A taxonomy of maximally elastic buffers

A taxonomy of maximally elastic buffers

Rudolf H. Mak

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
E-mail R.H.Mak@tue.nl

September 20, 2004

Abstract

In most producer-consumer (sender-receiver) systems buffers are used for traffic smoothing. To perform that task optimally it is required that these buffers can maintain a high throughput for a wide range of occupancies, a property called elasticity. The maximal elasticity obtainable is a function of two design parameters of the buffer, viz. its storage capacity and its average i/o-distance. Whether maximal elasticity is indeed obtained by a particular buffer design depends, however, on the detailed structure of the buffer. In this paper we investigate for which parameters optimally elastic buffers exist and what the structural complexity of those buffers is. To address the latter issue we adopt a compositional approach in which buffers are constructed according to prescribed rules from fixed sets of basic building blocks. By variation of the construction methods and the base sets a taxonomy of buffer classes is obtained that guides the search for maximally elastic buffers of a certain complexity.

1 Introduction

Consider a producer-consumer system in which both production and consumption rate fluctuate mildly, but independently, around a common rate. If in such a system the producer and consumer are connected directly to each other, the system incurs a performance penalty. At any time the slowest component dictates the system throughput, which will be less than the common production-consumption rate.

To avoid this penalty, it is common practice to insert a FIFO-buffer between producer and consumer. The observed buffer behavior is then the following. On average half of the storage elements of the buffer are occupied and the buffer runs at the common rate. When the consumer temporarily slows down (or the producer speeds up), the buffer adjusts its communication rate at the appropriate side to cater for this change. At the opposite side the buffer keeps its communication rate unchanged. As a consequence the occupancy of the buffer increases. If after a short enough period the consumer resumes its original rate, the producer will not notice the temporary drop in consumption rate. The only trace left is that the occupancy of the buffer has increased. If on the other hand the decrease of consumption rate lasts too long, the buffer becomes completely filled, and the slower consumption rate determines the system throughput. In fact this degradation of performance is not abrupt. When the occupancy of the buffer rises above a certain level, the throughput starts to drop towards the current consumption rate. A similar phenomenon occurs when the production rate decreases (consumption rate increases) and the occupancy of the buffer drops below a certain level. Hence there is a range of occupancies centered around a half-filled buffer in which the buffer will keep the system throughput optimal. In general the extent of this range depends on the buffer's structure. E.g. it is well-known (see [7]) that linear buffers only operate at maximum throughput at a single occupancy level, i.e. when half-filled. Tree buffers that consist of a fan-out tree followed by a fan-in tree, on the contrary, run at maximal throughput for most occupancy levels. One can therefore say that tree buffers are more flexible or *elastic* than the linear buffers.

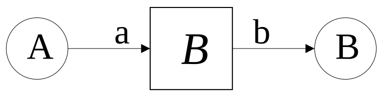
Clearly large elasticity is a desirable property for buffers when used as interconnect between a producer and a consumer. In [6] it has been shown that the maximum elasticity of a buffer is determined by only two parameters, viz. the storage capacity of the buffer and the i/o-distance of the buffer, which, as we will see in section 5, can be perceived as a dynamic diameter of the buffer. This upper bound is consistent with known results such as the one for linear buffers.

In general there are many buffer designs with identical storage capacity and i/o-distance. Whether any particular design indeed achieves maximum elasticity depends on structural details. So, from a design perspective, we can ask whether for a given capacity and i/o-distance there exists a maximally elastic buffer, if it is unique, and if not what the optimal buffer of least structural complexity looks like. To address the latter question we take the following approach. All buffers will be constructed from a (small) set of basic building blocks using a (small) set of construction rules. By varying the the set of building blocks and the set of construction rules we obtain a taxonomy of buffer classes. For each of these classes we investigate the presence or absence of maximally elastic buffers.

The remainder of this paper is organized as follows. In sections 2 and 3 we define the basic building blocks and the construction methods respectively. Section 4 defines a taxonomy of buffer classes based on their structural complexity. Sections 5 and 6 discuss design parameters and performance metrics respectively. Section 7 introduces an optimality criterion for buffers and indicates which optimal buffers we are looking for. Section 8 defines contour functions that can be used to search for optimal buffers of specified structural complexity. Section 9 explains how these contour functions can be computed and presents some results.

2 Basic building blocks

The buffers we shall consider throughout this paper are composed of instances of a fixed set of basic building blocks. In this section we define this set of components. The simplest component of all is the one-place buffer. It consumes a stream A of input values and produces a stream B of output values such that $B = A$. Figure 1 shows both a diagram and a VLSI program for this component. The diagram shows component *Buf*, its input stream A, and its output stream



```
Buf =
  proc (in a, out b)·
  |[ var x ⟫ (a?x; b!x)* ]|
```

Figure 1: A one-place buffer.

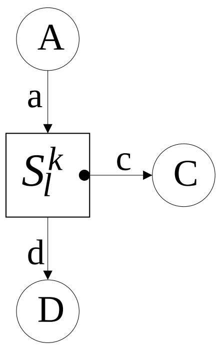
B. The program text resembles a procedure declaration such as one might encounter in any higher level programming language and serves to define the buffer process. It consists of a heading and a body. The heading specifies that the component is named *Buf* and that it has an *input port a* along which it receives the stream of input values A from its environment and an *output port b* along which its sends the stream of output values B to its environment. The body consists of a declaration part and a command. The declaration part declares a single variable x in which the buffer can store values. Hence it is a one-place buffer. The command defines the order of the communication events in which the component is involved. It expresses that *Buf* is capable of an infinite repetition, denoted by the Kleene star '*', of an input action $a?x$ followed by, denoted by the sequential composition operator ';', an output action $b!x$.

Besides components that store values, we also need components that divide and recombine streams. Such components are depicted in fig. 2. For $0 \leq k < l \leq 1$ component Split_l^k outputs the k-th[1] out of every l inputs along port c, and the remaining ones along port d. Note that in the diagram the output port that produces the selected input is marked with a black dot. Moreover, note that for $l = 1$ output port d is never used and the component behaves as if it would be the one-place buffer $\mathit{Buf}(a, c)$. Component Merge_l^k is the opposite of component Split_l^k in the sense that it inputs the k-th of every l outputs along port e and the remaining ones at port f. It can be verified by inspection of the program texts that connecting a merge component to the corresponding split component in such a way that the black dots match will yield the original stream again.

