Journal of Signal Processing Systems (2019) 91:21–32
https://doi.org/10.1007/s11265-018-1415-2

Monotonic Optimization of Dataflow Buffer Sizes

Martijn Hendriks[1,2] · Hadi Alizadeh Ara[2] · Marc Geilen[2] · Twan Basten[1,2] · Ruben Guerra Marin[3,4] · Rob de Jong[3] · Steven van der Vlugt[3,4]

Received: 3 April 2017 / Revised: 19 March 2018 / Accepted: 27 September 2018 / Published online: 23 October 2018

Abstract
Many high data-rate video-processing applications are subject to a trade-off between throughput and the sizes of buffers in the system (the storage distribution). These applications have strict requirements with respect to throughput as this directly relates to the functional correctness. Furthermore, the size of the storage distribution relates to resource usage which should be minimized in many practical cases. The computation kernels of high data-rate video-processing applications can often be specified by cyclo-static dataflow graphs. We therefore study the problem of minimization of the total (weighted) size of the storage distribution under a throughput constraint for cyclo-static dataflow graphs. By combining ideas from the area of monotonic optimization with the causal dependency analysis from a state-of-the-art storage optimization approach, we create an algorithm that scales better than the state-of-the-art approach. Our algorithm can provide a solution and a bound on the suboptimality of this solution at any time, and it iteratively improves this until the optimal solution is found. We evaluate our algorithm using several models from the literature, and on models of a high data-rate video-processing application from the healthcare domain. Our experiments show performance increases up to several orders of magnitude.

Keywords Monotonic optimization · Cyclo-static dataflow · Throughput · Buffer size

1 Introduction

Many high data-rate video processing applications have strict requirements on throughput as it affects the (visual) quality. It may even affect safety, as is the case for the medical video-processing application in the image-guided therapy domain that has motivated our work. Often, these types of applications are subject to a trade-off between throughput and the sizes of the buffers (the *storage distribution* from now on). Since buffer space uses expensive or scarce resources, one of the key design questions for these applications is how to minimize the storage distribution without violating the throughput constraint. We approach this problem using model-based design, and model and analyze the application using the cyclo-static dataflow (CSDF) formalism [5]. This formalism is a member of the dataflow family [6] and is suitable for modeling a broad class of streaming, parallel applications with cyclically changing behavior and finite buffers such as our video-processing applications. This model-based approach has the advantage that the analysis of the model usually is much faster than experimentation on a prototype. For instance, our driver case has an FPGA as implementation target. The hardware implementation step in the process takes several hours. Using a storage-distribution minimization algorithm with an analysis step that takes several hours clearly is infeasible for even small search spaces.

Efficient methods exist to compute the throughput of a CSDF graph with a given storage distribution. The problem that we consider in this article is to minimize the size of the storage distribution under a throughput constraint. In general, this problem is NP-hard [8], and we therefore present an *anytime* algorithm. The algorithm first tries to quickly find an initial storage distribution that realizes the throughput constraint. Then it iteratively improves the storage distribution. During this process, the algorithm

✉ Martijn Hendriks
martijn.hendriks@tno.nl

1 ESI (TNO), Eindhoven, The Netherlands

2 Eindhoven University of Technology, Eindhoven, The Netherlands

3 Philips Healthcare, Best, The Netherlands

4 TOPIC Embedded Systems, Best, The Netherlands

provides an upper bound on the difference between the size of the currently best storage distribution and the size of the (unknown) minimal storage distribution. This can be a useful feature because if the user has no patience to wait for a real minimum storage distribution (finding one can take long due to the NP-hardness), he can terminate the algorithm and still have a feasible storage distribution and an estimation of the quality of this solution.

Contribution In this work we combine principles from monotonic optimization [12, 13] and the concept of knee points of [7] with the causal dependency analysis from [10, 11]. This results in an algorithm that minimizes the storage distribution in CSDF graphs under a throughput constraint. This algorithm scales better than the state-of-the-art approach of [10, 11]. Our experiments show that the performance may be improved by several orders of magnitude. Furthermore, it is an anytime algorithm which is able to present at any moment (after the initialization phase and if it exists) a storage distribution that satisfies the throughput constraint and a bound on the suboptimality of this best solution so far. A secondary contribution is an elaboration of the concept of knee points that has been introduced in [7]. Knee points play a crucial role in our algorithm.

Related work Closely related work that adresses the problem of this article, optimization of the storage distribution under a throughput constraint for CSDF graphs, is the work of [3, 4, 10, 11, 15]. In [15], a fast approximation algorithm is proposed that over-estimates the size of the required storage distribution with an unknown factor. The work of [10, 11], on the other hand, presents an exact solution to a slightly more general problem than the problem of this article: [10, 11] compute the whole trade-off space, which can then be used to solve our problem. The work of [3, 4] is closely related to [15] and provides an approximate solution based on a relaxation of an integer-linear program. Our work is complementary to [10, 11] as it can be regarded as a fast heuristic to significantly prune the search space after which the exact method of [10, 11] is used to obtain the final solution. Our method can also be regarded as a domain-specific specialization of the domain-independent and generic monotonic optimization framework of [12, 13]. This framework to solve non-convex, but monotonic optimization problems, has succesfully been applied in the area of wireless communications [9, 14, 16], and we now introduce it in the dataflow domain. The key difference with the generic outer-polyblock approximation algorithm of [12, 13] is that we bound the optimal solutions from both the inside and the outside. This is similar to the approach of [7], which uses a constraint solver to build an approximation of the Pareto front of a multi-criteria optimization problem, using monotonicity implicitly. We use the concept of knee points of [7] to select a new point in the search space to explore, instead of using a binary search to compute the upper-boundary projection in the outer-polyblock approximation algorithm of [12, 13]. Furthermore, the fact that we limit the scope of the approach to CSDF allows us to take advantage of domain-specific properties and analysis methods, i.e., the causal dependency analysis of [10, 11], to make the search more efficient.

2 Explanation of the Approach

Let us consider a 2-dimensional buffer sizing problem modeled in CSDF with buffers b_1 and b_2. A *storage distribution* is a function $\delta : \{b_1, b_2\} \rightarrow \mathbb{N} \cup \{\infty\}$ that gives the size of each buffer, measured in dataflow tokens. The set $\mathcal{S}$ of *feasible* storage distributions (those storage distributions that satisfy the throughput constraint) is shown in Fig. 1 (it appears to have a smooth boundary due to the scale of the figure). Of course, we do not know $\mathcal{S}$ beforehand, and computing whether a storage distribution is feasible by invoking a CSDF throughput analysis can be time consuming. Let us assume in this example that both buffers contain items of equal size, and that we want to minimize the size of the total storage distribution, i.e., the sum of the two buffer sizes.

