges introduce lower bounds on the delay counts of the edges connecting the vertex pair B, C and the vertex pair C, D. (c) Auxiliary graph G_a augmented by an edge $B \rightarrow D$. This new constraint arises by combining the constraints of problem KDP with the constraints on the vertex pairs B, C and C, D.

5. GENERATION OF IMPLICIT LOWER BOUNDS

The constraints described in Lemmas 5 and 6 can be combined to generate additional lower bounds on path delays for every retiming that solves problem KDP. In this section we present these implicit necessary conditions and describe a polynomial-time procedure for generating them systematically.

```
AUX2(G, k)
 1 Q ← LB(G, k)
 2 repeat
 3       V_a ← V
 4       E_a ← E
 5       for each vertex pair u, v in Q
 6           do E_a ← E_a ∪ {u → v}
 7       for each edge u -e-> v in E_a
 8           do if e ∈ E
 9                 then w_a(e) ← w(e)
10                 else w_a(e) ← W(u, v) − k
11       T ← ∅
12       Compute all quantities W_a(u, v) using an all-pairs shortest paths algorithm
13       if there exists a directed cycle in G_a with negative edge weight
14           then return FAIL
15       for every vertex pair u, v in V with W(u, v) > W_a(u, v)
16           do T ← T ∪ {(u, v)}
17       Q ← Q ∪ T
18 until T = ∅
19 return G_a
```

Fig. 7. Algorithm AUX2 for generating the augmented auxiliary graph G_a.

The example in Figure 6 illustrates the circumstances under which the new necessary conditions can arise. Assume we wish to solve problem KDP on the original CDFG in Figure 6(a) for $k = 2$. The delay counts of the shortest and second-shortest paths between vertices B and D are 1 and 4, respectively. Thus, Lemma 6 does not introduce any constraint on the delays of the paths with endpoints B and D. The shortest path from B to D in any retimed CDFG must necessarily contain two delays, however, since the single delay on the edge $B \rightarrow D$ cannot be moved. From Lemma 6, the delay counts of every edge $B \rightarrow C$ and every edge $C \rightarrow D$ should be at least 2. Therefore, the delays along $B \rightsquigarrow C \rightsquigarrow D$ cannot be moved outside the endpoints B and D. From Corollary 2, it follows that the original delay on the edge $B \rightarrow D$ cannot be moved either.

The new necessary conditions can be expressed as single-source shortest-paths constraints on a modified version of the original auxiliary graph G_a generated by Algorithm AUX1. The modified G_a is obtained by iteratively augmenting the original G_a with appropriately weighted edges. Algorithm AUX2 in Figure 7 performs this augmentation. Initially, it identifies the set of vertex pairs $\mathcal{Q}$ that are used by Algorithm AUX1 to generate G_a (line 1). The set $\mathcal{Q}$ is updated throughout the execution of the algorithm to include new vertex pairs that constitute endpoints of paths with nonzero delay counts. The augmentation of the auxiliary graph occurs iteratively in the body of the **repeat** loop in lines 2–17. In the beginning of each iteration, a new G_a is constructed by inserting edges between the vertex pairs in $\mathcal{Q}$

(lines 3–10). Subsequently, new vertex pairs are included in $\mathcal{Q}$ (lines 11–17). In lines 12–14, the algorithm performs an all-pairs shortest-paths computation to determine the quantities $W_a(u, v)$ in the current G_a. If a negative weight cycle is detected, the problem is infeasible and Algorithm AUX2 returns FAIL. Each new vertex pair u, v is determined by comparing the delay count $W_a(u, v)$ in the graph G_a of the current iteration with the delay count $W(u, v)$ in the original graph G. If $W(u, v) > W_a(u, v)$, then an edge $u \xrightarrow{e} v$ with weight $w_a(e) = W(u, v) - k$ will be introduced in the beginning of the next iteration to ensure that $W_r(u, v) \geq k$. Intuitively, the difference $W(u, v) - k$ equals the *excess delays* on each path between the vertices u, v which may be "contributed" to the remainder of the graph. The new vertex pairs are temporarily stored in the set $\mathcal{T}$. The **repeat** loop terminates when the set $\mathcal{T}$ is empty, in which case no further augmentation of G_a is possible. When Algorithm AUX2 terminates, it returns the fully augmented graph G_a.

In the following lemma, we prove that the constraints introduced in each iteration of Algorithm AUX2 are necessary for problem KDP.