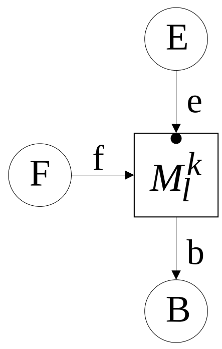
Given these basic building blocks we arrange them into an infinite sequence $\{\mathcal{C}_l \mid 1 \leq l\}$ of sets of increasing complexity. We define

$$\begin{aligned} \mathcal{C}_1 &= \{\mathit{Buf}\} \\ \mathcal{C}_{l+1} &= \mathcal{C}_l \cup \bigcup_{0 \leq k \leq l} \{\mathit{Split}_{l+1}^k\} \cup \bigcup_{0 \leq k \leq l} \{\mathit{Merge}_{l+1}^k\} \end{aligned}$$

[1]Beware that numbering starts at zero



```
$Split_l^k$ =
  $\mathbf{proc}\ (\mathbf{in}\ a, \mathbf{out}\ c, d)\cdot$
  $|[\ \mathbf{var}\ x$
  $\triangleright\ (\ \ (a?x; d!x)^k$
      $;\ \ (a?x; c!x)$
      $;\ \ (a?x; d!x)^{l-k-1}$
      $)^*$
  $]|$
```



```
$Merge_l^k$ =
  $\mathbf{proc}\ (\mathbf{in}\ e, f, \mathbf{out}\ b)\cdot$
  $|[\ \mathbf{var}\ x$
  $\triangleright\ (\ \ (f?x; b!x)^k$
      $;\ \ (e?x; b!x)$
      $;\ \ (f?x; b!x)^{l-k-1}$
      $)^*$
  $]|$
```

Figure 2: Split and merge components.

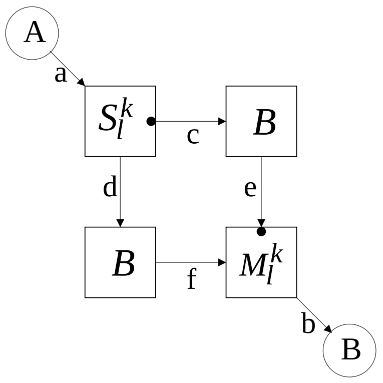
The set $\mathcal{C}$ of all basic building blocks is of course the union of these sets, i.e.

$$\mathcal{C} \quad = \quad \bigcup_{1\leq l} \mathcal{C}_l$$

3 Construction methods

So far we have encountered precisely one buffer, the single basic building block *Buf*. In general a buffer will be a single-input single-output (SISO) system of interconnected basic building blocks that has the first-in first-out (FIFO) property. The interconnection mechanism used is parallel composition. As an example consider the four-place buffer in fig. 3 which is a system that consists of a $Split_l^k$ component, a $Merge_l^k$ component, and two *Buf* components. These components are connected by four internal channels c, d, e, and f. Each channel connects precisely one input port of a subcomponent with precisely one output port of another subcomponent. In the diagram these channels are represented by arrows and in the program text they appear as the actual parameters that instantiate the port names in the component definitions. The latter reflects the fact that we adopt a CSP-like [5] communication model, viz. point-to-point communication with synchronous message passing.

Parallel composition is a general method to construct arbitrary systems (not necessarily buffers) from a set of basic components. As such it allows



$Buf4\ =$
 proc (**in** a, **out** b)·
 $[\![$ **chan** c, d, e, f
 $\triangleright$ $Split_l^k(a, c, d)$
 $\|$ $Buf(c, e)$
 $\|$ $Buf(d, f)$
 $\|$ $Merge_l^k(e, f, b)$
 $]\!]$

Figure 3: A four-place buffer.

the construction of buffers with arbitrary complex interconnection patterns. In this paper we demonstrate that this unbridled complexity is superfluous for the design of maximally elastic buffers. To this end we restrict the application of parallel composition. In particular we distinguish three simple construction methods that all have the property that when they are applied to buffers they yield another buffer. The first construction method is serial composition.

Definition 3.1 (Serial composition) *Let X and Y be systems each with a single input port and a single output port. Then the serial composition $SER(X, Y)$ of X and Y is obtained by connecting the output port of X to the input port of Y.*

$SER(X, Y)\ =$
 proc (**in** a, **out** b)·
 $[\![$ **chan** c $\triangleright$ $X(a, c) \parallel Y(c, b)]\!]$

$\square$

Obviously, the serial composition of two buffers is again a buffer.

The next construction method is wagging composition. It can be used to create buffers in the following manner. First the input stream is divided into two substreams each of which is fed through a buffer. Thereafter the two substreams are recombined into a single output stream. When the recombination process exactly matches the way in which the input stream is divided, the FIFO property is retained and the resulting system is again a buffer.

Definition 3.2 (Wagging composition) *Let X and Y be systems, each with a single input port and a single output port. Then for $2 \leq l$ and for $0 \leq k < l$ the wagging composition $WAG_l^k(X, Y)$ is obtained by surrounding these systems* *with a pair consisting of a $Split_l^k$ component and a $Merge_l^k$ component.*

```
$WAG_l^k(X, Y)$ =
 proc (in $a$, out $b$)·
 |[ chan $c, d, e, f$
 ▹ $Split_l^k(a, c, d)$
 || $X(c, e)$
 || $Y(d, f)$
 || $Merge_l^k(e, f, d)$
 ]|
```

□

Note that the number of items passing through subsystem Y is $l-1$ times the number of items passing through subsystem X.

The four-place buffer described at the start of this section is an example of a wagging composition, viz. $Buf4 = WAG_l^k(Buf, Buf)$.

The third and last construction method is a variation of wagging. Instead of only two substreams it allows an arbitrary number of substreams. Each substream, however, now carries the same fraction of items from the original input stream. Moreover, the subsystems responsible for division and recombination of the input stream are of a specific nature. The system used for division is called a multi-split component and is defined by

```
$Msplit_1$ =
 proc (in $a$, out $c[0 \,..\, 1)$)·
 |[ $Buf(a, c[0])$ ]|

$Msplit_{l+1}$ =
 proc (in $a$, out $c[0 \,..\, l{+}1)$)·
 |[ chan $d$
 ▹ $Split_{l+1}^l(a, c[l], d)$
 || $Msplit_l(d, c[0 \,..\, l))$
 ]|
```

Assume that the input stream is partitioned into consecutive blocks of l items. Then the output stream produced at port $c[k]$ of $Msplit_l$ consists of the k-th values of these blocks. The system used for recombination is called a multi-merge component and is defined by

```
$Mmerge_1$ =
 proc (in $e[0 \,..\, 1)$, out $b$)·
 |[ $Buf(e[0], b)$ ]|

$Mmerge_{l+1}$ =
 proc (in $e[0 \,..\, l{+}1)$, out $b$)·
 |[ chan $f$
 ▹ $Mmerge_l(e[1 \,..\, l+1), f)$
 || $Merge_{l+1}^0(e[0], f, b)$
 ]|
```

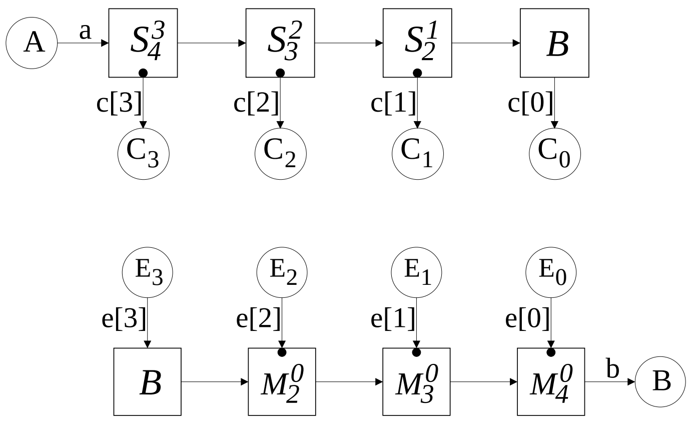


Figure 4: 4-way multi-split component (top) and multi-merge component (bottom).