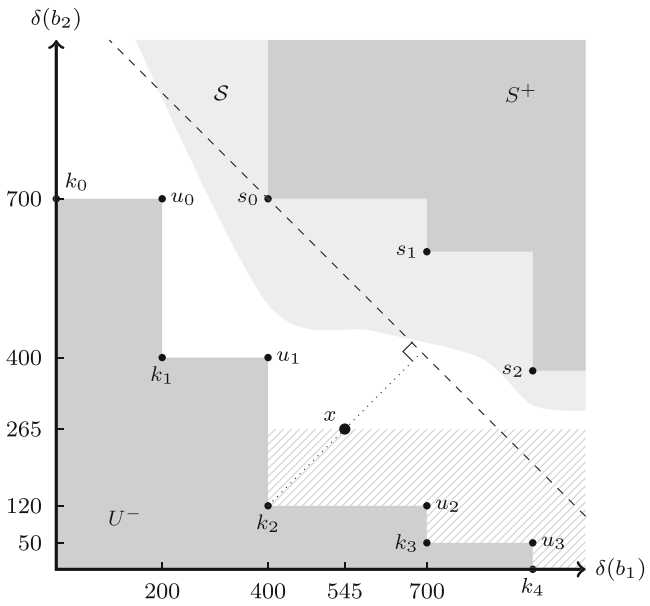

Figure 1 Sketch of the approach.

Our approach is centered around monotonicity of throughput and buffer sizes in the CSDF formalism: increasing buffer sizes will not decrease throughput. This monotonicity allows us to efficiently represent and bound the search space. Suppose that we have analyzed four points in the storage distribution space $u_0 - u_3$ and that these resulted in a less than required throughput (they are infeasible). Figure 1 shows these points, and because of monotonicity we know that the area U^- does not contain any feasible points. In a similar way, we can build a view of feasible points. Suppose that $s_0 - s_2$ are points that we have analyzed and that have been shown to be feasible. The area S^+ then only contains feasible points because of monotonicity. Because the total size of the storage distribution is also monotone, the best solution in S^+ is one of the points $s_0 - s_2$, namely s_0 with size $400+700 = 1100$. This representation of U^- and S^+ and the exploitation of monotonicity is closely related to the area of monotonic optimization [12, 13].

The second concept that we use are the special *knee points* $k_0 - k_4$ [7]. These are induced by $u_0 - u_3$ and are local minima in the sense that any feasible solution will have a size greater than the size of one of the knees (because of monotonicity). In particular, the knee points with the smallest size provide a lower bound on the size of any feasible solution. In this example, k_2 has the smallest size of $400 + 120 = 520$. This means that the optimal solution (at best the point $(401, 121)$) has a size of at least 522 and at most 1100 (size of s_0). We thus say that the maximal error Δ equals $1100 - 522 = 578$. Given the state of knowledge determined by the set of feasible points $s_0 - s_2$ and the set of infeasible points $u_0 - u_3$, we select a new point to check for feasibility. This selection process is based on the knees and on the hyperplane of points with size equal to the best solution so far (the dashed line through s_0): We select a point $x = (545, 265)$ halfway on the line segment between the hyperplane and a knee with the smallest size (k_2 in this example). Intuitively, this is the area where most can be gained. By choosing the point x halfway between k_2 and the hyperplane, we apply a multi-dimensional binary search. In the case that x is feasible, we extend S^+ with the area to the right and above x. Furthermore, x improves on s_0 and the maximal error Δ now equals $810 - 522 = 288$, which is approximately half of the previous maximal error.

The third ingredient in our approach is the causal dependency analysis of [11], which we use to bound the search space even further. Throughput analysis of a CSDF graph can, in addition to the throughput, also provide the channels that have a so-called *storage dependency*. Intuitively, a channel creates a storage dependency if the progress of the data processing depends on freeing storage space in the buffer associated to that channel. From [11] it follows that throughput can only increase if the size of at least one channel with a storage dependency is increased. Now, suppose that the analysis of x shows that it is infeasible and that only buffer b_2 has a storage dependency. This means that not increasing the size of buffer b_2 from point x will never result in a feasible point. We therefore can extend the infeasible point $x = (545, 265)$ to $(\infty, 265)$, resulting in a significant reduction in the search space: the area filled with the pattern is added to U^-. The knee points k_2, k_3 and k_4 are removed, and a new knee point $k_5 = (400, 265)$ is added. This makes $k_1 = (400, 200)$ the knee point with the smallest size in the new situation, and this reduces the maximal error Δ from 578 to $1100 - (401 + 201) = 498$.

Iteration of these steps reduces the gap between U^- and S^+ and also the maximal error Δ, and will eventually find the best feasible storage distribution.

3 Cyclo-Static Dataflow Graphs

We briefly repeat existing definitions and results concerning CSDF graphs based on [11]. We let $\mathbb{N}_{0,\infty}$ denote the set $\mathbb{N} \cup \{0, \infty\}$. Let P be a set of *ports*, and let *rate* be a function that assigns a finite sequence $(r_1, r_2, \ldots, r_n)$ of rates in $\mathbb{N}$ to each port (lengths of these sequences may differ among the ports). An *actor* is a tuple (I, O, T) consisting of $I \subseteq P$ input ports, $O \subseteq P$ output ports with $I \cap O = \emptyset$, and of $T = (t_1, t_2, \ldots, t_n)$ execution times.

Definition 1 (**CSDF graph**) A CSDF graph is a tuple (A, C) of a set of *actors* A, and a set of *channels* $C \subseteq P \times P$ such that (i) $(p, q) \in C$ implies that p is an output port and that q is an input port, and (ii) all ports are connected to exactly one channel.

The initial state of a CSDF graph is determined by the initial token distribution, which assigns a number (possibly 0) of initial tokens to each channel.

Consider, for instance, the CSDF graph in Fig. 2. It shows the graph of a sample-rate converter [1]. The nodes represent the actors and the edges represent the channels (ports are not explicitly shown). The execution

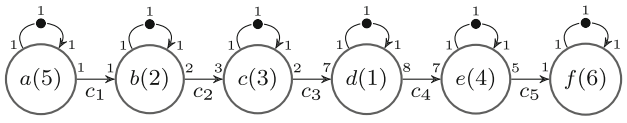

Figure 2 The CSDF graph for a sample-rate converter.

time sequence is shown in the actor nodes. The numbers at the beginning and end of the edges show the rates. Note that all execution time and rate sequences have length one in this example (effectively turning this model into a Synchronous Dataflow (SDF) graph). Each self-loop has a single initial token which limits auto-concurrency of the actors.

Channels have an unbounded storage space in the semantics. As it is commonly done in literature, we model finite buffer space of channels $C_{buf} \subseteq C$ by adding for each channel $(p, q) \in C_{buf}$ from actor $a \in A$ to actor $b \in A$ a new channel (p_δ, q_δ) from b to a where p_δ and q_δ are new ports with $rate(p_\delta) = rate(q)$ and $rate(q_\delta) = rate(p)$. The number of initial tokens on (p_δ, q_δ) equals the storage space of the channel (p, q) minus the number of initial tokens on (p, q).

Definition 2 (**Storage distribution**) Let (A, C) be a CSDF graph and let $C_{buf} \subseteq C$ be a set of *buffered channels*. A *storage distribution* for C_{buf} is a function $\delta : C_{buf} \to \mathbb{N}_{0,\infty}$. We let (A_δ, C_δ) denote the CSDF graph with the additional channels that realize the storage constraints,[1] and assume that it is strongly connected.[2]

Consider the CSDF graph (A, C) of Fig. 2, and let $C_{buf} = \{c_1, c_2, \ldots, c_5\}$. Consider the storage distribution δ such that $\delta(c_i) = b_i$ for $1 \leq i \leq 5$. Figure 3 shows (A_δ, C_δ). The dotted edges represent the special channels that model the storage constraints. For instance, the constraint on the channel c_1 is modeled by the dotted channel from b to a with b_1 initial tokens on it. This models that there can be at most b_1 tokens in c_1.