LEMMA 10. *Let $G = \langle V, E, w \rangle$ be a given CDFG, and let k be a given positive integer. Let $G_a^i = \langle V_a, E_a^i, w_a^i \rangle$ be the corresponding auxiliary graph generated by Algorithm AUX2 after i iterations of the **repeat** loop. Moreover, let $G_a^{i+1} = \langle V_a, E_a^{i+1}, w_a^{i+1} \rangle$ be the auxiliary graph generated after $i + 1$ iterations by augmenting E_a^i as follows: For every vertex pair u, v in V with $W(u, v) > W_a^i(u, v)$, an edge $u \xrightarrow{e} v$ with weight $w_a^{i+1}(e) = W(u, v) - k$ is introduced in E^i. Let $r : V \rightarrow \mathbf{Z}$ be a solution for problem KDP on G. If for every edge $u \xrightarrow{e} v \in E_a^i$, r satisfies*

$$r(v) - r(u) + w_a^i(e) \geq 0, \tag{23}$$

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

$$r(v) - r(u) + w_a^{i+1}(e) \geq 0. \tag{24}$$

PROOF. Let $u \xrightarrow{e} v \in E_a^i \cap E_a^{i+1}$. In this case, we have $w_a^{i+1}(e) = w_a^i(e)$, and Inequality (24) follows immediately from Inequality (23).

Now let $u \xrightarrow{e} v \in E_a^{i+1} - E_a^i$. By construction, we have $W(u, v) > W_a^i(u, v)$. Therefore, by the definition of $W_r(u, v)$ and Inequality (23), we have

$$\begin{aligned} W_r(u, v) &= W(u, v) + r(v) - r(u) \\ &> W_a^i(u, v) + r(v) - r(u) \\ &= W_{ar}^i(u, v) \\ &\geq 0, \end{aligned} \tag{25}$$

where $W^i_{ar}(u, v)$ denotes the delay count of the shortest path from u to v in the retimed auxiliary graph G^i_a. Since r is a solution to problem KDP, Inequality (25) implies that

$$W_r(u, v) \geq k.$$

(Otherwise some edge along the shortest path in G_r would contain fewer than k delays.) Therefore, for the edge $u \xrightarrow{e} v \in E^{i+1}_a$, we have

$$\begin{aligned} w^{i+1}_a(e) + r(v) - r(u) &= (W(u, v) - k) + r(v) - r(u) \\ &= (W(u, v) + r(v) - r(u)) - k \\ &= W_r(u, v) - k \\ &\geq 0. \end{aligned}$$

Thus, r satisfies Inequality (24). □

We can now prove that Algorithm AUX2 generates an auxiliary graph G_a that captures necessary conditions for problem KDP as shortest-paths constraints.

THEOREM 11. *In $O(V^3E + V^4\lg V)$ time, Algorithm AUX2 correctly generates an auxiliary graph $G_a = \langle V_a, E_a, w_a \rangle$ such that a retiming function $r : V \to \mathbf{Z}$ solves problem KDP only if for every edge $u \xrightarrow{e} v \in E_a$, we have*

$$r(v) - r(u) + w_a(e) \geq 0. \tag{26}$$

PROOF. The auxiliary graph computed in lines 1–10 during the first iteration of Algorithm AUX2 is identical to the auxiliary graph G_a computed by Algorithm AUX1. From Theorem 9 it follows that any retiming r that solves problem KDP satisfies the single-source shortest-paths constraints on G_a. From Lemma 10 we infer that any solution of problem KDP also satisfies the shortest-paths constraints in the augmented auxiliary graph G_a computed by Algorithm AUX2. Therefore, Algorithm AUX2 is correct.

Line 1 of Algorithm AUX2 completes in $O(V^2E)$ time. The **repeat** loop in lines 2–18 iterates $O(V^2)$ times, since there are at most $O(V^2)$ vertex pairs to include in the set $\mathcal{Q}$. In the body of the **repeat** loop, the generation of G_a takes $O(V^2)$ time, the all-pairs shortest-paths computation takes $O(VE + V^2\lg V)$ time, and the identification of new vertex pairs requires $O(V^2)$ time. Therefore, the overall running time of Algorithm AUX2 is $O(V^2E) + O(V^3E + V^4\lg V) + O(V^4) = O(V^3E + V^4\lg V)$. □