Figure 4 contains diagrams of the 4-way versions of these components. Given these components we can now define multi-wagging composition by

Definition 3.3 (Multi-wagging composition) *For* $2 \leq l$*, let* $\{X_i \mid 0 \leq i < l\}$ *be a collection of systems each with a single input port and a single output port. Then the multi-wagging composition of these systems is given by*

$$
\begin{array}{l}
\mathit{MW}_l(X_0, \ldots, X_{l-1}) \;\; = \\
\quad \mathbf{proc}\ (\mathbf{in}\ a, \mathbf{out}\ b) \cdot \\
\quad |[\ \mathbf{chan}\ c[0\ ..\ l), e[0\ ..\ l) \\
\quad \rangle\ \mathit{Msplit}_l(a, c) \\
\quad \|\ (\|i : 0 \leq i < l : X_i(c[i], e[i])) \\
\quad \|\ \mathit{Mmerge}_l(e, b) \\
\quad]|
\end{array}
$$

□

Because $\mathit{Split}_2^1(a, d, c) = \mathit{Split}_2^0(a, c, d)$, multi-wagging coincides with wagging for $l = 2$, i.e.

$$MW_2(X_0, X_1) \;\; = \;\; WAG_2^1(SER(X_1, \mathit{Buf}), SER(\mathit{Buf}, X_0)) \qquad (1)$$

For $l \geq 3$, however, multi-wagging can not be expressed in terms of wagging, because the initial split component Split_l^{l-1} does not match the final merge component Merge_l^0.

4 Buffer classes

In this section we define a number of buffer classes. Each class is characterized by a set of basic building blocks and a set of construction methods. This is done in an incremental fashion. Hence the buffer classes can be ordered by set-inclusion, resulting in a lattice of buffer classes. Classes in the lower part of the lattice contain only buffers of low structural complexity, whereas classes in the upper part of the lattice contain "complicated" buffers as well.

Definition 4.1 *For $1 \leq n$ the class $\mathcal{U}_n$ of all systems that can be constructed from the set of basic building blocks $\mathcal{C}_n$ using the parallel composition operator is the smallest class such that*

1. $\mathcal{C}_n \subseteq \mathcal{U}_n$*,*
2. *If $X, Y \in \mathcal{U}_n$, then $X \parallel Y \in \mathcal{U}_n$* [2]*.*

□

Then the universe $\mathcal{U}$, i.e. the set of all systems that can be constructed using all basic building blocks $\mathcal{C}$, is the union of all sets $\mathcal{U}_n$,

$$\mathcal{U} = \bigcup_{1 \leq n} \mathcal{U}_n$$

Furthermore, define $\mathcal{B} \subseteq \mathcal{U}$ as the set of all buffers, and define $\mathcal{B}_n = \mathcal{B} \cap \mathcal{U}_n$ as the set of buffers composed of basic building blocks from $\mathcal{C}_n$.

By restriction of the construction methods and limitation of the set of basic building blocks we now define a numbers of buffer classes. The simplest of these classes uses only serial composition.

Definition 4.2 *The class $\mathcal{S}$ of all buffers that can be constructed using only one-place buffers and serial composition is the smallest class such that*

1. *$Buf \in \mathcal{S}$,*
2. *If $X, Y \in \mathcal{S}$, then $SER(X, Y) \in \mathcal{S}$.*

□

Since the only way to interconnect one-place buffers is by serial composition, we have $\mathcal{S} = \mathcal{B}_1$. The buffers of class $\mathcal{S}$ are also known as linear buffers.

Definition 4.3 (Linear buffers) *The family of linear buffers $\{LBUF_l \mid 1 \leq l\}$ is defined by*

$$\begin{aligned} LBUF_1 &= Buf \\ LBUF_{l+1} &= SER(LBUF_l, Buf) \end{aligned}$$

□

Allowing also wagging composition results in many more buffer classes. To be precise, one for each class of basic components.

[2]There may be many valid ways to interconnect ports of X and Y. It should be understood that every one of these interconnection patterns yields a new element of the class.

Definition 4.4 *For $1 \leq n$ the class $\mathcal{W}_n$ of all buffers that can be constructed using serial and wagging composition and using only basic building blocks from $\mathcal{C}_n$ is the smallest class such that*

1. *$Buf \in \mathcal{W}_n$,*
2. *If $X, Y \in \mathcal{W}_n$, then $SER(X, Y) \in \mathcal{W}_n$,*
3. *If $X, Y \in \mathcal{W}_n$, then $WAG_l^k(X, Y) \in \mathcal{W}_n$, for all $0 \leq k < l \leq n$.*

□

Note that $\mathcal{W}_1 = \mathcal{S}$, because clause 3 in this definition is inapplicable, since $WAG_{k,l}$ is not defined for $l \leq 1$. The class $\mathcal{W}$ of all buffers that can be constructed using the set of all basic building blocks $\mathcal{C}$, but using only serial and wagging composition, is given by

$$\mathcal{W} \quad = \quad \bigcup_{1 \leq n} \mathcal{W}_n$$

An important family of buffers that can be constructed using wagging is the family of tree buffers.

Definition 4.5 (Tree buffers) *The family of tree buffers $\{TBUF_l \mid 1 \leq l\}$ is defined by*

$$\begin{aligned} TBUF_1 &= Buf \\ TBUF_2 &= SER(Buf, Buf) \\ TBUF_{l+2} &= WAG_2^0(TBUF_l, TBUF_l) \end{aligned}$$

□

As an example fig. 5 depicts the 10-place tree buffer $TBUF_5$.

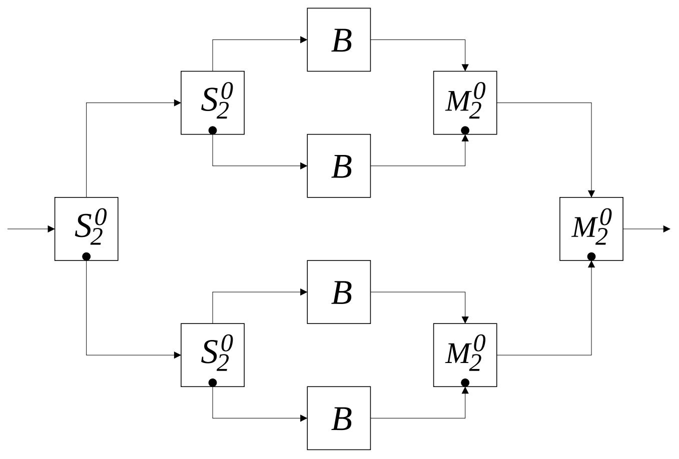


Figure 5: The tree buffer $TBUF_5$.

Whereas class $\mathcal{S}$ contains only linear buffers, class $\mathcal{W}$ contains much more than the tree buffers. To get some impression of the possibilities consider the following buffer

$$\mathit{Diamond} \quad = \quad \mathit{WAG}_3^0(\mathit{LBUF}_5, \mathit{TBUF}_5)$$

a diagram of which is given in fig. 6.