Let δ and δ' be two storage distributions. We say that $\delta \preceq \delta'$ if and only if $\delta(c) \leq \delta'(c)$ for all $c \in C_{buf}$. Since the tokens in different channels may represent data of different size we introduce a *cost* function $w : C_{buf} \to \mathbb{N}$ that assigns a (non-zero) cost to each buffer channel. The *cost* of a storage distribution δ, denoted by $|\delta|$, then is $\sum_{c \in C_{buf}} w(c) \cdot \delta(c)$.

Throughput of a CSDF graph is a well-defined concept and algorithms exist to compute it (see [11]). For (A, C) we let $\xi(A, C) \in \mathbb{R}$ denote its throughput. A throughput constraint $tc \in \mathbb{R}$ on (A, C) gives a lower bound on the necessary throughput. We say that a storage distribution δ is *feasible* if and only if the throughput constraint is satisfied, i.e., $\xi(A_\delta, C_\delta) \geq tc$. A useful property of the CSDF formalism is that throughput, and thereby feasibility of storage distributions, is monotone with respect to buffer sizes.

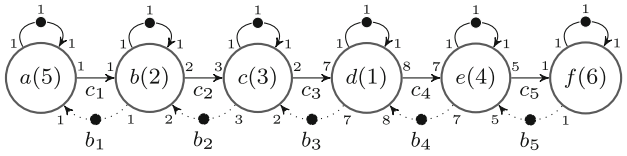

Figure 3 The CSDF graph for a sample rate converter with additional storage constraints.

Lemma 1 [11] *Let (A, C) be a CSDF graph, and let δ, δ' be storage distributions such that $\delta' \preceq \delta$. Then $\xi(A_{\delta'}, C_{\delta'}) \leq \xi(A_\delta, C_\delta)$.*

A key contribution of [11] is the concept of *storage dependencies* and we refer the reader to [11] for the precise definition. Analysis of the self-timed execution of (A_δ, C_δ) is used to compute a set $Dep_\delta \subseteq C_{buf}$ of buffered channels that have a storage dependency. The throughput of a CSDF graph cannot be increased without increasing the capacity of at least one such a channel. In Section 2, we have sketched how this can be used to reduce the search space (cutting off the area filled with the pattern in Fig. 1). The following lemma and corollary formalize this.

Lemma 2 [11] *Let (A, C) be a CSDF graph and let δ and δ' be a storage distributions such that $\delta \preceq \delta'$ and $\xi(A_\delta, C_\delta) < \xi(A'_\delta, C'_\delta)$. Then there is a channel $c \in Dep_\delta$ such that $\delta(c) < \delta'(c)$.*

The following corollaries follow from these lemmas. The first one states that increasing buffers that have no storage dependency does not increase the throughput.

Corollary 1 *Let (A, C) be a CSDF graph and let δ be a storage distribution. For every storage distribution δ' holds: if for all $c \in Dep_\delta$ we have that $\delta'(c) \leq \delta(c)$, then $\xi(A_{\delta'}, C_{\delta'}) \leq \xi(A_\delta, C_\delta)$.*

Proof Consider a storage distribution δ' with $\delta'(c) \leq \delta(c)$ for all $c \in Dep_\delta$, let $t = \xi(A_\delta, C_\delta)$, and let $t' = \xi(A_{\delta'}, C_{\delta'})$. Assume that $t' > t$. We define

$$\delta''(c) = \begin{cases} \delta(c) & if\ c \in Dep_\delta \\ max(\delta(c), \delta'(c)) & otherwise \end{cases}$$

Let $t'' = \xi(A_{\delta''}, C_{\delta''})$. We have that $\delta' \preceq \delta''$, and therefore by Lemma 1 that $t' \leq t''$, and thus $t < t''$. Furthermore, we have that $\delta \preceq \delta''$. We can then apply Lemma 2 to conclude that there is a channel $c \in Dep_\delta$ such that $\delta(c) < \delta''(c)$. This

[1] If $\delta(c) = \infty$, then we can model this by removing the buffer edge for channel c.

[2] Buffer sizing for non-strongly-connected graphs can be done for the strongly-connected components.

contradicts the definition of δ'', and therefore we conclude that $t' \not\succ t$. □

The second corollary informally states that if we have an infeasible storage distribution without buffered channels that have a storage dependency, then no feasible storage distribution exists.

Corollary 2 *Let (A, C) be a CSDF graph and let δ be an infeasible storage distribution. If $Dep_\delta = \emptyset$, then no feasible storage distribution exists.*

Proof Suppose that a feasible storage distribution δ' exists, which necessarily has a greater throughput than δ. Define δ'' as $\delta''(c) = max(\delta(c), \delta'(c))$ for all $c \in C_{buf}$. Then $\delta' \preceq \delta''$ and therefore $\xi(A_{\delta'}, C_{\delta'}) \leq \xi(A_{\delta''}, C_{\delta''})$ by Lemma 1. Thus, $\xi(A_\delta, C_\delta) < \xi(A_{\delta''}, C_{\delta''})$. We also have that $\delta \preceq \delta''$ and therefore we can apply Lemma 2 to conclude that there must be a channel $c \in Dep_\delta$ such that $\delta(c) < \delta''(c)$. This contradicts that $Dep_\delta = \emptyset$. □

In the remainder of this article, we assume that we have access to a CSDF analysis function *analyze* that, given a CSDF graph (A, C), a set of buffered channels $C_{buf} \subseteq C$ and a storage distribution δ for C_{buf} returns a tuple $(\xi(A_\delta, C_\delta), Dep_\delta)$ where $Dep_\delta \subseteq C_{buf}$ is the set of channels with a storage dependency in (A_δ, C_δ). The problem that we consider is the following:

Definition 3 (**Optimization problem**) Given are a CSDF graph (A, C), buffered channels $C_{buf} \subseteq C$, a throughput constraint tc, and a cost function $w : C_{buf} \to \mathbb{N}$. The buffer optimization problem is to find a feasible storage distribution δ such that for any other feasible storage distribution δ' holds that $|\delta| \leq |\delta'|$.

Consider the CSDF graph (A, C) from Fig. 2 again. It has throughput $1.04 \cdot 10^{-3}$, and we use this as the throughput constraint for (A_δ, C_δ) shown in Fig. 3. Then $\delta = \{c_1 \mapsto 1, c_2 \mapsto 4, c_3 \mapsto 8, c_4 \mapsto 14, c_5 \mapsto 5\}$ has a throughput of $9.19 \cdot 10^{-4}$ and thus is not feasible. The causal dependency analysis gives us that $Dep_\delta = \{c_1, c_2, c_3, c_5\}$. From this we can conclude that every buffer valuation $\{c_1 \mapsto 1, c_2 \mapsto 4, c_3 \mapsto 8, c_4 \mapsto x, c_5 \mapsto 5\}$ with $x \in \mathbb{N}_{0,\infty}$ is not feasible. That is, increasing only buffer c_4 in size and none of the other buffers, will not lead to a better throughput. We can use this to reduce the search space as explained in Section 2.

In the next section, we formally explain the framework that we use to approach the optimization problem of Definition 3.