6. PATHS WITH ZERO DELAYS

In addition to paths with minimum delay requirements, it is possible to identify paths that have zero delays in every retimed CDFG that satisfies

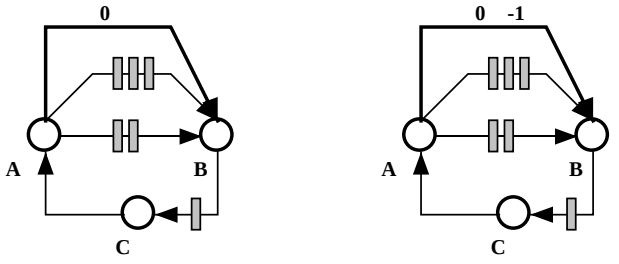


Fig. 8. Identifying paths with zero delays for $k = 2$. In Part (a) the bold edge $A \rightarrow B$ ensures that every path from A to B has at least 2 delays. In Part (b) the bold edge has been reweighted to ensure that no delays are left on $B \rightarrow C \rightarrow A \rightarrow B$.

Relation (6). In this section we explain how these paths arise. We also describe a polynomial-time algorithm that captures these necessary conditions along with the ones we described in Theorem 11 as a single-source shortest-paths problem on an appropriately weighted auxiliary graph.

The example in Figure 8 describes the derivation of necessary conditions on paths with zero delays. For $k = 2$, Lemma 5 holds for the vertex pair A, B of the graph G in Figure 8(a). The bold edge $A \rightarrow B$ with weight 0 imposes a lower bound of 2 on the delay count of every path $A \rightsquigarrow B$ in G. In this case, the delay of the edge $B \rightarrow C$ must be zero because the shortest directed path $B \rightarrow C \rightarrow A \rightarrow B$ in G has only one delay. To impose this condition as a shortest-paths constraint, the weight of the bold edge in part (b) is changed to $-W(B, A) = -1$.

In general, if $0 < W(u, v) + W_a(v, u) < k$ for some vertex pair u, v in V, then no retiming can achieve $W_r(u, v) \geq k$. Consequently, every path $u \rightsquigarrow v$ in a retimed graph that solves problem KDP must be free of delays. This condition can be ensured by introducing into the auxiliary graph G_a an edge $v \xrightarrow{e} u$ with weight $w_a(e) = -W(u, v)$. As the connectivity of the auxiliary graph G_a increases, more opportunities are created for discovering additional necessary conditions.

Algorithm AUX3 in Figure 9 generates the augmented auxiliary graph G_a. This algorithm is an adaptation of Algorithm AUX2 that introduces constraints for paths with zero and nonzero delays. The edges corresponding to the delay-free paths are introduced in lines 12–14 and are detected in lines 22–23. The variable $\mathcal{U}$ is used to temporarily store the vertex pairs in each iteration that give rise to these constraints.

The following lemma proves that the constraints introduced in each iteration of Algorithm AUX3 are necessary for problem KDP.

LEMMA 12. *Let $G = \langle V, E, w \rangle$ be a given CDFG, and let k be a given positive integer. Let $G_a^i = \langle V_a, E_a^i, w_a^i \rangle$ be an auxiliary graph generated by Algorithm AUX3 after i iterations of the* ***repeat*** *loop. Moreover, let $G_a^{i+1} =$*

```
AUX3(G, k)
 1 Q ← LB(G, k)
 2 R ← ∅
 3 repeat
 4       V_a ← V
 5       E_a ← E
 6       for each vertex pair u, v in Q
 7           do E_a ← E_a ∪ {u → v}
 8       for each edge u -e-> v in E_a
 9           do if e ∈ E
10                 then w_a(e) ← w(e)
11                 else w_a(e) ← W(u, v) − k
12       for each vertex pair u, v in R
13           do E_a ← E_a ∪ {v → u}
14              w_a(e) ← −W(u, v)
15       T ← ∅
16       U ← ∅
17       Compute all quantities W_a(u, v) using an all-pairs shortest paths algorithm
18       if there exists a directed cycle in G_a with negative edge weight
19           then return FAIL
20       for every vertex pair u, v in V with W(u, v) > W_a(u, v)
21           do T ← T ∪ {(u, v)}
22       for every vertex pair u, v in V with 0 < W(u, v) + W_a(v, u) < k
23           do U ← U ∪ {(u, v)}
24       Q ← Q ∪ T
25       R ← R ∪ U
26 until T = ∅ and U = ∅
27 return G_a
```

Fig. 9. Algorithm AUX3 for generating the augmented auxiliary graph G_a with constraints on paths with zero and nonzero delay counts.