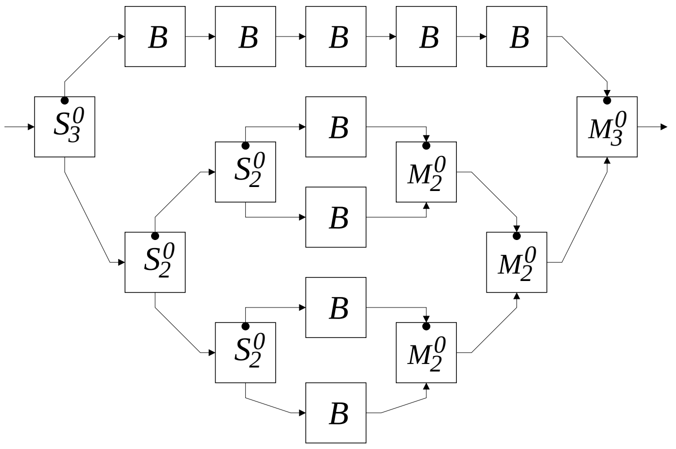


Figure 6: The diamond buffer.

Finally, allowing also multi-wagging results in yet another buffer class per class of basic building blocks.

Definition 4.6 *For $1 \leq n$ the class $\mathcal{M}_n$ of all buffers that can be constructed using serial composition, wagging composition, and multi-wagging composition and using only basic building blocks from $\mathcal{C}_n$ is the smallest class such that*

1. *$\mathit{Buf} \in \mathcal{M}_n$,*
2. *If $X, Y \in \mathcal{M}_n$, then $\mathit{SER}(X,Y) \in \mathcal{M}_n$,*
3. *If $X, Y \in \mathcal{M}_n$, then $\mathit{WAG}_l^k(X,Y) \in \mathcal{M}_n$, for $0 \leq k < l \leq n$,*
4. *If $X_i \in \mathcal{M}_n$, for all $0 \leq i < n$, then $\mathit{MW}_k(X_0, \ldots, X_{k-1}) \in \mathcal{M}_n$, for $2 \leq k < n$.*

□

Because for $l = 1$ multi-wagging is not defined and for $l = 2$ multi-wagging is a special case of wagging (see formula **??**), the first two multi-wagging classes coincide with the first two wagging classes, i.e. $\mathcal{M}_1 = \mathcal{W}_1$ and $\mathcal{M}_2 = \mathcal{W}_2$. Let $\mathcal{M}$ be the class of all buffers that can be constructed using all basic building blocks $\mathcal{C}$ and all three construction methods. Then

$$\mathcal{M} \quad = \quad \bigcup_{1 \leq n} \mathcal{M}_n$$

An example of buffers that require multi-wagging for their construction is given by the family of square buffers introduced by Brunvand [2].

Definition 4.7 (Square buffers) *The family of square buffers* $\{SBUF_l \mid 1 \leq l\}$ *is defined by*

$$\begin{array}{rcl} SBUF_1 & = & Buf \\ SBUF_2 & = & WAG_2^0(Buf, Buf) \\ SBUF_{l+2} & = & MW_{l+2}(LBUF_l, \ldots, LBUF_l) \end{array}$$

□

Figure 7 contains a diagram of $SBUF_5$.

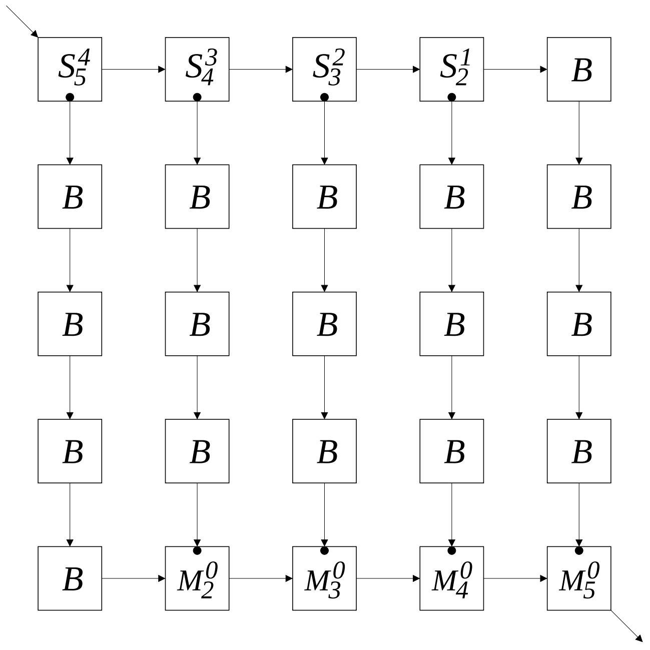


Figure 7: The square buffer $SBUF_5$. The building blocks of the top row constitute a $Msplit_5$ component and the building blocks of the bottom row constitute a $Mmerge_5$ component.

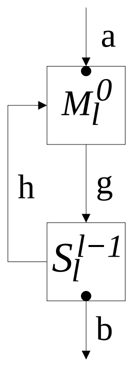
When we order the buffer classes defined above using set inclusion $\subseteq$, a lattice of buffer classes emerges (see figure 8). In this lattice three chains can be distinguished, viz. the $\mathcal{W}$-chain, the $\mathcal{M}$-chain, and the $\mathcal{B}$-chain. For $\mathcal{X} \in \{\mathcal{W}, \mathcal{M}, \mathcal{B}\}$ we call $\mathcal{X}_l$ the class of *index l* in chain $\mathcal{X}$.

With the exception of the equalities already mentioned above all inclusions between buffer classes in the lattice are proper. Within the chains and between classes of the $\mathcal{W}$-chain and the $\mathcal{M}$-chain with the same index this is obvious from the class definitions. To demonstrate that the inclusions between classes of the $\mathcal{M}$-chain and the $\mathcal{B}$-chain with the same index are proper as well, we indicate for each index l, $2 \leq l$, a buffer that belongs to $\mathcal{B}_l$ but not to $\mathcal{M}_l$.

Consider component RW_l displayed in fig. 9. Somewhat surprising this component is a buffer. This can be seen by following an item on its path through the system. After the merge component has accepted an item from the environment along port a, it passes the item to the split component along channel

$$\begin{array}{ccccccc}
 & & \mathcal{W} & \subset & \mathcal{M} & \subset & \mathcal{B} \\
 & & \cup & & \cup & & \cup \\
 & & \vdots & & \vdots & & \vdots \\
 & & \cup & & \cup & & \cup \\
 & & \mathcal{W}_3 & \subset & \mathcal{M}_3 & \subset & \mathcal{B}_3 \\
 & & \cup & & \cup & & \cup \\
 & & \mathcal{W}_2 & = & \mathcal{M}_2 & \subset & \mathcal{B}_2 \\
 & & \cup & & \cup & & \cup \\
\mathcal{S} & = & \mathcal{W}_1 & = & \mathcal{M}_1 & = & \mathcal{B}_1
\end{array}$$

Figure 8: A lattice of buffer classes.



```
$RW_l$ =
  **proc** (**in** $a$, **out** $b$)·
  |[ **chan** $g, h$,
  ▷ $Merge_l^0(a, h, g)$
  || $Split_l^{l-1}(g, b, h)$
  ]|
```

Figure 9: Two-place reverse-wagging buffer.

g. Subsequently the split component returns the item to the merge component along channel h, whereafter it is again forwarded to the split component along channel g. This happens $l-1$ times, until finally the split component delivers the item to the environment along port b. Thus each item passes l times through both the merge and the split component before leaving the system. Only when an item has entered the split component for the last time can the system accept a new item. Therefore successive items cannot overtake each other and the system is a FIFO buffer. By definition this buffer is a member of class $\mathcal{B}_l$, but clearly it is neither a member of class $\mathcal{W}_l$ nor of class $\mathcal{M}_l$.