4 Monotonic Optimization

We assume the problem setting of Definition 3 and let $d = |C_{buf}|$. A storage distribution δ is represented by a point $x = (x_1, x_2, \ldots, x_d)$ in $\mathbb{N}^d_{0,\infty}$ given a bijection $index : C_{buf} \to \{1, 2, \ldots, d\}$ as follows: $x_{index(c)} = \delta(c)$ for all $c \in C_{buf}$. The cost of x, denoted by $|x|$ is defined as $|\delta|$. We say that an x is feasible if and only if δ is feasible. We use the abbreviation $x[i \leftarrow v]$ for the point $(x_1, x_2, \ldots, x_{i-1}, v, x_{i+1}, \ldots, x_d)$, i.e., the i-th element of x is replaced by v. In the remainder of this article, we assume that the sets D, E, K, S, and U all are subsets of $\mathbb{N}^d_{0,\infty}$, and that k, s, u, q, x, x', y, y', z, and z' all are elements of $\mathbb{N}^d_{0,\infty}$. The set complement operation is assumed to act with respect to the universe $\mathbb{N}^d_{0,\infty}$, i.e., $\overline{U} = \mathbb{N}^d_{0,\infty} \setminus U$.

The forward $(+)$ and backward $(-)$ cones of x and their strict versions $(++, =)$ are defined as follows:

$$\begin{aligned} x^+ &= \{x' \mid \forall_{1 \leq i \leq d}\ x'_i \geq x_i\} \\ x^{++} &= \{x' \mid \forall_{1 \leq i \leq d}\ x'_i > x_i\} \\ x^- &= \{x' \mid \forall_{1 \leq i \leq d}\ x'_i \leq x_i\} \\ x^= &= \{x' \mid \forall_{1 \leq i \leq d}\ x'_i < x_i\} \end{aligned} \tag{1}$$

If U is closed under $-$, then its complement is closed under $+$, and vice versa:

$$\begin{aligned} \overline{U^-}^+ &= \overline{U^-} \\ \overline{U^=}^+ &= \overline{U^=} \\ \overline{U^+}^- &= \overline{U^+} \\ \overline{U^{++}}^- &= \overline{U^{++}} \end{aligned} \tag{2}$$

The backward (forward) cone of some set U is the union of the backward (forward) cones of the elements of U. A point $x \in U$ is *maximal* in U if and only if for all $y \in U, y \neq x$ holds that $x \notin y^-$. Similarly, $x \in U$ is *minimal* if and only if for all $y \in U, y \neq x$ holds that $x \notin y^+$. We use this definition for the maximal elements of sets as follows: $max(U) = \{x \in U \mid x \text{ is maximal in } U\}$, and $min(U) = \{x \in U \mid x \text{ is minimal in } U\}$. A set U is *maximal* if and only if $max(U) = U$ and *minimal* if and only if $min(U) = U$. The cones of a finite set U can be represented by a unique subset of itself containing only maximal or minimal elements:

$$\begin{aligned} U^+ &= min(U)^+ \\ U^- &= max(U)^- \\ U^{++} &= min(U)^{++} \\ U^= &= max(U)^= \end{aligned} \tag{3}$$

In some other contexts, these sets of minimal or maximal points are called Pareto points. In Fig. 1, for instance, the backward cone of the set of maximal points $U = \{u_0, u_1, u_2, u_3\}$ is shown, as well as the forward cone of the set of minimal points $S = \{s_0, s_1, s_2\}$. Note that the strict cones exclude the points on the boundaries, and that the knee points of U, $\{k_0, \ldots, k_4\}$, are elements of U^-. In fact, the knee points can be regarded as the duals of the points in $max(U)$. The following definition is an alternative characterization of knee points as originally introduced in [7].

Definition 4 (**Knee points**) The set of points (knees) of a finite set $U \subseteq \mathbb{N}^d_{0,\infty}$, denoted by $knee(U)$ is the set $knee(U) = min(\overline{U^=})$.

The following corollary states an equivalent characterization of knee points, which we use further below.

Corollary 3 *$K = knee(U)$ if and only if K is minimal and $K^+ = \overline{U^=}$, i.e., $K^+ \cup U^= = \mathbb{N}^d_{0,\infty} \wedge K^+ \cap U^= = \emptyset$.*

Proof ($\Rightarrow$) Straightforward with Eqs. 2 and 3.

($\Leftarrow$) We have that $K^+ = \overline{U^=}$. Thus, $min(K^+) = min(\overline{U^=})$. Because clearly $min(K^+) = min(K)$, we have by the assumption that K is minimal that $min(K^+) = K$ and thus $K = min(\overline{U^=}) = knee(U)$. □

The next corollary states that under a specific condition, the union of the strict forward cones of the knees K is equal to the complement of the union of the backward cones of the unsat points U. The extra condition on U is needed because the hyperplane lower boundaries (i.e., the points in which at least one dimension equals 0) by definition are not part of K^{++} and hence should be included in U^-. This extra condition is true for at least every set U for which holds that $\{\infty[k \leftarrow 0] \mid 1 \leq k \leq d\} \subseteq U^-$. In the context of buffer sizing this makes perfect sense, as it models the situation in which storage distributions that have buffers of size 0 are infeasible.

Corollary 4 *Let $K = knee(U)$ and let U be such that for every point $x \notin U^-$, there is some point $y \in U^-$ such that $y \in x^=$. Then $K^{++} = \overline{U^-}$.*

Proof ($K^{++} \subseteq \overline{U^-}$) Let $x \in K^{++}$. We need to show that $x \notin U^-$. Then by definition of K, $x \in \overline{U^=}^{++}$. Hence, there is some $y \in \overline{U^=}$ such that $y \in x^=$. We have that $y \in \overline{U^=}$, so $y \notin U^=$ and therefore, for any $u \in U$, $y \notin u^=$ (1). Assume towards a contradiction that there is some $z \in U$ such that $x \in z^-$. From $y \in x^=$ and $x \in z^-$ it follows that $y \in z^=$, which contradicts (1).

($\overline{U^-} \subseteq K^{++}$) Let $x \in \overline{U^-}$, i.e. $x \notin U^-$. We need to show now that $x \in K^{++}$, i.e., that $x \in \overline{U^=}^{++}$. Let z_0 be such that $z_0 \in U^-$ and $z_0 \in x^=$. Here we use the additional assumption to ensure it exists. If $z_0 \notin U^=$, we have some $z \notin U^=$ with $z \in x^=$, thus $x \in \overline{U^=}^{++}$ and we are done. Otherwise, $z_0 \in U^=$, $x \notin U^-$ and $z_0 \in x^=$. Hence, there must be some z_1 such that $z_1 \neq z_0$, $z_0 \in z_1^-$, $z_1 \in U^-$, and $z_1 \in x^=$. Again, if $z_1 \notin U^=$ we are done. Otherwise we continue similarly with z_2, z_3, etcetera. Eventually, we must find z_k such that $z_k \notin U^=$, because $|x| - |z_k|$ decreases in every step and cannot go negative. □

Finally, the following corollary formalizes that the knee points are part of the backward cone of the generating set (see, e.g., Fig. 1).