$\langle V_a, E_a^{i+1}, w_a^{i+1}\rangle$ be the auxiliary graph generated after $i + 1$ iterations by augmenting E_a^i as follows: For every vertex pair u, v in V with $W(u, v) + W_a^i(v, u) < k$, an edge $u \xrightarrow{e} v$ with weight $w_a^{i+1}(e) = -W(u, v)$ is introduced in E^i. Let $r : V \to \mathbf{Z}$ be a solution for problem KDP on G. If for every edge $u \xrightarrow{e} v \in E_a^i$, r satisfies

$$r(v) - r(u) + w_a^i(e) \geq 0, \tag{27}$$

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

$$r(v) - r(u) + w_a^{i+1}(e) \geq 0. \tag{28}$$

PROOF. If $u \xrightarrow{e} v \in E_a^i \cap E_a^{i+1}$, we have $w_a^{i+1}(e) = w_a^i(e)$, and Inequality (28) follows immediately from Inequality (27).

Now consider an edge $v \xrightarrow{e} u \in E_a^{i+1} - E_a^i$. For the vertex pair $u, v \in V$, we have $W(u, v) + W_a^i(v, u) < k$ by the construction of E_a^{i+1}. Moreover, from Inequality (27) we infer that $W_{ar}^i(v, u) \geq 0$. Therefore,

$$\begin{aligned} W_r(u, v) &\leq W_r(u, v) + W_{ar}^i(v, u) \\ &= (W(u, v) + r(v) - r(u)) + (W_a^i(v, u) + r(u) - r(v)) \\ &= W(u, v) + W_a^i(v, u) \\ &< k. \end{aligned} \tag{29}$$

Since r solves problem KDP, it satisfies Relation (6). Therefore, from Inequality (29) it follows that $W_r(u, v) = 0$. (If $W_r(u, v) > 0$, then some edge along the shortest path from u to v in E contains fewer than k delays.) Thus, for the edge $v \xrightarrow{e} u \in E_a^{i+1}$, we have

$$\begin{aligned} w^{i+1}(e) + r(u) - r(v) &= -W(u, v) + r(u) - r(v) \\ &= -W_r(u, v) \\ &= 0. \end{aligned}$$

Therefore, r satisfies Inequality (28). □

It is now straightforward to show that Algorithm AUX3 generates an augmented auxiliary graph G_a that captures necessary conditions for the feasibility of Problem KDP as single-source shortest-paths constraints. These constraints can be generated and solved in polynomially many steps.

THEOREM 13. *In $O(V^3E + V^4 \lg V)$ time, Algorithm AUX3 correctly generates an auxiliary augmented graph $G_a = \langle V_a, E_a, w_a \rangle$ such that a retiming function $r : V \to \mathbf{Z}$ solves problem KDP only if for every edge $u \xrightarrow{e} v \in E_a$, we have*

$$r(v) - r(u) + w_a(e) \geq 0. \tag{30}$$

PROOF. Similar to the proof of Theorem 11. □

7. A PRACTICAL BRANCH-AND-BOUND ALGORITHM

In this section we describe a practical branch-and-bound scheme for solving problem KDP. Our scheme relies on the necessary conditions derived in Sections 4, 5, and 6 to prune the search space and quickly discover a solution.

Figure 10 gives a pseudocode description of Algorithm KDP for problem KDP. After generating the auxiliary graph G_a, the algorithm searches for a

```
KDP(G, k)
1 G_a ← AUX3(G, k)
2 r ← BB(G_a, k)
3 return r
```

Fig. 10. Algorithm KDP for solving problem KDP.

```
BB(G_a, k)
 1 Compute r by a single-source shortest-paths computation on G_a
 2 if no solution r exists
 3   then return FAIL
 4 if there exists edge u -e-> v ∈ E such that 0 < w_r(e) < k
 5   then E'_a ← E_a ∪ { u -e'-> v }
 6        w_a(e') ← w(e) − k
 7        r ← BB(⟨V_a, E'_a, w_a⟩, k)
 8        if r = FAIL
 9          then E'_a ← E_a ∪ { v -e'-> u }
10               w_a(e') ← −w(e)
11               r ← BB(⟨V_a, E'_a, w_a⟩, k)
12               if r = FAIL
13                 then return FAIL
14                 else return r
15          else return r
16   else return r
```