In fact $\{RW_l \mid 2 \leq l\}$ can be seen as a family of increasingly clumsy two-place buffers. Compared to the standard two-place buffer $SER(Buf, Buf)$ they run at lower throughput (see section 6 for a definition of this quantity). In addition they consume more energy and have larger latencies, because they circulate the data items. They are, however, maximally elastic, but so is $SER(Buf, Buf)$. Hence they illustrate one of the main conclusions of this paper, that there is no added value in searching for maximally elastic buffers outside class $\mathcal{M}$.

5 Design parameters

In this section we adopt the point of view that a buffer is nothing but a graph whose nodes are one-place storage components and whose edges are the channels that connect these storage components. When we know the number of nodes of this graph and its diameter we have already a good indication of the structure of a buffer. We are interested in the diameter of a buffer is, because it represents the length of the longest path. Assuming this path runs from the input to the output, it is an indication of the maximal number of storage cells visited by any data item on its passage through the buffer. Note, however, that by the very nature of the split and merge components used, all buffers satisfy the property that the path taken by a data item is only a function of the rank of that data item in the input stream. So instead of just the maximum number of storage locations it is possible to compute the average number of storage location visited. The relevance of this quantity will become apparent in the next section (lemma 6.1).

So the design parameters of a buffer that we are interested in are the total storage capacity and the average number of storage cells visited by a data item on its passage through the buffer.

Definition 5.1 (Capacity) *The (storage) capacity $\kappa(X)$ of a system X is the sum of the number of storage locations (represented by variables in the program texts) of each of its basic building blocks.*

Since each basic building block contains a single variable, the capacity of a buffer is equal to the number of its basic building blocks.

Definition 5.2 (I/o-distance) *The i/o-distance $\delta(X)$ of a system X is the average number of storage locations visited by any data item on its passage through the system. When all data items visit the same number of storage locations, we say that the system is* ***equidistant****.* □

Whereas the capacity of a buffer is always a natural number, the i/o-distance of a buffer can also be fractional. E.g. the buffer depicted in fig. 10 has i/o-distance $\frac{3l-1}{l}$, since one item out of every l items visits 2 storage locations and the other ones 3. Of course the i/o-distance of an equidistant buffer is an integer.

Obviously linear buffers are equidistant, and it is not hard to verify that tree buffers are equidistant as well. Moreover, notice that in a multi-wagging composition MW_l the k-th input item out of every input block of length l visits precisely $l-k$ storage locations of the multi-split component and $k+1$ storage locations of the multi-merge component. Hence the total number of storage locations visited in both these components equals $l+1$ for every data item. Therefore also the square buffers are equidistant.

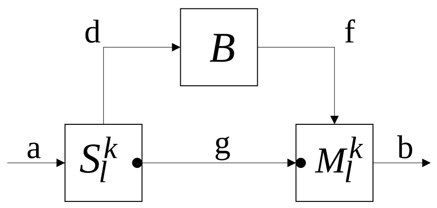


Figure 10: Buffer with fractional i/o-distance.

Definition 5.3 **((κ,δ)-buffers)** *A buffer with capacity κ and i/o-distance δ is called a (κ,δ)-buffer. The set of all (κ,δ)-buffers is denoted by $\mathcal{B}^{\kappa}_{\delta}$.* □

As an example, LBUF_l is a (l,l)-buffer, SBUF_l is a $(l^2, 2l{-}1)$-buffer, and for TBUF_l it depends on the parity of l. TBUF_{2m} is a $(2^{m+1}{-}2, 2m)$-buffer, and TBUF_{2m+1} is a $(3\cdot 2^m{-}2, 2m{+}1)$-buffer.

In general there will be (κ,δ)-buffers for almost any pair of (κ,δ)-coordinates. Also given a pair of coordinates there will be many buffers with those coordinates. Amongst these we are interested in the ones with high elasticity and preferably low structural complexity. Forgetting about elasticity for the moment, we first investigate whether constraining the structural complexity limits the number of admissible (κ,δ)-pairs.

The following properties state the effects of the various construction methods with respect to the design parameters. For serial composition we have

Property 5.1 *For arbitrary buffers X and Y, let $Z = \mathit{SER}(X,Y)$. Then*

$$\begin{aligned}\kappa(Z) &= \kappa(X)+\kappa(Y)\\ \delta(Z) &= \delta(X)+\delta(Y)\end{aligned}$$

□

An immediate consequence of this property is that for any buffer of class $\mathcal{S}$ the i/o-distance and capacity are equal. For wagging composition we have

Property 5.2 *For $2 \le l$ and arbitrary buffers X and Y, let $Z = \mathit{WAG}_l^k(X,Y)$. Then*

$$\begin{aligned}\kappa(Z) &= 2+\kappa(X)+\kappa(Y)\\ \delta(Z) &= 2+\frac{\delta(X)+(l{-}1)\delta(Y)}{l}\end{aligned}$$

□

For multi-wagging composition we have

Property 5.3 *For $2 \le l$ and a family of arbitrary buffers $\{X_i \in\mid 0 \le i < l\}$, let $Z = \mathit{MW}_l(X_0,\ldots,X_{l-1})$. Then*

$$\begin{aligned}\kappa(Z) &= 2l+\sum_{0\le i<l}\kappa(X_i)\\ \delta(Z) &= l+1+\frac{1}{l}\sum_{0\le i<l}\delta(X_i)\end{aligned}$$

□

Combination of these three properties shows that restricting the construction of a buffer to these methods will result in a buffer whose i/o-distance cannot exceed its capacity.

Theorem 5.1 *For all buffers $X \in \mathcal{M}$, the i/o-distance $\delta(X)$ is at most the capacity $\kappa(X)$.*

Proof. By structural induction. For $X = Buf$ the statement is obviously true. Simple computations using properties 5.1, 5.2, and 5.3 show that neither by serial composition nor by wagging or multi-wagging composition a buffer can be constructed with i/o-distance larger than its capacity, unless already one of the composing parts has an i/o-distance larger than its capacity. □

First of all note that this bound can not be improved upon by limiting the class of basic components, since the linear buffers already have i/o-distance equal to their capacity. Furthermore, note that for buffers outside class $\mathcal{M}$ the i/o-distance is in fact unbounded, because component RW_l is a $(2, 2l)$-buffer, and serial composition with a linear buffer of capacity m yields buffer $SER(LBUF_m, RW_l)$, which is a $(m+2, m+2l)$-buffer. Hence for every capacity $\kappa \geq 2$ there exists a $(\kappa, \kappa+2(l-1))$-buffer, for all $l \geq 2$.