Corollary 5 *Let $K = knee(U)$. If $U \neq \emptyset$, then $K \subseteq U^-$.*

Proof Suppose that $U \neq \emptyset$ and $K \not\subseteq U^-$. Then we have that $\neg\forall_{k\in K}\exists_{u\in U} k \in u^-$. I.e., $\exists_{k\in K}\forall_{u\in U} k \notin u^-$. Consider such a k. We distinguish two cases. First, $k = 0^d$. Because we have assumed that $U \neq \emptyset$, there is at least one $u \in U$, and clearly $k \in u^-$, which is a contradiction. Hence, $K \subseteq U^-$. Second, $k \neq 0^d$. Define the point x such that $x_i = max(k_i - 1, 0)$ for $1 \leq i \leq d$. In that case, $x \neq k$ and clearly $x \notin k^+$ and also $x \notin K^+$ because K consists of minimal points. Furthermore, $x \notin u^-$ since $k \notin u^-$ for all $u \in U$ by assumption. Therefore also $x \notin U^=$. Thus, we have that $x \notin K^+$ and $x \notin U^=$, which constradicts Corollary 3. Hence, $K \subseteq U^-$. □

As our algorithm sketched in Section 2 progresses, it can happen that a point x that just has been analyzed is infeasible. This has impact on the existing knee points of the infeasible set U that are in the backward cone of x. Figure 4 shows an example of such a situation with $U = \{u_0, \ldots, u_3\}$. The knee points k_1, k_2 and k_3 are in x^-. Adding x to the infeasible set U has the consequence that k_1, k_2 and k_3 are not knee points anymore. A point k in x^- has d extensions to x, namely the points $\{k[i \leftarrow x_i] | 1 \leq i \leq d\}$ where each time the value of one dimension of k is replaced by the corresponding value of x. The knee points k_1, k_2 and k_3 in Fig. 4 are replaced by the points e_{12} and e_{31}, which are the minimal points of the extensions of the knees to x.

This is formalized by the following two lemmas and Theorem 1 on knee generation.

Lemma 3 (**Extension completeness**) *If $k \in x^-$, then $k^+ \setminus x^= = \{k[i \leftarrow x_i] \mid 1 \leq i \leq d\}^+$.*

Proof First, let $z \in \{k[i \leftarrow x_i] \mid 1 \leq i \leq d\}^+$. Then $z \in k[i \leftarrow x_i]^+$ for some dimension i. We let $q = k[i \leftarrow x_i]$

and thus have $z \in q^+$. By definition we have $z_j \geq q_j$ for all $1 \leq j \leq d$. We also have that $q_j \geq k_j$ for $1 \leq j \neq i \leq d$ and $q_i \geq x_i \geq k_i$. Thus, $z_j \geq k_j$ which is to say that $z \in k^+$, but also $z_i \geq x_i$, which implies that $z \notin x^=$.

Second, let $z \in k^+ \setminus x^=$. This means that $z_j \geq k_j$ for all $1 \leq j \leq d$ and some i exists such that $z_i \geq x_i$. Let $q = k[i \leftarrow x_i]$. Then we have $q_j = k_j$ for all $1 \leq j \neq i \leq d$ and $q_i = x_i \geq k_i$. Thus, $z_j \geq q_j$ for $1 \leq j \leq d$, and $z \in q^+$. Therefore, $z \in \{k[i \leftarrow x_i] \mid 1 \leq i \leq d\}^+$. □

The second lemma is a generalization of Lemma 3 and is the basis for our knee computation method. It formalizes (and generalizes to an arbitrary number of dimensions) the situation sketched in Fig. 4.

Lemma 4 *Consider a point x and a set K such that $x \in K^+$. Then $K^+ \setminus x^= = ((K \setminus D) \cup min(E))^+$ where $D = K \cap x^-$ and $E = \{y[i \leftarrow x_i] \mid y \in D \wedge 1 \leq i \leq d\}$.*

Proof Note that this proof treats the term $min(E)^+$ first as E^+ and as a last step reasons why this is valid.

First suppose that $z \in K^+ \setminus x^=$. Then $z \in k^+$ for some $k \in K$. We distinguish two cases: (a) $k \notin x^-$, and (b) $k \in x^-$. In case (a) k is not removed from K through D, thus $k \in (K \setminus D) \cup E$. Therefore, $z \in ((K \setminus D) \cup E)^+$. In case (b) k is removed as part of D. However, the extensions of D are added through set E, and by Lemma 3 we know that $z \in ((K \setminus D) \cup E)^+$.

Second, suppose that $z \in ((K \setminus D) \cup E)^+$. Then (a) some $k \in K \setminus D$ exists such that $z \in k^+$, or (b) some $k \in K$ and i exist such that $z \in k[i \leftarrow x_i]^+$. In case (a) we thus have that $z \in k^+$, $k \in K$ and $k \notin x^-$ and thus $z \notin x^-$. Therefore, $z \in K^+ \setminus x^=$. In case (b) we have that $z \in k[i \leftarrow x_i]^+$, $k \in K$ and $k \in x^-$. Clearly also $z \in k^+$ because $x_i \geq k_i$ and thus $z \in K^+$. Furthermore, $z \notin x^=$ because $z_i \geq x_i$.

Now we have proven that $K^+ \setminus x^= = ((K \setminus D) \cup E)^+$. It is clear that $(U_1 \cup U_2)^+ = U_1^+ \cup U_2^+$, and the combination with $min(E)^+ = E^+$ yields the desired result. □

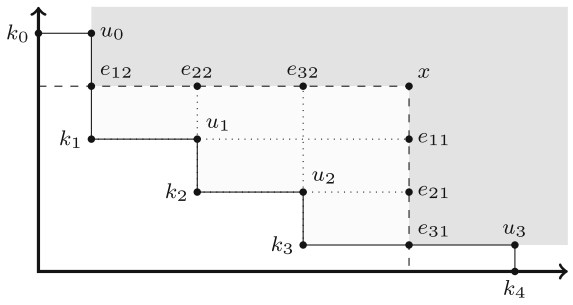

Figure 4 An example of knee generation.

The following theorem is fundamental to our method as it provides a means to compute knee points efficiently.

Theorem 1 (**Knee generation**) *Consider a set U and let $K = knee(U)$. Then $K' = knee(U \cup \{x\})$ can be computed as follows:*

- *if $x \in U^-$, then $K' = K$, and otherwise:*
- *$K' = (K \setminus D) \cup min(E)$ where $D = K \cap x^-$ and $E = \{y[i \leftarrow x_i] \mid y \in D \wedge 1 \leq i \leq d\}$*

Proof The case for $x \in U^-$ follows straightforwardly because then $(U \cup \{x\})^- = U^-$. Now consider the case $x \notin U^-$. Using Corollary 3, we have to prove that:

1. $K'^+ \cap (U \cup \{x\})^= = \emptyset$,
2. $K'^+ \cup (U \cup \{x\})^= = \mathbb{N}^d_{0,\infty}$, and
3. $K' = min(K')$.

Item 1 reduces to $((K \setminus D) \cup min(E))^+ \cap (U \cup \{x\})^= = \emptyset$. From Lemma 4, we have that $((K \setminus D) \cup min(E))^+ = K^+ \setminus x^=$ and we thus have to show that:

$$\begin{array}{ll} (K^+ \setminus x^=) \cap (U \cup \{x\})^= = \emptyset & \Leftrightarrow \\ (K^+ \setminus x^=) \cap (U^= \cup x^=) = \emptyset & \Leftrightarrow \\ (K^+ \setminus x^=) \cap U^= = \emptyset \wedge (K^+ \setminus x^=) \cap x^= = \emptyset & \end{array}$$

From our assumption that $K = knee(U)$, we have via Corollary 3 that $K^+ \cap U^= = \emptyset$, which proves the first part of the conjunction. The second part is straightforward from definitions of set operations.

