



# Throughput and Memory Optimization for Parallel Implementations of Dataflow Networks Using Multi-Reader Buffers

Martin Letras  

Friedrich-Alexander Universität Erlangen-Nürnberg (FAU), Germany

Joachim Falk 

Friedrich-Alexander Universität Erlangen-Nürnberg (FAU), Germany

Jürgen Teich 

Friedrich-Alexander Universität Erlangen-Nürnberg (FAU), Germany

---

## Abstract

In this paper, we introduce the concept of Multi-Reader Buffers (MRBs) for high throughput and memory-efficient implementation of dataflow applications. Our work is motivated by the huge amount of data that needs to be processed and typically accessed in a FIFO manner, particularly in image and video processing applications. Here, multi-cast, fork, and merge operator implementations known today produce huge memory overheads by storing and communicating copies of the same data. As a remedy, we first introduce MRBs as buffers preserving FIFO semantics for a finite number of readers of the same data while storing each data item only once. Second, we present an approach for memory minimization of data flow networks by replacing all multi-cast actors and connected FIFOs with MRBs. Third, we present a Design Space Exploration approach to selectively replace multi-cast actors with MRBs in order to explore memory, throughput, and processor resource allocation tradeoffs. Our results show that the explored Pareto fronts of our approach improve the solution quality over a reference by 78% in average for six benchmark applications in terms of a hypervolume indicator.

**2012 ACM Subject Classification** Hardware → Static timing analysis

**Keywords and phrases** Dataflow, Memory Optimization, MPSoCs, Design Space Exploration

**Digital Object Identifier** 10.4230/OASICS.NG-RES.2023.6

**Funding** This work has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants 45098171 and 146371743.

## 1 Introduction

Generally, an image processing application consists of a graph of image processing filters, where each filter operates on its input and produces transformed image data at its outputs. Two important aspects must be considered in the design of efficient implementations of image processing applications on multicore architectures 1) the application's concurrency and exploitable parallelism and 2) its memory footprint. As imperative programming languages are a poor fit for developing concurrent applications, dataflow processing [4, 7, 17], which is naturally suited to expressing concurrency, is widely adopted to program modern Multi-Processor Systems-on-a-Chip (MPSoCs). Each filter of an image processing application can be modeled by an *actor* consuming and producing data. Each communication between two filters can be realized via a First In First Out (FIFO) buffer that needs to be mapped to a region in memory. Now, one constraint in dataflow modeling is that each FIFO buffer is exclusively written by one producer actor and read by one consumer actor. When multiple consumers require to read the same data, some dataflow modeling frameworks [12, 10, 5] propose to solve this issue by introducing so-called *multi-cast actors* that just read information



© Martin Letras, Joachim Falk, and Jürgen Teich;  
licensed under Creative Commons License CC-BY 4.0

Fourth Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2023).

Editors: Federico Terraneo and Daniele Cattaneo; Article No. 6; pp. 6:1–6:13

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and replicate it for a finite number of consumers. Accordingly, the introduction of multi-cast actors might negatively impact the *memory footprint* of image processing applications due to the data redundancy induced to respect dataflow semantics.

To avoid any memory overhead and as a first contribution, we introduce the concept of a *Multi-Reader Buffer (MRB)* preserving FIFO semantics for a finite number of readers of the same data while storing each data item only once. As a second contribution, we present an approach for memory minimization of dataflow networks by replacing each multi-cast actor and the FIFOs connected to it by a MRB. But whereas using MRBs instead of multi-cast actors reduces the memory footprint by removing data redundancy thus, delivering optimal memory footprint implementations, this transformation might negatively impact the throughput of a given application, as a bounded-length MRB will free space in its FIFO memory only once the *last* reader will have consumed the respective data. Accordingly, our third contribution is to explore the tradeoff between memory footprint minimization and throughput maximization by *selectively* replacing multi-cast actors by MRBs.

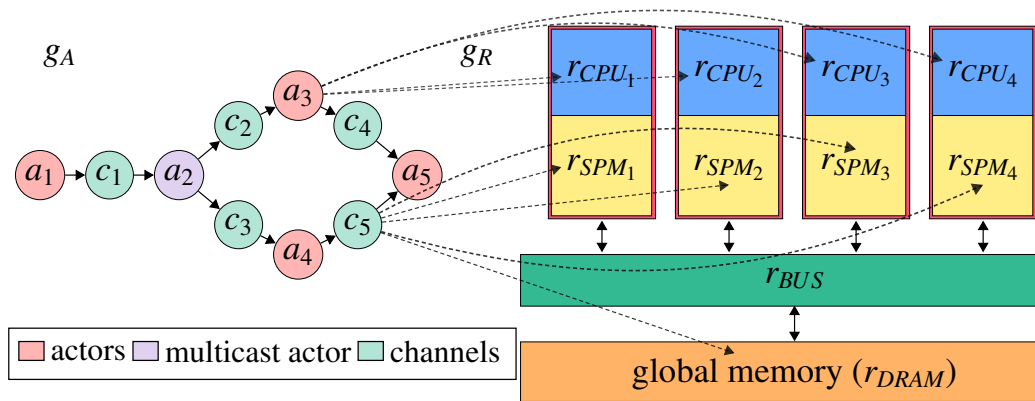
Here, we propose a multi-objective Design Space Exploration (DSE) for the mapping and scheduling of a data flow specification onto symmetric multi-processor target platforms with a global shared memory and each processor having an additional local scratchpad memory. In addition to memory footprint to be minimized and throughput to be maximized, we consider the number of allocated cores for each explored mapping and schedule as an additional cost metric. We subsequently compare our proposed DSE results named  $MRB_{\text{Explore}}$  to a *Reference* approach that only optimizes the mappings without introducing any MRBs. We also compare the DSE results  $MRB_{\text{Explore}}$  against an approach named  $MRB_{\text{Always}}$  in which all multi-cast actors and their adjacent FIFOs are replaced by MRBs. Our experiments show that  $MRB_{\text{Always}}$  is able to improve the quality of found solutions in terms of a hypervolume indicator by 67 % on average compared to a state-of-the-art reference approach. Moreover, our proposed approach  $MRB_{\text{Explore}}$  is shown to be able to find even better fronts of solutions by improving the hypervolume to 78 % on average against the approach *Reference*.