Having established an upper bound for the i/o-distance of buffers of class $\mathcal{M}$, we now investigate whether there exists a non-trivial, i.e. larger than constant, lower bound for the i/o-distance of buffers of that class. In general this is not the case, not even for buffers of subclass $\mathcal{W}$, because it follows from property 5.2 that $WAG_l^0(LBUF_{l+1}, Buf)$ is a $(l+4, 4)$-buffer. For equidistant buffers the situation, however, is slightly better. For those buffers there exists a logarithmic lower bound.

Theorem 5.2 *Let X be an equidistant buffer with capacity κ and i/o-distance δ. If δ is even, then $\kappa \leq 2^{\frac{\delta}{2}+1} - 2$ and if δ is odd, then $\kappa \leq 3 \cdot 2^{\frac{\delta-1}{2}} - 2$.*

Proof. Let X be an arbitrary equidistant buffer. Assign each basic component of X to a layer according to the following procedure. Layer 0 consists of the basic component that contains the input port of X. Layer $l+1$ contains the basic components that have not yet been assigned to a layer and that are connected to a basic component in layer l by an internal channel. Since every basic component has at most two output ports, the number of components in layer $l+1$ is at most twice the number of basic components in layer l. Because the storage capacity of each basic component is 1, it therefore follows that the storage capacity of the first m layers is at most 2^m-1. Of course a similar result can be obtained for the last m layers, when we start the layer assignment at the the basic component that contains the output port of the buffer. Because X is equidistant the number of layers is at most δ. Hence a combination of the forward and backward layer assignments yields the desired result. □

Because equality holds for the tree buffer $TBUF_\delta$, the upper bound in this theorem is tight. Combination of theorems 5.1 and 5.2 yields

Corollary 5.1 *Let X be an equidistant buffer in class $\mathcal{M}$. Then $E(\kappa(X)) \le \delta(X) \le \kappa(X)$, where function E is defined* [3] *by*

$$\begin{array}{rcl} E(\kappa) & = & \downarrow \{\delta \mid \delta \text{ is even} \wedge \kappa \le 2^{\frac{\delta}{2}+1} - 2\} \\ & \downarrow & \downarrow \{\delta \mid \delta \text{ is odd} \wedge \kappa \le 3 \cdot 2^{\frac{\delta-1}{2}} - 2\} \end{array} \qquad (2)$$

□

Based on the above corollary, the (κ, δ)-space can be partitioned into three areas, see fig. 11. In the following sections it will be shown that the buffers in area A_3 perform poorly with respect to throughput and that for the buffers in area A_1 there is strong indication that they can not be maximally elastic. Therefore our prime interest is in the buffers of area A_2.

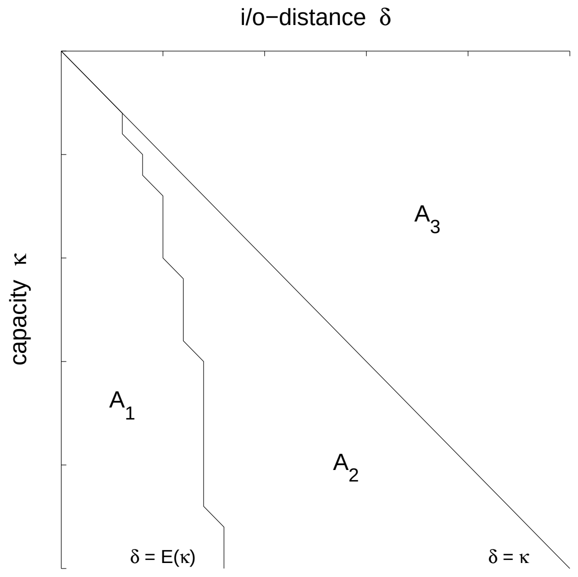


Figure 11: Three-partition of the (κ, δ)-space.

Sofar only the effect of restricting the construction methods to those of class $\mathcal{M}$ have been considered. The next theorem shows that neither a further restriction of the construction methods to those of class $\mathcal{W}$ nor limiting the set of basic building blocks to a subclass $\mathcal{C}_l \subseteq \mathcal{C}$, provided $2 \le l$, will lead to more stringent bounds for the i/o-distance.

Theorem 5.3 *For all (κ, δ)-pairs with $E(\kappa) \le \delta \le \kappa$ there exists an equidistant (κ, δ)-buffer in class $\mathcal{W}_2$.*

Proof. First of all note that the same range of (κ, δ)-pairs is considered, when

[3] Here and elsewhere in this paper the down-arrow '$\downarrow$' denotes a minimum operator, either in its monadic form as the minimum over a set or in its dyadic form as the minimum of two numbers. The minimum over the empty set is defined as $+\infty$. Likewise the up-arrow '$\uparrow$' denotes a maximum operator.

we replace constraint $E(\kappa) \leq \delta \leq \kappa$ by $\delta \leq \kappa \leq E^{-1}(\delta)$. Next, for arbitrary δ, consider $TBUF_\delta$, the tree buffer with i/o-distance δ. Recall that this tree buffer is equidistant and that its capacity is $E^{-1}(\delta)$. Therefore, if we can show that there exists a sequence of transformations that transforms $TBUF_\delta$ into the linear buffer $LBUF_\delta$, which has the same i/o-distance δ, but also has capacity δ, in such a way that each transformation in the sequence has the property that:

1. it reduces the capacity by 1
2. it preserves the i/o-distance
3. it preserves equidistance
4. it preserves membership of $\mathcal{W}_2$

then we are done, because then we have shown that for each (κ, δ)-pair in the range $\delta \leq \kappa \leq E^{-1}(\delta)$ there exists an equidistant buffer in class $\mathcal{W}_2$. For $\delta = 1$ and $\delta = 2$ the sequence is empty, because for those values the tree buffers and the linear buffers are identical. For $\delta = 3$ the sequence consists of the single transformation that replaces $TBUF_3$ by $LBUF_3$. For $\delta \geq 4$ we proceed by recursion. First we transform $TBUF_\delta = WAG_2^0(TBUF_{\delta-2}, TBUF_{\delta-2})$ into $WAG_2^0(LBUF_{\delta-2}, LBUF_{\delta-2})$ by application of a sequence of transformations to each of the two sub-buffers $TBUF_{\delta-2}$. Then we transform the resulting buffer $WAG_2^0(LBUF_{\delta-2}, LBUF_{\delta-2})$ into $LBUF_\delta$ by pushing the split-component towards the merge-component by a series of push-transformations, see fig. 12, until we have obtained buffer $SER(LBUF_{\delta-3}, TBUF_3)$, upon which we apply a final transformation that replaces $TBUF_3$ by $LBUF_3$. $\square$

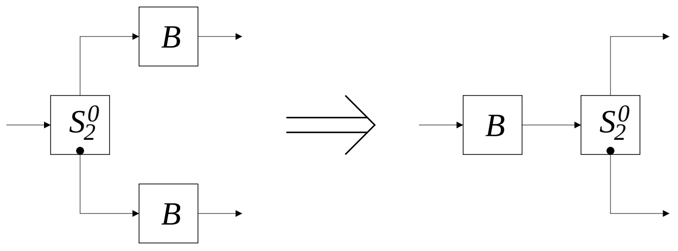


Figure 12: Push transformation.