Fig. 11. Pseudocode for procedure BB.

solution by recursively calling Procedure BB in Figure 11. This procedure first performs a single-source shortest-paths algorithm to compute a retiming function r that satisfies Inequality 6 in G_a. If the resulting r satisfies Relation 6, it returns r. Otherwise, for some edge $u \xrightarrow{e} v$ such that $1 \leq w_r(e) \leq k - 1$, it introduces a constraint edge $u \xrightarrow{e'} v$ with weight $w_a(e') = w(e) - k$ and calls itself on the augmented constraints set that ensures $w_r(e) \geq k$ (lines 5–7). If the recursive call returns a solution r, Procedure BB passes it to its caller. If the recursive call fails, however, Procedure BB introduces a constraint edge $v \xrightarrow{e'} u$ with weight $w_a(e') = -w(e)$ and invokes itself to compute a retiming function on the augmented constraints set that ensures $w_r(e) = 0$ (lines 9–11). If this call returns a solution, the procedure passes it to its caller, otherwise it fails. It is possible to improve the practical efficiency of Algorithm KDP by recomputing the auxiliary graph G_a each time a new constraint is introduced. The augmented G_a may reveal a solution relatively early, thus reducing the number of recursive calls.

In addition to verifying the feasibility of a given block-processing factor k, Algorithm KDP can be used to compute the maximum block-processing

Design	# cycles with original CDFG	$k_{\max}$	# cycles with optimized CDFG	Improvement (%)
Zivojnovic 1	960	4	600	37
Zivojnovic 2	1,200	4	660	45
Echo Canceler	1,215	3	840	31
Video Coder	22,500	2	13,500	40

Fig. 12. Performance improvements achievable by block-processing.

factor $k_{\max}$. This number equals the maximum number of delays that can be placed on any edge and is bounded from above by F. Thus, $k_{\max}$ can be determined by a binary search over the integers in the range $[1, F]$. The feasibility of each value is checked using Algorithm KDP.

8. EXPERIMENTAL RESULTS

We have developed two programs that use binary search to compute the optimal block-processing factor $k_{\max}$. In this section we present results from the application of these programs on real and synthetic DSP computations. The objectives of our experiments were to determine the performance improvements that can be achieved by block-processing computations, compare the efficiency of our two implementations, and evaluate the effectiveness of the necessary conditions we developed.

Our first program, called ILP, solves the integer linear programming formulation of problem KDP that we described in Section 3. It first generates the ILP constraints and then solves the integer program separately using CPLEX, a commercial solver for mixed-integer linear programs. Our second program, called BB, is an implementation of Algorithm KDP which relies on the necessary conditions from Sections 4, 5, and 6 to effectively prune the search space.

In order to explore the computational speedups achievable by block-processing, we applied our k-delay optimization to the computation data-flow graphs of four real DSP programs. Our test suite comprised an adaptive voice echo canceler, an adaptive video coder, and two examples from Zivojnovic et al. 1994]. The size of the CDFGs for the DSP programs ranged from 10 to 25 nodes.

The results of our speedup experiments are given in the table of Figure 12. These data were obtained for uniprocessor implementations of the four computations in our test suite. The first column in the table gives the name of each computation. The second column gives the number of cycles required to perform $k_{\max}$ iterations of the computation. This number is given by the product $k_{\max} \cdot (H + C)$, where H is the context-switch overhead, and C is the sum of computation times over all blocks in the CDFG. The third column in the table gives the maximum block-processing factor $k_{\max}$ that can be achieved by retiming the original CDFG. The fourth column gives the number of cycles required to perform the $k_{\max}$ iterations

on the retimed CDFG using block-processing. The last column in the table gives the relative performance gains achieved by block-processing with $k_{\max}$. For each CDFG, the improvement is given by the fraction

$$1 - \frac{\text{cycles with optimized CDFG}}{\text{cycles with original CDFG}}.$$

In our experiments, block-processing reduced the number of cycles required for each application by at least one third of the original cycle count. For Zivojnovic 2, the number of cycles required was cut almost in half. Estimates for the initiation time H and the computation time C were obtained using measurements on typical general purpose DSP processors such as TMS32020 and Motorola 56000.