This paper is structured as follows: Section 2 presents the state-of-the-art. Section 3 presents the a formalization of the optimization problem. Then, Section 4 presents the semantics of our proposed MRB. Section 5 presents the DSE approach to selectively implement MRBs and the experimental results for six applications in terms of the quality of the found solution sets. Finally, Section 6 concludes this paper.

## 2 Related Work

Approaches for optimizing parallel implementation of applications specified as dataflow networks [17] perform multi-objective optimization of conflicting design objectives, e.g., throughput, number of allocated cores, and memory footprint. On the one hand, approaches such as [11, 7] optimize dataflow applications' throughput and the number of allocated cores in a given architecture. E.g., [7] proposed a clustering approach of static actors into a so-called cluster. Through the proposed clustering approach, the scheduling of connected static data flow sub-graphs can be coordinated to exploit the predictability and efficiency of the static data flow model. Moreover, clustering reduces scheduling overhead by reducing the number of checking guards of the actors composing a cluster, thus improving the throughput of applications. However, the previously presented approaches do not consider any memory footprint evaluation of implementations during DSE.

On the other hand, approaches for memory footprint minimization can be classified into two main categories: 1) approaches minimizing the size of FIFOs and 2) approaches implementing memory-reuse strategies that allow different FIFOs to be mapped into overlapping



■ **Figure 1** On the left, an application graph  $g_A$  consisting of actors connected by communication channels. On the right, a multi-core architecture that is modeled by an architecture graph  $g_R$ . Dashed lines represent mappings from actors to processors and channels to memories.

memory spaces or track individual token lifetimes to exploit memory footprint reductions over the execution of an application. In the first category, techniques such as FIFO sizing have been widely studied to reduce the memory footprint of Synchronous Dataflow (SDF) applications [15, 1]. Such approaches determine the minimal buffer size of an SDF application under throughput constraints. However, those approaches do not consider any memory-reuse strategy because each buffer is studied as a separate unit allocated in memory, and no shared memory address space is considered. In the second category, the approach presented in [6] derives overlapping memory allocations for individual tokens communicated during the execution of an SDF graph. As a requirement, the SDF graph has to be transformed into a single-rate SDF graph inducing a significant analysis overhead that leads to an approach ill-suited for usage within a DSE [6].

Apart from performing an agnostic memory footprint minimization, some approaches exploit the knowledge about the application and actor characteristics. For instance, dataflow frameworks [5, 18, 13] targeting image processing apply memory minimization strategies based on the behavior of a set of specialized actors performing operations like multi-cast, fork, and join of data. For instance, the employed memory minimization strategy described in [13] merges all outgoing buffers of a multi-cast actor by replacing them with a broadcast FIFO that supports a single writer but multiple readers [13]. However, this implementation is only able to handle single rate dataflow applications. Moreover, no other design objectives apart from memory footprint are explored. In this paper, we propose a holistic approach that considers not only the minimization of memory footprint but also the mapping and scheduling of communication channels and actors onto an MPSoC as well as the number of allocated CPUs as exploration objectives.

### 3 Fundamentals

Mapping problems of applications to embedded systems, i.e., multi-core target architectures, are often described by a *specification graph* [2, 16] composed of (i) an *application graph*, (ii) an *architecture graph*, and (iii) *mappings* connecting the application and the architecture graphs.

### 3.1 Application Graph

An application is modeled as a bipartite graph of actors and channels. First, we formalize actors.

► **Definition 1** (Actor [8]). *An actor is a tuple  $a = (I, O, \psi, \kappa, \tau)$  containing a set of actor input ports  $I$  and actor output ports  $O$ . The function  $\psi : O \rightarrow \mathbb{N}$  assigns the production rate to all output ports and the function  $\kappa : I \rightarrow \mathbb{N}$  assigns the consumption rate on each input port per firing. Finally,  $\tau \in \mathbb{R}$  represents the actor's execution time.*

Next, we model a given data flow specification of communicating actors by a bipartite application graph:

► **Definition 2** (Application Graph). *The application graph  $g_A = (A, C, E, \delta, \gamma, \varphi)$  is a bipartite graph with its vertices partitioned into a set of actors  $A$  and a set of communication channels  $C$ . Each channel represents a FIFO buffer. The set of directed edges  $E \subseteq (A.O \times C) \cup (C \times A.I)$  connects actors and channels. The delay function  $\delta : C \rightarrow \mathbb{N}_0$ , capacity function  $\gamma : C \rightarrow \mathbb{N}$ , and size function  $\varphi : C \rightarrow \mathbb{N}$ , respectively, assign each channel a number of initial tokens, a maximal number of tokens that can be stored, and the token size in bytes.*

In Figure 1, an example of an application graph  $g_A$  consisting of five actors  $A = \{a_1, \dots, a_5\}$  communicating via five communication channels  $C = \{c_1, \dots, c_5\}$  is given.

From an application itself, it is possible to determine the *memory footprint*  $M_F = \sum_{c \in C} \gamma(c) \cdot \varphi(c)$  by summing up each channel's memory requirement derived from the channel capacity in tokens  $\gamma(c)$  and the token size in bytes  $\varphi(c)$ .

Generally, an application contains multi-cast actors, e.g., actor  $a_2$  in Figure 1. Multi-cast actors just replicate the data tokens at their input, producing identical copies of data as tokens on the communication channels connected to their output ports. To exemplify,  $a_2$  copies its input tokens to  $a_3$  and  $a_4$  via the communication channels  $c_2$  and  $c_3$ , respectively. In the following, let the set of multi-cast actors of an application be denoted by  $A_M \subset A$ . Now, each multi-cast actor represents an opportunity for memory footprint reduction, as shown in Figure 2. As precisely one channel is connected to each actor port, the domain of the functions  $\delta$ ,  $\gamma$ , and  $\varphi$  can be extended to all the ports of a multi-cast actor. If one of these functions is applied to an actor port, it will be equivalent to applying the function to the channel that is connected to this port. With these definitions, a multi-cast actor satisfies the following sets of constraints:

$$\forall a \in A_M : |a.I| = 1 \wedge \forall i \in a.I : \kappa(i) = 1 \quad (1)$$