This theorem suggests that it is sufficient to consider only very simple buffers. So apparently class $\mathcal{W}_2$ is already rich enough to provide us with an equidistant buffer for any (κ, δ)-pair in area A_2. Once we take the elasticity of buffers into account, however, this is no longer the case and the classes of the lattice that lie between $\mathcal{W}_2$ and $\mathcal{M}$ become of interest.

6 Performance metrics

The performance metrics introduced in this section assume a discrete timing model. A proper formal treatment (see [6]) requires the introduction of so-called *schedules* that map every communication event of a system onto a specific time-slot in such a manner that the communication order of the basic building blocks, as specified by their program texts, is respected. Since the metrics of interest can be defined and understood without the introduction of these schedules, and the crucial relationships between them hold for all schedules, we will take this theoretical foundation for granted, and make do with slightly less formal definitions. The reader who is nevertheless interested in the technical details is referred to [6].

The first performance metric considered is the average throughput.

Definition 6.1 (Average throughput) *For system X the average throughput $\Theta(X)$ is the average number of items accepted (or delivered) per unit of time.* □

There is a close relationship between the average throughput of a system and the average throughput of its basic building blocks.

Lemma 6.1 *Let X be a system with i/o-distance δ, and let $\{C_j \mid 0 \le j < n\}$ be the set of basic building blocks from which X is composed. Furthermore, let θ_j be the average throughput of block C_j, when the system X runs at average throughput θ. Then*

$$\theta\delta \quad = \quad \sum_{0 \le j < n} \theta_j$$

Proof. Let ϕ_j be the fraction of all data items that visit component C_j on their passage through the system. Then, by definition, the contribution of C_j to the i/o-distance of X is ϕ_j. Since $\theta_j = \phi_j \theta$, the result follows. □

The second performance metric considered is the average occupancy.

Definition 6.2 (Average occupancy) *For system X we define the instantaneous occupancy $\omega_t(X)$ at time slot t as the number of items accepted by the system before time slot t minus the number of items delivered by the system before time slot t. The average occupancy $\Omega(X)$ is given by*

$$\Omega(X) \quad = \quad \lim_{t \to \infty} \frac{1}{t} \sum_{0 \le \tau < t} \omega_\tau(X)$$

□

Note that the average occupancy of a system is the sum of the average occupancies of the basic building blocks from which it is constructed.

Lemma 6.2 *Let X be a system, and let $\{C_j \mid 0 \le j < n\}$ be the set of basic building blocks from which X is composed. Furthermore, let ω_j be the average occupancy of block C_j, when the system X runs at average occupancy ω. Then*

$$\omega \quad = \quad \sum_{0 \le j < n} \omega_j$$

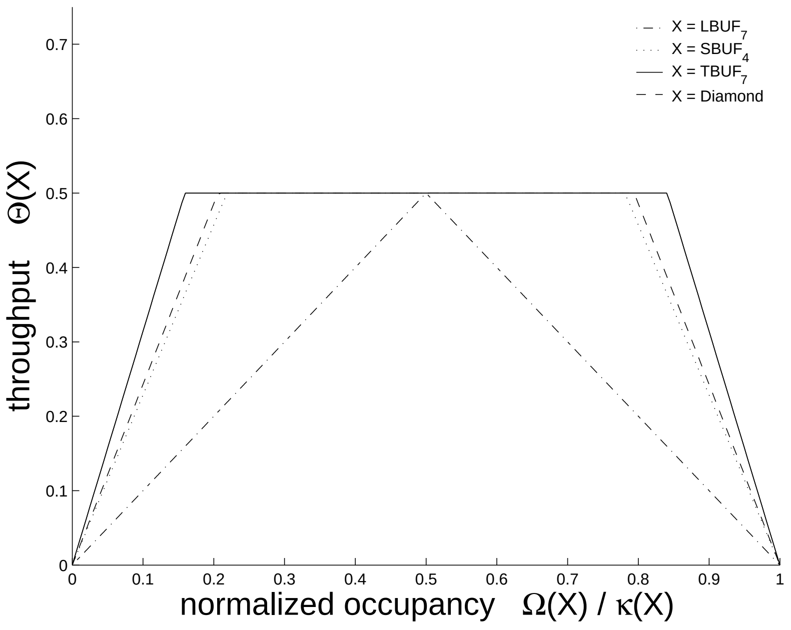


Figure 13: Throughput as function of the normalized occupancy, for some buffer designs.

Proof. Consider the instantaneous occupancy ω_t at any time slot t. This counts the number of items present in the system at the beginning of time slot t. Each of these items must be present in precisely one of the basic building blocks C_j. Hence the instantaneous occupancy of the buffer equals the sum of the instantaneous occupancies of its building blocks and therefore the same holds for the average occupancy. □

Obviously the instantaneous occupancy, and therefore also the average occupancy, varies between zero and the storage capacity of a buffer. Therefore it is sometimes convenient to express the average occupancy as a fraction of the storage capacity. We shall refer to this fraction as the *normalized occupancy.*

Figure 13 shows the average throughput of various buffer designs as a function of the normalized occupancy. Note that all buffers have i/o-distance 7, but that their capacities increase from 7 for the linear buffer, via 16 for the square buffer, and 17 for the diamond buffer, to 22 for the tree buffer. As already indicated in the introduction we see that when the (normalized) occupancy of a buffer drops below, or rises above, a certain level the average throughput of that buffer decreases. Between these levels the buffers maintain a constant maximal throughput, which for all buffers in fig. 13 apparently equals $\frac{1}{2}$. It can be observed that this *plateau of maximal average throughput* becomes longer with increasing κ, or, more accurately, with increasing $\frac{\kappa}{\delta}$. Why the maximal throughput is at most $\frac{1}{2}$ will be explained below.

The curves shown in fig. 13 have been derived from theoretical considerations given in the remainder of the paper. It is, however, not difficult to define an experiment that obtains similar curves from measurements on real hardware buffers. In such an experiment one connects the output of the buffer to its input, thus creating an oscillator [8]. This oscillator is then loaded with data items to the required normalized occupancy. If one subsequently let the oscillator run free, it will oscillate at its maximum frequency from which the maximum throughput can be derived. For square buffers such experiments have been done, see [3], and the results correspond with high accuracy to the theoretical predictions [6].