In order to evaluate the efficiency of our implementations, we experimented with large synthetic graphs in addition to the real DSP programs. These synthetic graphs were generated using the `sprand` function of the random graph generator described in Cherkassky et al. [1993]. Given a vertex count V and an edge count E, `sprand` generates a graph by randomly placing E edges between V vertices. Edge weights are also assigned randomly from a specified range. The graphs we generated for our experiments had 10 to 300 vertices and 20 to 750 edges. All graphs were connected, and edge-weights were chosen uniformly in the range [0,5].

The results from the application of the two programs on our synthetic test suite are summarized in the table of Figure 13. The first three columns give the name and size of each graph. The fourth column gives the maximum block-processing factor $k_{\max}$ that can be achieved by retiming each graph. This maximum is computed by performing a binary search in the range of possible values for k. In each iteration of the binary search, the ILP formulation of problem KDP for the corresponding k is solved using CPLEX. The last two columns of the table give the CPU times required by our two programs to compute $k_{\max}$ and the corresponding retimed CDFG. Running times were obtained on a Sun UltraSparc 1 with 64MB of main memory.

Our results show that the running time of program ILP increases at a relatively fast rate with the size of the input CDFG. For graphs with up to 20 vertices, program ILP can compute a solution within a few minutes. For graphs with 50 vertices, however, it requires more than 1 hour of CPU time. During the binary search, solutions to feasible problems are computed relatively fast. The detection of infeasible problems is extremely time-consuming, however, because the ILP solver has no quick way to infer infeasibility and thus searches a large portion of the solution space.

The results of our experiment also suggest that our necessary conditions are very effective in pruning the search space. For each graph in our test suite, BB was faster than ILP by one to three orders of magnitude. Moreover, it can handle inputs of signficantly larger size than ILP. For example, program BB was almost 1,000 times faster than ILP on the graph

CDFG name	V	E	k_{max}	CPU time (sec)	
				ILP	BB
g10-20.1	10	20	4	1.2	0.0
g10-40.1	10	40	1	1.4	0.0
g20-30.1	20	30	4	193.5	0.4
g20-50.1	20	50	2	674.6	0.5
g30-50.1	30	50	5	1,591.2	1.9
g30-70.1	30	70	3	1,136.4	1.4
g40-60.1	40	60	5	1,825.3	6.3
g40-100.1	40	100	2	1,597.3	4.8
g50-80.1	50	80	7	3,544.8	7.1
g50-130.1	50	130	3	3,899.2	8.1
g70-100.1	70	100	11	4,378.0	14.3
g70-170.1	70	170	2	–	50.5
g100-250.1	100	250	2	7,983.5	232.2
g150-200.1	150	200	7	13,605.1	358.3
g150-400.1	150	400	2	20,619.6	733.6
g200-300.1	200	300	6	21,727.0	1,122.4
g200-450.1	200	450	2	–	1,926.7
g250-350.1	250	350	6	–	3,049.0
g250-650.1	250	650	1	–	142.3
g300-450.1	300	450	4	–	15,250.0
g300-750.1	300	750	2	–	9,971.0

Fig. 13. Running times (in CPU seconds) of programs ILP and BB when retiming random graphs to achieve the maximum block-processing factor k_{max}. Entries marked with "—" indicate running times exceeding 30,000 CPU seconds.

g30-70.1. Moreover, program BB computed a solution for the graph g70-170.1 within 50.5 seconds, whereas program ILP exceeded the 30,000 second timeout limit.

9. CONCLUSION AND FUTURE WORK

Block-processing speeds up the execution of computations by amortizing context-switching overheads over several data samples. In this paper we investigated the problem of improving the block-processing factor of DSP programs using the retiming transformation. We formulated the problem of computing a retiming that achieves a given block-processing factor k as an integer linear program. We then presented a set of necessary conditions for this problem that can be computed in polynomial time. By relying on these conditions, we designed a practical branch-and-bound scheme for computing a regular and linear block-processing of any given computation. In our experiments with real and synthetic CDFGs, our branch-and-bound program was orders of magnitude more efficient than a software implementation of the general integer linear programming approach.

An important question that remains open is whether our necessary conditions are also sufficient for the k-delay problem. So far, we have not

been able to prove their sufficiency. On the other hand, we have not discovered a situation in which they are feasible while the k-delay problem is infeasible. The main difficulty in proving or disproving the sufficiency of these conditions is that, for any given CDFG, the constructive procedure that generates the corresponding constraints can result in different sets of inequalities, depending on the order in which vertex pairs are considered. We suspect that this property could be used as the basis for proving that our necessary conditions are not sufficient.