$$\wedge |a.O| > 1 \wedge \forall o \in a.O : \psi(o) = 1 \quad (2)$$

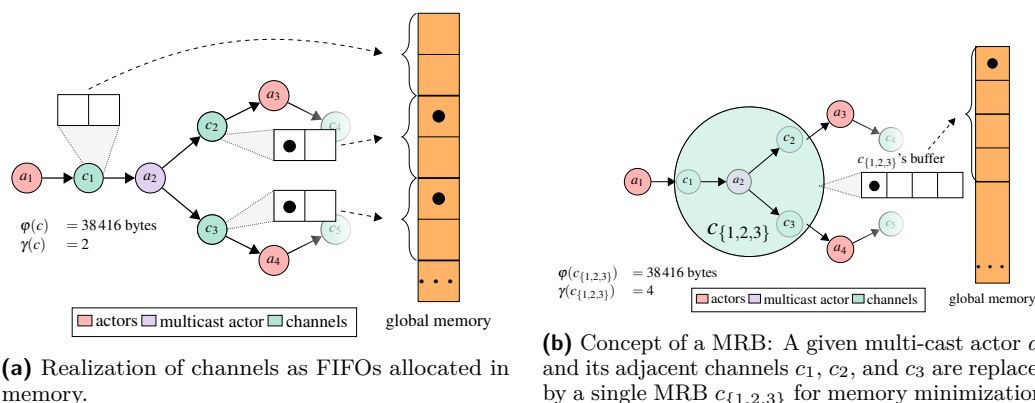
$$\wedge \forall i \in a.I, o \in a.O : \varphi(i) = \varphi(o) \quad (3)$$

$$\wedge \forall o', o'' \in a.O : \delta(o') = 0 \wedge \gamma(o') = \gamma(o'') \quad (4)$$

First, a multi-cast actor must have exactly one input port consuming one token per actor firing (see Equation (1)) and at least two output ports, each port producing one token per firing (see Equation (2)). Moreover, the token size of all consumed and produced tokens must be identical (see Equation (3)). Finally, output channels are assumed to be free of any token, and the channel capacities of the output channels are assumed to be identical (see Equation (4)).

### 3.2 Architecture Graph

A symmetric multi-core target architecture as shown in Figure 1, right, is modeled formally by an *architecture graph*:



■ **Figure 2** In (a), each channel connected to multi-cast actor  $a_2$  is realized as a FIFO allocated in memory storing the same information. In (b), memory minimization is performed by merging those redundant communication channels into MRB  $c_{\{1,2,3\}}$ .

► **Definition 3** (Architecture Graph). An architecture graph  $g_R$  is a tuple  $(R, L, b_{\overline{\omega}})$  composed of a set of vertices  $R$  modeling hardware resources (such as processors, memories, and a bus used to transport data from memories to processors and vice-versa) and a set of edges  $L \subseteq R \times R$  denoting communication links. Finally,  $b_{\overline{\omega}}$  denotes the bus bandwidth.

To exemplify, consider again the target architecture shown in Figure 1. The set of resources  $R$  is partitioned into a set of CPUs  $R_P = \{r_{\text{CPU}_1}, \dots, r_{\text{CPU}_4}\}$ , a set of memories  $R_M = \{r_{\text{SPM}_1}, \dots, r_{\text{SPM}_4}, r_{\text{DRAM}}\}$ , and the bus  $r_{\text{BUS}}$ . Here, each processor  $r_{\text{CPU}_i} \in R_P \subset R : 1 \leq i \leq |R_P|$  is assumed to have a local scratchpad memory  $r_{\text{SPM}_i} \in R_M \subset R$  reachable via the link  $(r_{\text{CPU}_i}, r_{\text{SPM}_i}) \in L$ . Furthermore, the processors can access the global memory  $r_{\text{DRAM}}$  via the bus  $r_{\text{BUS}}$ .

### 3.3 Specification graph

To perform explorations of allocations and mappings of actors to cores, and of channels to memories including the scheduling of actors and of data transfers between resources, a specification contains a set of mappings  $M = M_A \cup M_C$  that is partitioned into a set of potential mappings  $M_A \subseteq A \times R_P$  of actors to processors and mappings  $M_C \subseteq C \times R_M$  of channels to memories. Then, a *specification graph* can be formally defined as follows:

► **Definition 4** (Specification Graph). A specification graph  $g_S = (A \cup C \cup R, E \cup L \cup M)$  contains the architecture graph  $g_R$ , the application graph  $g_A$  and the set of potential mappings  $M$ .

### 3.4 Actor and Communication Channel Binding

A specification, in general, allows for multiple implementations. To derive a specific implementation, a DSE must determine *bindings* for all actors ( $\beta_A \subseteq M_A$ ) and all communication channels ( $\beta_C \subseteq M_C$ ). This step is often also called mapping. Here, each actor must be bound to exactly one processing resource (see Equation (5)). Conversely, each channel must be bound to exactly one memory resource (see Equation (6)).

$$\forall a \in A : |\beta_A \cap (\{a\} \times R_P)| = 1 \quad (5)$$

$$\forall c \in C : |\beta_C \cap (\{c\} \times R_M)| = 1 \quad (6)$$

Moreover, the binding of each channel  $c$  is constrained to either be bound to global memory  $r_{\text{DRAM}}$  or a scratchpad memory  $r_{\text{SPM}_i}$  local to the processor  $r_{\text{CPU}_i}$  onto which an actor is bound that either writes to or reads from channel  $c$ . This condition can be formalized as follows:

$$\begin{aligned} \forall (c, r) \in \beta_C : r &= r_{\text{DRAM}} \\ \vee \exists (a, r_{\text{CPU}_i}) \in \beta_A, o \in a.O : (o, c) \in E \wedge r &= r_{\text{SPM}_i} \\ \vee \exists (a, r_{\text{CPU}_i}) \in \beta_A, i \in a.I : (c, i) \in E \wedge r &= r_{\text{SPM}_i} \end{aligned} \quad (7)$$

Formally, each *feasible implementation*  $g_I$  must satisfy Equations (5)–(7). Note that different implementations may have identical actor bindings  $\beta_A$ , but differing in *schedules* due to differing channel bindings  $\beta_C$ , a limited bus bandwidth  $b_{\varpi}$  delaying those communications using the bus to transport data, or simply using a different scheduling algorithm. Note also that any processor  $r_{\text{CPU}_i}$  writing to its local scratchpad memory  $r_{\text{SPM}_i}$  creates no impact on scheduling because the bus is not used to transfer data. In contrast, the bus is utilized when writing to any other scratchpad or global memory which could create interference. Formally, for any bound channel  $(c, r_m) \in \beta_C$ , the transfer delay  $\tau(c, \eta)$  of transporting  $\eta$  data tokens over the bus assuming no bus contention is calculated as  $\tau(c, \eta) = \frac{\varphi(c) \times \eta \text{ [bytes]}}{b_{\varpi} \text{ [Gb/s]}}$ <sup>1</sup>.