For item 2, we have to show – using the same reduction as above – that $(K^+ \setminus x^=) \cup (U \cup \{x\})^= = \mathbb{N}^d_{0,\infty}$. Using our assumption that $K = knee(U)$, we can derive in a similar way as above that this indeed is the case.

For item 3, notice that $K \setminus D$ is minimal because K is minimal. Furthermore, $min(E)$ is minimal by definition. Note that for $e \in min(E)$ holds that $e \in x^-$. Therefore, such an extension e does not have a point in $K \setminus D$ in its backward cone, which is to say that no point in $K \setminus D$ has e in its forward cone. Furthermore, no extension $e \in min(E)$ has a point in $K \setminus D$ in its forward cone, because then K would not have been minimal. Therefore, K' is minimal: $K' = min(K')$. □

The following theorem states that knees of a set of infeasible storage distributions give a (non-tight) lower bound on the cost of a feasible storage distribution.

Theorem 2 (**Lower bound feasible cost**) *Consider a set U of infeasible points such that $\{\infty^d[k \leftarrow 0] \mid 1 \leq k \leq d\} \subseteq$*

U^- and a feasible point x. Then $|x| \geq min\{|k + 1^d| \mid k \in knee(U)\}$.

Proof Let $K = knee(U)$. First, we have that for every point $x \notin U^-$ holds that every buffer has at least size 1, because we assume that $\{\infty^d[k \leftarrow 0] \mid 1 \leq k \leq d\} \subseteq U^-$. We can apply Corollary 4 and have that $K^{++} = \overline{U^-}$.

Every point in U^- is infeasible by Lemma 1 and our representation of the storage distributions. Thus, a feasible point is part of $\overline{U^-}$, which thus equals K^{++}. Hence, there is some knee $k \in K$ such that $k \in x^=$ and $|x| \geq |k + 1^d|$. Thus $|x| \geq min\{|k + 1^d| \mid k \in knee(U)\}$. □

In the next section, we apply the mechanism explained in this section to the optimization problem.

5 Optimization Algorithm

Algorithm 1 solves the minimization problem of Definition 3. There are four local variables: the set U contains infeasible points, K contains the knees of U, x is the point that represents the storage distribution that is analyzed, and S contains the feasible points (also see Fig. 1). A key invariant throughout the algorithm is that $K = knee(U)$ holds.

Algorithm 1 Monotonic Buffer Sizing (MBS).

1: $U = \{\infty^d[k \leftarrow 0] \mid 1 \leq k \leq d\}$ // initial infeasible set
2: $K = \{0^d\}$ // initial knee set
3: $x \leftarrow init()$ // initial storage distribution; see, e.g., [2]
4: $(t, Dep) \leftarrow analyze(x)$
5: **while** $t < tc$ **do**
6: $\quad (U, K) \leftarrow handleInfeasible(U, K, x, Dep)$
7: $\quad$ **for** $c \in Dep$ **do**
8: $\quad\quad x \leftarrow x[index(c) \leftarrow 2 \cdot x_{index(c)}]$
9: $\quad$ **end for**
10: $\quad (t, Dep) \leftarrow analyze(x)$
11: **end while**
12: $S \leftarrow \{x\}$
13: $x \leftarrow select(U, K, S, \ldots)$
14: **while** $x \neq \perp$ **do**
15: $\quad (t, Dep) \leftarrow analyze(x)$
16: $\quad$ **if** $t < tc$ **then**
17: $\quad\quad (U, K) \leftarrow handleInfeasible(U, K, x, Dep)$
18: $\quad$ **else**
19: $\quad\quad S \leftarrow min(S \cup \{x\})$
20: $\quad$ **end if**
21: $\quad x \leftarrow select(U, K, S, \ldots)$
22: **end while**
23: Call Alg. 1 of [11] with $\{k+1^d \mid k \in K \wedge \forall_{s \in S} |k+1^d| < |s|\}$ as the set of unexplored storage distributions

Algorithm 2 *handleInfeasible*(U, K, x, Dep)

1: **if** $Dep = \emptyset$ **then**
2: $\quad$ **error:** throughput requirement cannot be met
3: **end if**
4: $y \leftarrow x$
5: **for** $c \in C_{buf} \setminus Dep$ **do**
6: $\quad y \leftarrow y[index(c) \leftarrow \infty]$
7: **end for**
8: $K \leftarrow updateKnees(U, K, y)$ // Theorem 1
9: $U \leftarrow max(U \cup \{y\})$

Algorithm 3 *select*(U, K, S, t)

1: Values of following three variables are remembered between invocations:
2: STATIC $k \leftarrow \perp$ // the knee that is being explored
3: STATIC $m \leftarrow 1$
4: STATIC $v \leftarrow \perp$
5: **if** $k \neq \perp \wedge t < tc$ **then**
6: $\quad m \leftarrow m \cdot 2$
7: $\quad y \leftarrow \lfloor k + m \cdot v \rfloor$
8: $\quad$ **if** $y \notin S^+$ **then**
9: $\quad\quad$ **return** y
10: $\quad$ **end if**
11: **end if**
12: $m \leftarrow 1$
13: $k \leftarrow$ some $l \in K$ s.t. $|l| \leq |l'|$ for every $l' \in K$
14: $d \leftarrow min\{|s| \mid s \in S\}$
15: $v \leftarrow \frac{1}{2} \cdot \frac{d - k \cdot w}{w \cdot w} \cdot w$
16: $y \leftarrow \lfloor k + v \rfloor$
17: **if** $y \in U^- \vee y \in S^+$ **then**
18: $\quad$ **return** $\perp$
19: **else**
20: $\quad$ **return** y
21: **end if**

The algorithm consists of three phases. First, lines 1–11 form the initialization phase in which a first feasible solution is created, starting from the initial storage distribution that gives each buffer a minimal necessary size for deadlock-free execution (see, e.g., [2]). The while-loop iteratively doubles the buffer sizes of the buffers with a storage dependency until it finds a feasible solution. The *handleInfeasible* function, which is defined in Algorithm 2, updates the set of infeasible points U and the knees K for every infeasible point that is encountered. Note that this function reports an error if an infeasible point is encountered with no storage dependencies, which implies that there is no feasible storage distribution. Also note that lines 5 – 7 of *handleInfeasible* apply the additional pruning of the search space using the causal dependency information. This is formalized in the following lemma.

Lemma 5 *Let x be infeasible and let Dep be the storage dependencies. Define $y = (y_1, y_2, \ldots, y_d)$ as follows:*

$$y_i = \begin{cases} \infty \ if \ index^{-1}(i) \notin Dep \\ x_i \ \ otherwise \end{cases}$$

for all $1 \leq i \leq d$. Then every point in y^- is also infeasible.