An interesting direction for further investigation is the reduction of critical path length in conjunction with the maximization of the block-processing factor. Specifically, among all solutions of the k-delay problem we would like to compute the ones that minimize the total computation time between any two consecutive iterations in the computation. Our preliminary work suggests that it is possible to express critical path requirements in the form of constraint edges in the auxiliary graph G_a.

Future research in this area could explore the applicability of our techniques for compiling code on Very Long Instruction Word (VLIW) architectures. The main challenge with VLIW machines is to issue as many instructions as possible in the same clock cycle. By viewing the instructions in a given program as delay elements in a computation graph, one could model the compilation problem for VLIW architectures as a block-processing problem on a CDFG.

REFERENCES

BLAHUT, R. E. 1985. *Fast Algorithms for Digital Signal Processing*. Addison-Wesley, Reading, MA.

CHERKASSKY, B., GOLDBERG, A., AND RADZIK, T. 1993. Shortest paths algorithms:: theory and experimental evaluation. Tech. Rep. STAN-CS-93-1480. Stanford University, Stanford, CA.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

DE MICHELI, G. 1991. Synchronous logic synthesis: Algorithms for cycle-time optimization. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 10*, 1 (Jan. 1991), 63–73.

GIBBS, W. W. 1994. Software's chronic crisis. *Sci. Am.* (Sept. 1994), 86–95.

GUERRA, L., POTKONJAK, M., AND RABAEY, J. 1994. System-level design guidance using algorithm properties. In *Proceedings of the IEEE Workshop on VLSI Signal Processing VII*, IEEE Press, Piscataway, NJ, 73–82.

ISHII, A. T., LEISERSON, C. E., AND PAPAEFTHYMIOU, M. C. 1992. Optimizing two-phase, level-clocked circuitry. In *Proceedings of the 1992 Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, T. Knight and J. Savage, Eds. MIT Press, Cambridge, MA, 245–264.

KU, D. C. AND DE MICHELI, G. 1992. Relative scheduling under timing constraints: Algorithms for high-level synthesis of digital circuits. *IEEE Trans. CAD/ICAS 11*, 6 (June), 696–718.

KUNG, S. Y. 1988. *VLSI Array Processors*. Prentice-Hall, Englewood Cliffs, NJ.

LALGUDI, K. N. AND PAPAEFTHYMIOU, M. C. 1995. DELAY: An efficient tool for retiming with realistic delay modeling. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16, 1995), B. T. Preas, Ed. ACM Press, New York, NY, 304–309.

LALGUDI, K. N. AND PAPAEFTHYMIOU, M. C. 1997. Computing strictly-second shortest paths. *Inf. Process. Lett. 63*, 4, 177–181.

LEISERSON, C. E. AND SAXE, J. B. 1991. Retiming synchronous circuitry. *Algorithmica 6*, 1, 5–35.

LIAO, S., DEVADAS, S., KEUTZER, K., TJIANG, S., AND WANG, A. 1995. Storage assignment to decrease code size. *SIGPLAN Not. 30*, 6 (June 1995), 186–195.

LOCKYEAR, B. AND EBELING, C. 1993. The practical application of retiming to the design of high-performance systems. In *Proceedings of the International Conference on Computer-Aided Design* (ICCAD '93, Santa Clara, CA, Nov. 7–11), M. Lightner and J. A. G. Jess, Eds. IEEE Computer Society Press, Los Alamitos, CA, 288–295.

MALIK, S., SENTOVICH, E., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1990. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. In *Proceedings of the 23rd Annual Hawaii International Conference on System Sciences*, 397–406.

RITZ, S., PANKERT, M., ZIVOJNOVIC, V., AND MEYR, H. 1993. Optimum vectorization of scalable synchronous dataflow graphs. In *Proceedings of the International Conference on Application-Specific Array Processors* (Oct. 1993), 285–296.

WALKER, R. A. AND THOMAS, D. E. 1989. Behavioral transformations for algorithmic level IC design. *IEEE Trans. CAD/ICAS 8*, 10 (Oct.), 1115–1127.

ZIVOJNOVIC, V., RITZ, S., AND MEYR, H. 1994. Retiming of DSP programs for optimum vectorization. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing 2*, 19–22.

Received: June 1997; accepted: July 1998