#### 4 Multi-Reader Buffers (MRBs) for Memory Footprint Minimization

Using multi-cast actors in a Dataflow Graph (DFG) may result in sub-optimal implementations in terms of memory footprint. E.g., Figure 2a presents FIFO realizations for channels  $c_1$ ,  $c_2$ , and  $c_3$  of the application graph shown in Figure 1. There, the multi-cast actor  $a_2$  propagates identical data tokens to  $c_2$  and  $c_3$ . Figure 2b now introduces our concept of an MRB. By replacing a multi-cast actor and its adjacent channels by a single MRB node in which all outgoing channel buffers are replaced internally by just a single (shared) buffer. Semantically, the MRB acts as a channel in the transformed application graph that technically stores only one copy of live data shared between actors  $a_3$  and  $a_4$ .

Formally, the transformation of replacing a given multi-cast actor  $a_m$  with an MRB for a given application graph  $g_A$  is detailed in Algorithm 1. This algorithm returns a transformed application graph where the given multi-cast actor and the channels connected to it have been replaced by a corresponding MRB. For finding minimal memory footprint implementations, Algorithm 1 is simply applied to all multi-cast actors of an application.

Now, we present a possible MRB realization and its principle of operation. By definition, a MRB  $c_m$  has one writer  $a_w$  and multiple readers  $a_{r_i} \in A_r \subseteq A : 1 \leq i \leq |A_r|$ . Each MRB  $c_m$  has a write index  $\omega(c_m) \in \{0, 1, \dots, \gamma(c_m) - 1\}$  that indicates the next position in  $c_m$ 's buffer to be filled with the next token produced by the writer  $a_w$ . Similarly, each  $c_m$  manages read indices  $\rho_i(c_m) \in \{-1, 0, 1, \dots, \gamma(c_m) - 1\} : 1 \leq i \leq |A_r|$ , each index  $\rho_i(c_m)$  indicating a position in  $c_m$ 's buffer from which the next token consumed by reader  $a_{r_i}$  is read. The special value  $-1$  of a read index  $\rho_i(c_m)$  denotes that  $c_m$  is empty from  $a_{r_i}$ 's perspective. Then, the number of available tokens  $T(c_m, a_{r_i})$  from the perspective of each reader  $a_{r_i}$  and the number of free places  $F(c_m)$  in  $c_m$  from the perspective of the writer  $a_w$  can be determined as follows:

<sup>1</sup> Times for reading and writing local scratchpad data are assumed to be part of each actor's execution time.

■ **Algorithm 1** Multi-Reader Buffer (MRB) replacement.

---

```

1 Function insertMRB( $g_A, a_m$ )
2    $C_{del} \leftarrow \{c \in g_A.C \mid g_A.E \cap (\{c\} \times a_m.I \cup a_m.O \times \{c\}) \neq \emptyset\}$  // All channels connected to
    $a_m$ 
3    $c_m \leftarrow \text{createMRB}(a_m, C_{del})$  // Create MRB
4    $g_A.A \leftarrow g_A.A \setminus \{a_m\}$  // Remove multicast actor
5    $c' \leftarrow c : \exists i \in a_m.I, (c, i) \in g_A.E$  // input channel of  $a_m$ 
6    $g_A.E \leftarrow g_A.E \cup \{(o, c_m) \mid \exists a \in g_A.A, o \in a.O : (o, c') \in g_A.E\}$  // Connect  $c_m$  writer
7   for  $c'' \in C_{del} \setminus \{c'\}$  do // All output channels of  $a_m$ 
8      $g_A.E \leftarrow g_A.E \cup \{(c_m, i) \mid \exists a \in g_A.A, i \in a.I : (c'', i) \in g_A.E\}$  // Connect  $c_m$  reader
9      $\gamma(c_m) \leftarrow \gamma(c') + \gamma(c'')$  // Set capacity of MRB
10   $\delta(c_m) \leftarrow \delta(c')$  // Set initial tokens for MRB
11   $\varphi(c_m) \leftarrow \varphi(c')$  // Set token size of MRB
12   $g_A.C \leftarrow \{c_m\} + g_A.C \setminus C_{del}$  // Replace  $C_{del}$  by  $c_m$ 
13   $g_A.E \leftarrow \{(n, m) \in g_A.E \mid n \notin C_{del} \vee m \notin C_{del}\}$  // Remove edges connecting
   removed channels
14  return  $g_A$ 

```

---

$$T(c_m, a_{r_i}) = \begin{cases} 0 & \text{if } \rho_i(c_m) < 0 \\ ((\omega(c_m) - \rho_i(c_m) - 1) \bmod \gamma(c_m)) + 1 & \text{otherwise} \end{cases} \quad (8)$$

$$F(c_m) = \gamma(c_m) - \max_{a_{r_i} \in A_r} T(c_m, a_{r_i}) \quad (9)$$

Assuming the reader  $a_{r_i}$  consumes  $\kappa(a_{r_i})$  tokens, then it is blocked from firing as long as  $T(c_m, a_{r_i}) < \kappa(a_{r_i})$  holds. Accordingly, upon each read by an actor  $a_{r_i}$ , the corresponding read index  $\rho_i(c_m)$  is updated as follows:

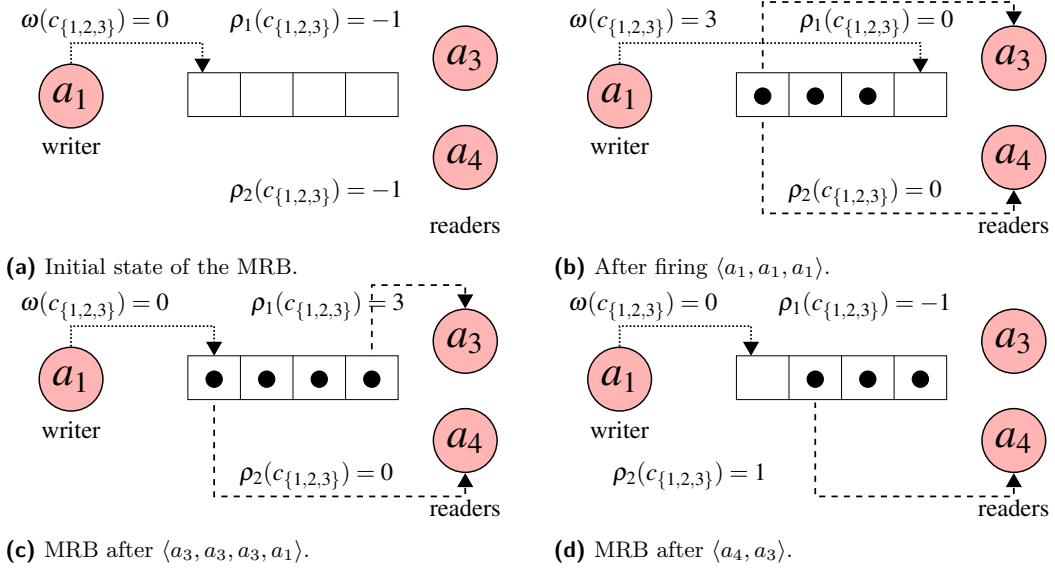
$$\rho_i(c_m) \leftarrow \begin{cases} -1 & \text{if } T(c_m, a_{r_i}) = \kappa(a_{r_i}) \\ (\rho_i(c_m) + \kappa(a_{r_i})) \bmod \gamma(c_m) & \text{otherwise} \end{cases} \quad (10)$$

Equivalently, assuming the writer  $a_w$  produces  $\psi(a_w)$  tokens, then it is blocked from firing as long as  $F(c_m) < \psi(a_w)$  holds. Accordingly, upon each write of actor  $a_w$ , Equation (11) is applied, which sets each read index  $\rho_i(c_m)$  with the value  $\omega(c_m)$  if  $\rho_i(c_m) = -1$ . Next, Equation (12) is applied, which advances the writer index  $\omega(c_m)$  by the number of produced tokens.

$$\forall_{1 \leq i \leq |A_r|} \rho_i(c_m) \leftarrow \begin{cases} \omega(c_m) & \text{if } \rho_i(c_m) = -1 \\ \rho_i(c_m) & \text{otherwise} \end{cases} \quad (11)$$

$$\omega(c_m) \leftarrow (\omega(c_m) + \psi(a_w)) \bmod \gamma(c_m) \quad (12)$$

To exemplify, consider the application in Figure 2. For the MRB  $c_{\{1,2,3\}}$ , the writer  $a_w$  is  $a_1$ , and the set of readers  $A_r$  is  $\{a_3, a_4\}$ . Of course, our presented MRB realization supports multi-rate dataflow. However, to ease the understanding of the presented example,  $a_1$ 's production rate is assumed here to be one, i.e.,  $\psi(a_1) = 1$ , and the same holds for the consumption rates of the readers, i.e.,  $\kappa(a_3) = \kappa(a_4) = 1$ . The MRB's read and write indices after various firings of the connected actors  $a_1$ ,  $a_3$ , and  $a_4$  are depicted in Figure 3. Assuming the MRB is initially empty, these read and write indices have values as shown in Figure 3a. Thus,  $T(c_{\{1,2,3\}}, a_3) = T(c_{\{1,2,3\}}, a_4) = 0$  and  $F(c_{\{1,2,3\}}) = \gamma(c_{\{1,2,3\}}) - \max\{0, 0\} = 4$ .



■ **Figure 3** MRB with one write index (pointer) indicating the location of the next token to be written. Moreover, each reading actor requires an index pointing to the position of the next token to read.

At this point (see Figure 3a), it is only possible to perform write operations. Before firing  $a_1$ , we must check if sufficient free places are available for the produced tokens, i.e.,  $F(c_{\{1,2,3\}}) = 4 \geq \psi(a_1) = 1$ . Next, assume actor  $a_1$  fires three times resulting in the state shown in Figure 3b. There, the write index  $\omega(c_{\{1,2,3\}})$  has advanced to 3 pointing to the next free place in the MRB's buffer. The read indices  $\rho_1(c_{\{1,2,3\}})$  and  $\rho_2(c_{\{1,2,3\}})$  have been updated during the first firing of actor  $a_1$  from  $-1$  to  $0$  pointing to the first token contained in the MRB.

At this point (see Figure 3b), we can also perform read operations. Before firing a reader  $a_{r_i}$ , we need to verify if there exist sufficient tokens to be consumed by the reader, i.e.,  $T(c_{\{1,2,3\}}, a_{r_i}) \geq \kappa(a_{r_i})$ . For instance, we are able to fire actor  $a_3$  because  $T(c_{\{1,2,3\}}, a_3) = ((3 - 0 - 1) \bmod 4) + 1 = 3 \geq 1$ . After firing the sequence  $\langle a_3, a_3, a_3, a_1 \rangle$ , the resulting state is shown in Figure 3c. There, the readers track different information about the state of the MRB. The reader  $a_3$  points to  $\rho_1(c_{\{1,2,3\}}) = 3$  and observes  $T(c_{\{1,2,3\}}, a_3) = ((0 - 3 - 1) \bmod 4) + 1 = 1$  token on the MRB whereas, reader  $a_4$  points to  $\rho_2(c_{\{1,2,3\}}) = 0$  and observes  $T(c_{\{1,2,3\}}, a_4) = ((0 - 0 - 1) \bmod 4) + 1 = 4$  tokens. From the perspective of the writer  $a_1$ , the MRB is full.

At this point (see Figure 3c), let the firing sequence  $\langle a_4, a_3 \rangle$  be observed. The resulting state of the MRB is shown in Figure 3d. From the perspective of  $a_3$ , the MRB is empty, i.e.,  $\rho_1(c_{\{1,2,3\}})$  is  $-1$ . The token placed at position 0 has been consumed because  $a_4$  has read it now seeing  $T(c_{\{1,2,3\}}, a_4) = ((0 - 1 - 1) \bmod 4) + 1 = 3$  more tokens. From the perspective of  $a_1$ , there is one free place as  $F(c_{\{1,2,3\}}) = \gamma(c_{\{1,2,3\}}) - \max\{0, 3\} = 4 - 3 = 1$ .