Proof Let x represent the infeasible storage distribution δ and consider a storage distribution δ' that is represented by a point in y^-. By definition, $\delta'(c) \leq \delta(c)$ for all $c \in Dep$. Thus, by Corollary 1 we know that the throughput for δ' is not greater than the throughput for δ. Therefore, δ' is also infeasible. □

The second phase of Algorithm 1, lines 12–22, form the optimization phase which starts after a first feasible point is found by the initialization phase. This phase iteratively chooses a new point x to analyze and calls *handleInfeasible* if the point is infeasible, and otherwise adds it to S. The selection function is flexible (hence the ... notation in the list of parameters). A requirement is that implementations either return a point that is neither in U^- nor in S^+, or $\perp$ (in case it cannot find a good point to explore). Our current implementation, shown in Algorithm 3, selects points on the line through a knee with minimal cost, and the closest point on the cost hyperplane of the best solution so far (see Fig. 1). It starts halfway the line segment (lines 12 – 21), and doubles the distance to the knee as long as the point is infeasible and not part of S^+ to prune as much of the space as possible (lines 5 – 11). When the function cannot select a point in the unexplored space between U^- and S^+, it returns $\perp$. This will eventually happen because we work in $\mathbb{N}^d_{0,\infty}$.

The third phase of Algorithm 1, the final enumeration phase, starts in line 23. It calls the algorithm from [11] with the knee points that have the potential of leading to a feasible point with a cost smaller than the cost of the best point so far. This still is necessary because, in general, a *select* function may have left some points between U^- and S^+ that may give a better solution than the best one we have found so far.

Invariant 1 *At lines 10 and 21 of Algorithm 1 it holds that (i) U^- only contains infeasible points, (ii) S^+ only contains feasible points, and (iii) $K = knee(U)$.*

Proof Straightforward using Lemma 5 and Theorem 1. □

Theorem 3 *Algorithm 1 solves the optimization problem of Definition 3.*

Proof The initialization phase in fact is a greedy version of the algorithm in [11] that takes exponentially growing steps in the direction of the storage dependencies. Therefore, if a feasible point exists, then the initialization phase will find one. The conclusion that no feasible solution exists for an empty set of storage dependencies is valid according to Corollary 2. The optimization phase extends the sets U, K and S until the selection function returns $\perp$. This happens eventually, because the extension part in lines 5 – 11 eventually will find that $y \in S$ in which case a new point is selected in lines 12 – 21. If U and S are sufficiently close, then, due to the fact that we have a discrete search space, we cannot find a point in between. Furthermore, if we find a point in between and process it, then either S comes closer to U (in case of a feasible point), or U comes closer to S (in case of an infeasible point). Invariant 1 ensures that U, K and S are built in a proper way. Finally, the algorithm from [11] is invoked with the still promising knee points as a starting point. These are good starting points because any feasible point must be part of K^{++}, and by correctness of the algorithm in [11] we thus solve the problem of Definition 3. □

Note that the algorithm can also be interrupted; in that case the optimization and enumeration phases are stopped or skipped, and the best result so far x and the maximal cost error $\Delta = |x| - min\{|k + 1^d| \mid k \in K\}$ are returned (see Theorem 2). This interruption logic is not shown in Algorithm 1 for readability.

6 Experimental Evaluation

We compare with the state-of-the-art approach of [11] that computes the full buffer-size – throughput trade-off space. Since the optimization problem of this article (see Definition 3) is a slightly more restricted problem, [11] can be used to solve it. In this section, we compare the approaches, because no other reference algorithm exists. We therefore set the throughput constraint to the throughput of the self-timed execution of the graph, which is the highest throughput possible. The approach of [11] terminates as soon as it has analyzed the storage distributions up to and including this self-timed throughput. Earlier results in the algorithm on trade-off points with lower throughput are needed for this, so [11] needs all earlier computations to reach the final trade-off point of the self-timed throughput. This makes the approaches comparable for the case in which we optimize the size of the storage distribution under the constraint that the throughput is maximal, i.e., equal to the self-timed throughput.

We use the following models from the SDF3 website [1]: an MP3 playback application, an H.263 decoder, a sample-rate converter, and a satellite receiver. These are all SDF models (i.e., CSDF models with constant rates and execution times). The models MRF-32, MRF-64 and MRF-128 are models from a real-life image processing application, a multi-resolution filter with different input sizes, from the healthcare domain. The MRF models are all rather complex CSDF models with many different rates for a number of actors due to data dependencies. The cost function that we use gives each buffer a weight of one in each model.

Table 1 shows the results. For each model, we list whether it is an SDF model or a CSDF model, the number of actors $|A|$ and the number of sized buffers $|C_{buf}|$. For the state-of-the-art approach SGB08 [11], we then give the size of the obtained storage distribution $|\delta|$, the number of throughput analysis calls and a running time for a given multiplication factor of the step size n. This multiplication factor is an approximation mechanism, i.e., a factor greater than one trades computation effort against accuracy of the obtained result: the obtained storage distribution may not be minimal anymore. The models for the MP3 playback application, the H.263 decoder and the multi-resolution filter are not analyzable within reasonable time with $n = 1$ (indicated by - in the table). The first two models even require many throughput calls with $n = 10$. For our approach, Monotonic Buffer Sizing (MBS; Algorithm 1), we also show the obtained storage distribution, the number of throughput analysis calls and a running time. The Δ column indicates the absolute maximal error in the storage distribution that we tolerate for the optimization process. This number is derived from the multiplication factor n and the model properties. For instance, $n = 10$ for the MP3 playback application allows an optimization error in the storage distribution of 18 (there are two buffers, and each buffer allows an error of at most $10 - 1$ storage units assuming a step size of one; see [11]). Algorithm 1 needs 27 throughput analysis calls to obtain a solution with at most this error, and this is reported in the table.

To compare the performance of the approaches we primarily use the number of throughput analysis calls, and not the running time. The reason is that our prototype implementation has been written in JAVA and invokes an external SDF3 executable for each throughput analysis, which has a significant overhead. The approach of SGB08 has, on the other hand, been fully integrated in a single SDF3 executable. In both approaches the throughput analysis dominates the overall running time, and therefore we use the number of throughput analysis calls as a measure of performance to abstract from implementation details. The running times are, nevertheless, also shown in Table 1, and we expect that the values for a fully integrated MBS implementation will be smaller.

The results show that both approaches obtain the same size of the storage distribution when an optimal solution is expected (i.e, for a step size of one and a Δ of zero). When a suboptimal solution with a bounded error is accepted, then both approaches result in storage distributions of similar size, which is as expected. The MP3 playback application and the H.263 decoder models are difficult for SGB08, but easy for our approach. We believe that this is caused by explicit visitation of a large part of the search space by SGB08 to achieve the optimal throughput, whereas our approach takes large steps and skips analysis of many intermediate storage distributions. Both methods show similar performance for the models of the sample-rate converter and satellite receiver. These models differ from

Table 1 Experimental results.

					SGB08 [11]				MBS (Algorithm1)			
	SDF	CSDF	$\|A\|$	$\|C_{buf}\|$	n	$\|\delta\|$	# calls	Time (s)	Δ	$\|\delta\|$	# calls	Time (s)
MP3 playback	✓		3	2	1	–	–	–	0	2898	34	5
					10	2907	1944724	8702	18	2906	27	4
H.263 decoder	✓		4	3	1	–	–	–	0	8006	46	13
					10	8023	1223707	10751	27	8029	40	12
Sample rate	✓		6	5	1	34	16	0	0	34	14	1
Satellite	✓		22	26	1	1544	38	2	0	1544	32	4
MRF-32		✓	21	4	1	–	–	–	0	500	106	31
					5	503	211	18	16	505	46	10
MRF-64		✓	21	4	1	–	–	–	0	985	115	272
					5	993	749	1002	16	993	57	106
MRF-128		✓	21	4	1	–	–	–	0	1962	149	4974
					10	1968	506	8144	36	1965	51	1112

the MP3 playback and the H.263 decoder models in the fact that the initial storage distribution that can be calculated by a fast analysis is close to the optimal storage distribution with the required throughput. The results also show that our approach scales better than SGB08 for the rather complex CSDF models of the image-processing application from the healthcare domain.