To evaluate the benefits of MRBs, the following section presents a DSE that decides whether to replace a multi-cast actor and its connected channels with an MRB.



■ **Table 1** DFGs employed to compare the presented DSE.

| Application          | # of instances | $ A $ | $ C $ | $ A_M $ | $\gamma(C)$ | $M_F$ (Reference) [MiB] |
|----------------------|----------------|-------|-------|---------|-------------|-------------------------|
| Sobel-4              | 1              | 27    | 33    | 4       | 1           | 82.6                    |
| Sobel-8              | 1              | 51    | 65    | 8       | 3           | 707.5                   |
| Multicamera          | 2              | 123   | 226   | 46      | 3           | 252.4                   |
| Optical flow         | 4              | 89    | 112   | 15      | 3           | 996.8                   |
| Object counting      | 5              | 96    | 120   | 15      | 2           | 105.6                   |
| Foreground detection | 7              | 139   | 170   | 24      | 2           | $5.01 \cdot 10^3$       |

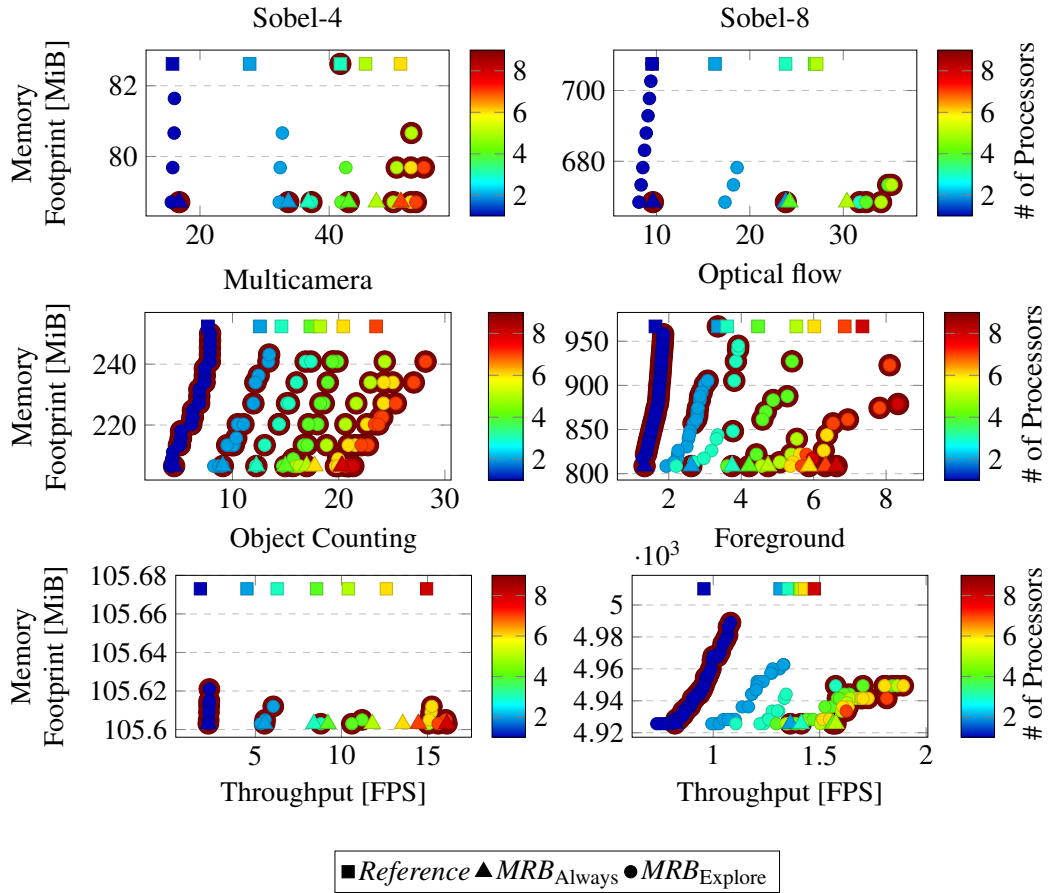
## 5 Experimental Results

For design space exploration, we consider the set of objectives  $\mathcal{F}^{\text{obj}} = \{f_1^{\text{max}}, f_2^{\text{min}}, f_3^{\text{min}}\}$  to be optimized. Here,  $f_1^{\text{max}}$ ,  $f_2^{\text{min}}$ , and  $f_3^{\text{min}}$  correspond to throughput, the number of allocated cores, and memory footprint, respectively. Our proposed DSE optimizes the mappings where each feasible implementation must satisfy the constraints presented in Equations (5)–(7). For each explored mapping, a schedule of actors and communications is determined using a First Come First Serve (FCFS) scheduler. The throughput is evaluated as the average Frames per Second (FPS) of running ten iterations of the obtained schedule. Moreover, the memory footprint is determined as presented in Section 3.

We employed the OpenDSE [14] framework for DSE using the NGSa-II elitist genetic algorithm [3] with a population size of 100 individuals, each generation generating 25 new individuals, and the crossover rate being set to 0.95. As a target architecture, we used a symmetric eight-core MPSoC, which connects each processor to a communication bus that, in its turn, is connected to a global memory. As target applications, Table 1 presents a benchmark composed of six real-world image processing applications obtained from self-developed Matlab/Simulink test cases [12]. As can be seen, for some applications, not only one, but even up to seven instances of application graphs were considered to run simultaneously. Shown in the table are also the number of actors, the number and capacity of channels, and the number of multi-cast actors contained in each application.

To quantify the effects of MRBs, we implemented three approaches *Reference*,  $MRB_{\text{Always}}$ , and  $MRB_{\text{Explore}}$ . Here, *Reference* only performs the optimization of mappings. The last column in Table 1 presents the memory footprint of each application obtained by the *Reference* approach, which serves as a baseline to compare the memory footprint reductions obtained by introducing MRBs in the applications. The approach  $MRB_{\text{Always}}$  applies Algorithm 1 to all the multi-cast actors in the application as a pre-processing step and then performs the optimization of mappings, thus resulting in memory-efficient implementations. Finally, approach  $MRB_{\text{Explore}}$ , besides optimizing the mappings, also explores selectively for each multi-cast actor the choice of its replacement by a MRB.