7 Conclusions

We have introduced an algorithm to optimize the storage distribution size given a throughput constraint for CSDF graphs. This algorithm is based on three ingredients: (i) the causal dependency analysis from [10, 11], (ii) principles from the area of monotonic optimization [12, 13], and (iii) the concept of knee points introduced in [7]. A useful property of our algorithm is that it can provide some feasible storage distribution and an upper bound on the size difference with an optimal feasible storage distribution any time after the initialization phase. The experimental results show that our approach is better suited for buffer minimization under a throughput constraint than (the more general) approach of [10, 11] in the sense that solutions can be obtained with fewer throughput analysis calls.

Our algorithm can in principle be applied to other models of computation and other optimization problems than buffer sizing in CSDF by removing or adapting the parts with respect to causal dependency analysis (i.e., line 23 in Algorithm 1 and lines 5 – 7 in Algorithm 2). The only requirement is that the function that defines the feasibility is monotone with respect to the optimization parameters (in our case the throughput is monotone with respect to the buffer sizes; see Lemma 1). The resulting approach then would be closely related to the generic monotonic optimization frameworks as presented in [12, 13].

Acknowledgements This research was supported by the ARTEMIS joint undertaking under grant agreement no 621439 (ALMARVI).

References

1. Sdf3 website. http://www.es.ele.tue.nl/sdf3/.
2. Adé, M., Lauwereins, R., Peperstraete, J.A. (1997). Data memory minimisation for synchronous data flow graphs emulated on dsp-fpga targets. In *Proceedings of the 34th annual design automation conference*. New York: ACM.
3. Benazouz, M., Marchetti, O., Munier-Kordon, A., Michel, T. (2010). A new method for minimizing buffer sizes for cyclo-static dataflow graphs. In *8th IEEE workshop on embedded systems for real-time multimedia*.
4. Benazouz, M., & Munier-Kordon, A. (2013). Cyclo-static dataflow phases scheduling optimization for buffer sizes minimization. In *Proceedings of the 16th international workshop on software and compilers for embedded systems*. New York: ACM.
5. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J. (1996). Cycle-static dataflow. *IEEE Transactions on Signal Processing*, *44*(2), 397–408.
6. Lee, E.A., & Parks, T.M. (1995). Dataflow process networks. *Proceedings of the IEEE*, *83*(5), 773–801.
7. Legriel, J., Le Guernic, C., Cotton, S., Maler, O. (2010). Approximating the Pareto front of multi-criteria optimization problems. In *Proceedings of the 16th international conference on tools and algorithms for the construction and analysis of systems*. Berlin: Springer.
8. Moreira, O., Basten, T., Geilen, M., Stuijk, S. (2010). Buffer sizing for rate-optimal single-rate data-flow scheduling revisited. *IEEE Transactions on Computers*, *59*(2), 188–201.
9. Qian, L.P., Zhang, Y.J., Huang, J. (2009). Mapel: achieving global optimality for a non-convex wireless power control problem. *IEEE Transactions on Wireless Communications*, *8*(3), 1553–1563.
10. Stuijk, S., Geilen, M., Basten, T. (2006). Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proceedings of the 43rd annual design automation conference*. New York: ACM.
11. Stuijk, S., Geilen, M., Basten, T. (2008). Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *IEEE Transactions on Computers*, *57*(10), 1331–1345.
12. Tuy, H. (2000). Monotonic optimization: problems and solution approaches. *SIAM Journal on Optimization*, *11*(2), 464–491.
13. Tuy, H., Al-Khayyal, F., Thach, P. (2005). *Monotonic optimization: branch and cut methods*, (pp. 39–78). US: Springer.
14. Utschick, W., & Brehmer, J. (2012). Monotonic optimization framework for coordinated beamforming in multicell networks. *IEEE Transactions on Signal Processing*, *60*(4), 1899–1909.
15. Wiggers, M.H., Bekooij, M.J.G., Smit, G.J.M. (2007). Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *44th ACM/IEEE design automation conference*.
16. Xing, C., Ma, S., Zhou, Y. (2015). Matrix-monotonic optimization for mimo systems. *IEEE Transactions on Signal Processing*, *63*(2), 334–348.

Martijn Hendriks is a research fellow with ESI, TNO, the Netherlands. He holds an M.Sc. (2002) and Ph.D. (2006) in computing science from Radboud University Nijmegen. His current research interests include the modeling and analysis of embedded and cyber-physical systems, with an emphasis on performance engineering.

Hadi Alizadeh Ara has received his master's degree in Control Systems from the Tehran Polytechnic in 2013. He is currently a last year PhD candidate at the Eindhoven University of Technology in Electronic Systems group of the Electrical Engineering Department. His research interests include embedded systems, cyber-physical systems, model-based design, discrete event systems, (max,+) algebra, and trade-off analysis for realtime streaming applications. He has (co)authored publications on these topics.

Marc Geilen is an assistant professor in the Department of Electrical Engineering at Eindhoven University of Technology. He holds an MSc and a PhD from Eindhoven University of Technology. In 2010, he was a McKay Visiting Professor at the University of California, Berkeley. His research interests include modeling, simulation and programming of multimedia systems, formal models-of-computation, model-based design processes, multiprocessor systems-on-chip, networked embedded systems and cyber-physical systems, and multi-objective optimization and trade-off analysis. He is a member of IEEE. He has been involved with several national and international research projects and programs on the above topics with strong industrial connections. He has served on various TPCs and on organizing committees for several conferences including DATE as a topic chair and member of the executive committee.

Twan Basten is a Professor with the Department of Electrical Engineering, Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands, where he chairs the Electronic Systems group. He is also a Senior Research Fellow with ESI, TNO, Eindhoven. He received the M.Sc. and Ph.D. degrees in computing science from TU/e. His current research interests include the design of embedded and cyber-physical systems, dependable computing, and computational models.

Ruben Guerra Marin received his M.Sc. degree in Embedded Systems from the Eindhoven University of Technology in 2016. Since then he worked in several projects through Topic Embedded Systems, including Philips Healthcare where he was an FPGA engineer for the ARTEMIS ALMARVI project. Since 2018 he is working for TMC as a Hardware Designer, focusing mainly on FPGA design. His main areas of research interest include embedded systems, real-time systems, broadcasting, healthcare and airspace technology.

Rob de Jong joined Philips in 1985 and currently is a system designer for medical X-ray systems within the Image Guided Therapy department of Philips Healthcare. A large part of his work involves realtime embedded FPGA based image processing and control devices including model driven resource-usage and development-time optimization techniques.

Steven van der Vlugt received his master's degree in Electrical Engineering from Eindhoven University of Technology in 2014. Since then he has been working as consultant for Topic Embedded Systems. He was at Philips Healthcare until mid 2017 where he worked with FPGA HLS tooling for real-time image processing. In that time he also joined the ARTEMIS ALMARVI project as lead engineer and work package leader. His interests are especially with real-time embedded systems, heterogeneous computing, healthcare and image processing applications. He is currently on a project at ASML where he is working on new compute architectures for control applications.