To explore the placement of the actors and channels, we define an *integer genotype* for each actor and channel. For an actor  $a$ , we assume that it can be mapped to all cores  $R_P$  and, hence, the genotype for an actor is  $\{1, 2, \dots, |R_P|\}$ . A given channel  $c$  can be mapped to (1) the global memory  $r_{\text{DRAM}}$ , (2) the producer core’s scratchpad memory, or (3) the consumer core’s scratchpad memory (as presented in Equation (7)), i.e., the genotype for a channel is  $\{1, 2, 3\}$ . Accordingly, the genotype of the *Reference* and  $MRB_{\text{Always}}$  approaches are given by  $\mathcal{G}_{\text{Always}} = \mathcal{G}_{\text{Reference}} = \{1, 2, \dots, |R_P|\}^{|A|} \times \{1, 2, 3\}^{|C|}$ .



■ **Figure 4** Pareto fronts of the last generation obtained for the six presented applications after 3,500 generations. Points circled brown are non-dominated points of the union of the three Pareto fronts.

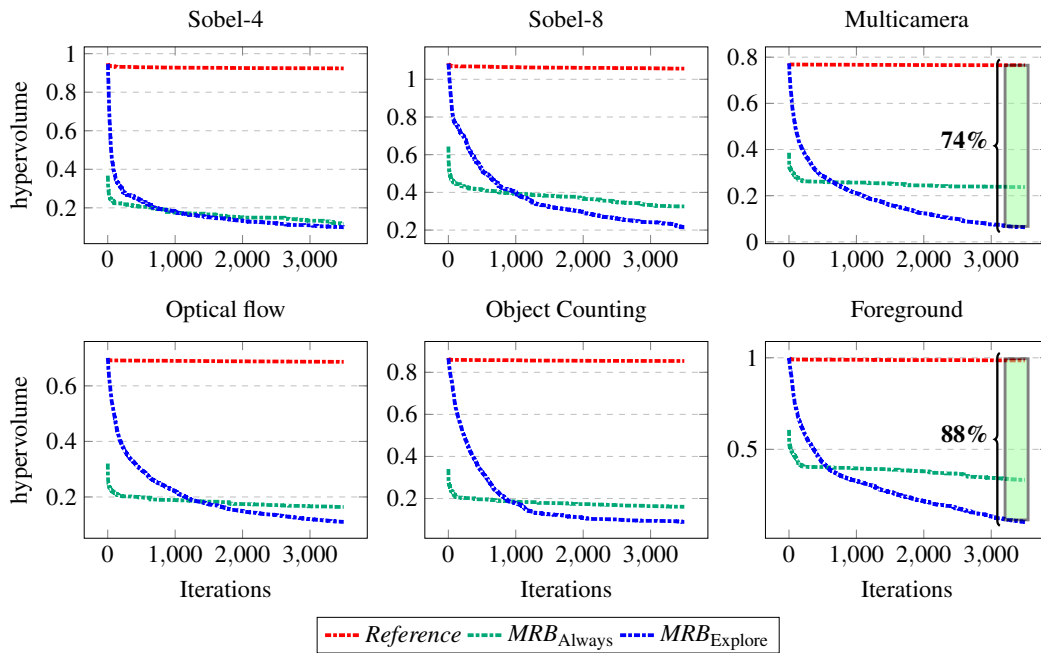
The  $MRB_{Explore}$  approach uses a binary genotype  $\{0, 1\}$  for each multi-cast actor to explore if the multi-cast actor and its connected channels are replaced by a MRB, where a 1 indicates a replacement. After replacing a given multi-cast actor  $a_m$  and its channels, the mapping of the newly introduced MRB is the same as the mapping of the channel being read by the replaced multi-cast actor  $a_m$ . Thus, the genotype of the  $MRB_{Explore}$  approach is given by  $\mathcal{G}_{Explore} = \mathcal{G}_{Reference} \times \{0, 1\}^{|AM|}$ .

In the following, we will show that our  $MRB_{Explore}$  approach can find better quality solutions than the *Reference* and  $MRB_{Always}$  approaches. For the three approaches under investigation, we performed ten independent DSE runs for each considered application. Each exploration ran 3,500 generations, recording those implementation candidates with optimal throughput at each point during exploration.

## 5.1 Comparison of Exploration Results

Due to multiple objectives to optimize, there not exists a single optimal solution due to the conflicting set of objectives. Figure 4 presents the Pareto fronts<sup>2</sup> of the last explored generation obtained for the six test applications for each of the investigated approaches. The

<sup>2</sup> Shown are the efficient (non-dominated) sets of solutions of the last generation as found during each explorative search. Note that these sets are approximations of a true Pareto front.



■ **Figure 5** Hypervolume scores obtained for the six applications.

color on each mark represents the number of allocated cores in each solution, ranging from single-core in blue to eight-core implementations in red. From the Pareto fronts, we observe that *Reference* delivers least efficient implementations in terms of memory footprint (see squares on top), only trading throughput for allocated cores. In contrast, the *MRB<sub>Always</sub>* front consists only of the least memory consuming implementations (see triangles at the bottom). When only looking at the Pareto fronts of *Reference* and *MRB<sub>Always</sub>*, we cannot observe any clear tendency indicating that replacing all the multi-cast actors with MRBs leads to higher throughput implementations. For instance, we can observe that *Reference* found better throughput solutions for the optical flow and multicamera, whereas *MRB<sub>Always</sub>* found better throughput solutions for the other applications.

Now, by selectively exploring the replacement of multi-cast actors according to our proposed approach *MRB<sub>Explore</sub>*, we observe that even higher throughput solutions could be found by trading the memory footprint in contrast to *Reference* and *MRB<sub>Always</sub>*. For applications where *Reference* obtains higher throughputs than *MRB<sub>Always</sub>*, e.g., optical flow and multicamera, our approach trades higher memory footprint to find even higher throughput solutions by performing fewer replacements of multi-cast actors. Conversely, for those applications where *MRB<sub>Always</sub>* finds higher throughput solutions than *Reference*, our approach also finds solutions applying more replacements of multi-cast actors reflected in solutions with less memory footprint. Notably, the highest throughput solutions found by our approach *MRB<sub>Explore</sub>* are up to 22% and 14% higher in average over the considered applications compared to the highest throughput solutions of *Reference*. In order to be able to compare the quality of the obtained Pareto fronts, we use the *hypervolume* indicator [9] which delivers a single indicator measuring the performance quality of each approach. For this purpose, we utilize the multi-objective metric Hypervolume [9], which delivers a single indicator measuring the performance quality of a given approach. Figure 5 presents the hypervolume indicator for each explored application. Each plot shows the average hypervolume of ten runs of each of the three approaches under observation over 3,500 generations. A value closer to

0 indicates a better quality of solutions. As can be seen, a significantly better (on average 67%) hypervolume indicator value can be observed at the exploration end of the  $MRB_{Always}$  approach compared to *Reference*.  $MRB_{Explore}$  improves the hypervolume indicator even 78% on average over all the investigated applications.

## 6 Conclusions

This paper introduced the concept of Multi-Reader Buffers (MRBs) as a memory-efficient implementation of multi-cast actors and their replacement as a graph transformation. Rather than replicating produced tokens for all readers, an MRB stores only one copy of data for all readers. Data is alive as long as the last reader has consumed it. MRBs provide minimal buffer implementations that are obtained by replacing all multi-cast actors in an application with MRBs. But as the replacement of a multi-cast actor by a MRB may affect the overall throughput of the application, i.e., in case of small buffer sizes, we proposed a DSE approach to explore the space of selective MRB replacements. It was shown that solutions can be found with up to 22% higher throughput compared to a reference approach. On average, the highest throughput implementations on the Pareto front were 14% higher over the considered applications and 78% in solution quality measured by a hypervolume indicator are reported.

---

## References

- 1 Mohamed Benazouz, Olivier Marchetti, Alix Munier-Kordon, and Pascal Urard. A new approach for minimizing buffer capacities with throughput constraint for embedded system design. In *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, pages 1–8, 2010. doi:10.1109/AICCSA.2010.5586972.
- 2 Tobias Blickle, Jürgen Teich, and Lothar Thiele. System-level synthesis using evolutionary algorithms. *Design Automation for Embedded Systems*, 3(1):23–58, 1998.
- 3 K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *Trans. Evol. Comp.*, 6(2):182–197, April 2002.
- 4 Jack Dennis. First Version of a Data Flow Procedure Language. In B. Robinet, editor, *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, Heidelberg, 1974.
- 5 Karol Desnos, Maxime Pelcat, Jean-François Nezan, and Slaheddine Aridhi. On Memory Reuse Between Inputs and Outputs of Dataflow Actors. *ACM Transactions on Embedded Computing Systems*, 15(2), February 2016. doi:10.1145/2871744.
- 6 Karol Desnos, Maxime Pelcat, Jean-François Nezan, and Slaheddine Aridhi. Memory analysis and optimized allocation of dataflow applications on shared-memory mpsoes. *Journal of Signal Processing Systems*, 80(1):19–37, 2015.
- 7 J. Falk, J. Keinert, C. Haubelt, J. Teich, and S. Bhattacharyya. A Generalized Static Data Flow Clustering Algorithm for MPSoC Scheduling of Multimedia Applications. In *Proceedings of ACM International Conference on Embedded Software*, pages 189–198, October 2008.
- 8 Joachim Falk, Christian Haubelt, and Jürgen Teich. Efficient representation and simulation of model-based designs in SystemC. In *Proceedings of the Forum on Specification and Design Languages*, volume 6, 2006.
- 9 Andreia P. Guerreiro, Carlos M. Fonseca, and Luís Paquete. The hypervolume indicator: Computational problems and algorithms. *ACM Comput. Surv.*, 54(6), July 2021. doi:10.1145/3453474.
- 10 J. Keinert, M. Streubühr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith. SYSTEMCODESIGNER - an Automatic ESL Synthesis Approach by Design Space Exploration and Behavioral Synthesis for Streaming Applications. *ACM Trans. on Design Automation of Electronic Systems*, 14(1):1:1–1:23, January 2009.

- 11 Martin Letras, Joachim Falk, Tobias Schwarzer, and Jürgen Teich. Multi-objective Optimization of Mapping Dataflow Applications to MPSoCs Using a Hybrid Evaluation Combining Analytic Models and Measurements. *ACM Trans. on Design Automation of Electronic Systems*, 26:1–33, 2020. doi:10.1145/3431814.
- 12 Martin Letras, Joachim Falk, Stefan Wildermann, and Jürgen Teich. Automatic Conversion of Simulink Models to SystemoC Actor Networks. In *Proc. of SCOPES*, pages 81–84, New York, NY, USA, 2017. ACM. doi:10.1145/3078659.3078668.
- 13 Amith R. Mamidala, Daniel Faraj, Sameer Kumar, Douglas Miller, Michael Blocksome, Thomas Gooding, Philip Heidelberger, and Gabor Dozsa. Optimizing mpi collectives using efficient intra-node communication techniques over the blue gene/p supercomputer. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 771–780, 2011. doi:10.1109/IPDPS.2011.220.
- 14 OpenDSE. “Open Design Space Exploration Framework”, 2018. URL: <http://opendse.sf.net/>.
- 15 S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proceedings of Design Automation Conference (DAC)*, pages 899–904, 2006. doi:10.1145/1146909.1147138.
- 16 J. Teich. Hardware/Software Codesign: The Past, the Present, and Predicting the Future. *Proc. IEEE*, 100(Special Centennial Issue):1411–1430, May 2012.
- 17 L. Thiele, K. Strehl, D. Ziegenglein, R. Ernst, and J. Teich. Funstate—an internal design representation for codesign. In *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051)*, pages 558–565, 1999. doi:10.1109/ICCAD.1999.810711.
- 18 Hervé Yviquel, Alexandre Sanchez, Pekka Jääskeläinen, Jarmo Takala, Mickaël Raulet, and Emmanuel Casseau. Embedded multi-core systems dedicated to dynamic dataflow programs. *Journal of Signal Processing Systems*, 80(1):121–136, 2015.