Hence we define $\Omega{\downarrow}_{\Theta=\theta}(X)$ as the minimal average occupancy for which buffer X can maintain average throughput θ. Using results from [6] it can be proved that for any buffer X

$$\theta\delta(X) \quad \leq \quad \Omega{\downarrow}_{\Theta=\theta}(X) \tag{3}$$

Similarly we define $\Omega{\uparrow}_{\Theta=\theta}(X)$ as the maximum average occupancy for which the buffer can maintain average throughput θ. For this quantity it can be proved that

$$\Omega{\uparrow}_{\Theta=\theta}(X) \quad \leq \quad \kappa(X) - \theta\delta(X) \tag{4}$$

Most buffers can attain neither lower bound 3 nor upper bound 4, because of synchronization that occurs inside the buffer. This synchronization usually takes place to prevent data items that follow a short path through the buffer from overtaking data items that follow a longer path, which would manifest itself in the environment of the buffer as a violation of the FIFO-property. However, when a buffer attains any one of its extreme occupancy levels, then so do all its basic building blocks.

Theorem 6.1 *Let X be a buffer, and let $\{C_j \mid 0 \leq j < n\}$ be the set of basic building blocks from which X is composed. If, for any given average throughput θ, the buffer assumes its minimal average occupancy $\theta\delta(X)$ (or maximal average occupancy $\kappa(X)-\theta\delta(X)$), then all basic building blocks C_j assume their minimal average occupancy $\theta_j\delta(C_j)$ (or maximal average occupancy $\kappa(C_j)-\theta_j\delta(C_j)$).*

Proof. We only prove the case of minimal average occupancy. The other case can be proved by similar reasoning.

Let X run at average throughput θ and at average occupancy $\theta\delta(X)$. Furthermore, let θ_j be the corresponding average throughput and ω_j the corresponding average occupancy of C_j. Then by formula 3

$$\omega_j - \theta_j\delta(C_j) \quad \geq \quad 0 \tag{5}$$

for all j, $0 \leq j < n$. Because C_j is a basic building block we have $\delta(C_j) = 1$. Hence by lemma 6.1

$$\sum_{0\leq j<n} \theta_j\delta(C_j) \quad = \quad \sum_{0\leq j<n} \theta_j \quad = \quad \theta\delta(X)$$

By assumption $\Omega(X) = \theta\delta(X)$. Hence by lemma 6.2

$$\sum_{0\leq j<n} \omega_j \quad = \quad \Omega(X) \quad = \quad \theta\delta(X)$$

Combination of these equations gives

$$\sum_{0\leq j<n} (\omega_j - \theta_j \delta(C_j)) \quad = \quad 0 \tag{6}$$

Since the sum of nonnegative numbers is zero if and only if all numbers are zero, it follows from formulae 5 and 6 that $\omega_j = \theta_j\delta(C_j)$. So each basic building block assumes its minimal possible average occupancy. □

Next we define the elasticity as the length of the plateau of maximal average throughput in the throughput versus normalized occupancy plot, i.e.

Definition 6.3 (Elasticity) *For system X we define the elasticity $\varepsilon_\theta(X)$ at average throughput θ by*

$$\varepsilon_\theta(X) \quad = \quad \frac{\Omega{\uparrow}_{\Theta=\theta}(X) - \Omega{\downarrow}_{\Theta=\theta}(X)}{\kappa(X)}$$

□

From bounds 3 and 4 we immediately obtain an upper bound for the elasticity

$$\varepsilon_\theta(X) \quad \leq \quad 1 - \frac{2\theta\delta(X)}{\kappa(X)} \tag{7}$$

This bound implies that linear buffers are totally inelastic, when run at $\theta = \frac{1}{2}$, i.e. $\varepsilon_{\frac{1}{2}}(LBUF_l) = 0$ irrespective of the size of the buffer. For square buffers on the other hand we find that $\varepsilon_{\frac{1}{2}}(SBUF_l) \leq 1 - \frac{2l-1}{l^2}$. So for increasing l the upper bound on the elasticity of a square buffer goes to 1. The same holds for the tree buffers.

Inequality 7 also has important implications for the average throughput of a buffer. Since by definition the elasticity is at least zero, the average throughput is bounded by

$$\theta \quad \leq \quad \frac{\kappa}{2\delta} \tag{8}$$

Because all basic components have capacity and i/o-distance equal to one, it follows that $\theta \leq \frac{1}{2}$ for all basic components. This can be understood as follows. The behavior of all basic components consists of an alternation of input and output communications, and a data item accepted at an input event is passed on immediately at the following output event. When these components operate at their maximum throughput, by definition no time is wasted between successive communication events. Hence, taking the time of a communication event as the unit of time, all basic components require exactly two time units to process one item, or (which is the same) have a throughput of halve an item per time unit. Note that this result also implies that for every single-input single-output (SISO) system, like a buffer, the average system throughput is at most $\frac{1}{2}$, because the throughput of the system is limited by the average throughput of the basic components via which it communicates with its environment.

From formula 8 we also derive that for buffers with an i/o-distance larger than their capacity, like the reverse-wagging buffers RW_l, the maximum attainable average throughput drops below $\frac{1}{2}$. Since this is clearly undesirable, we shall restrict our attention to buffers whose i/o-distance is at most their capacity. Note that by theorem 5.1 all buffers in class $\mathcal{M}$ satisfy this constraint!

7 Optimal buffers

As we have seen in the previous section, the elasticity of a buffer is bounded from above, and the upper bound depends only on the capacity and the i/o-distance of a buffer. Amongst the many buffers with a given capacity and i/o-distance we are interested in the ones that are maximally elastic.

Definition 7.1 (Optimal buffer) *A buffer X is optimal, if it achieves maximal elasticity at every throughput, i.e. for all θ*

$$\varepsilon_\theta(X) = 1 - \frac{2\theta\delta(X)}{\kappa(X)}$$

For any buffer class $\mathcal{X}$ we denote the class of optimal buffers in $\mathcal{X}$ by $\heartsuit\mathcal{X}$. Hence $\heartsuit\mathcal{B}$ is the class of all optimal buffers. $\square$

Let X be an optimal buffer that can be viewed as the composition of a buffer Y and a component Z, i.e. $X = Y \parallel Z$. Then theorem 6.1 implies that buffer Y exhibits maximal elasticity for any throughput θ that it can attain in the context of Z. In particular assume that τ is the throughput of Y, when X runs at its maximum throughput θ_{max}. Then Y runs at throughput $\frac{\theta}{\theta_{max}}\tau$, when X runs at throughput $\theta \leq \theta_{max}$. So Y is almost, but not quite, an optimal buffer. The only thing that can prevent Y from being optimal is that there exists a throughput θ, $\tau \leq \theta$ for which Y is not maximally elastic. Currently the author is not aware of the existence of such buffers.

The above observation suggests that we should construct optimal buffers using optimal buffers as subcomponents. To get started we need at least one optimal buffer.

Theorem 7.1 *The one-place buffer Buf is optimal, i.e.*

$$\varepsilon_\theta(\mathit{Buf}) \quad = \quad 1 - 2\theta \tag{9}$$

Proof. Let *Buf* run at average throughput θ. Consider a single iteration of the buffer process, i.e. a period of time in which the buffer processes precisely one data item. Such a period can be divided into 4 smaller periods, viz. two periods in which the communication events $a?x$ and $b!x$ take place and two periods, one after each communication event (see fig. 14), in which the buffer is idle. Next assume that each communication event takes 1 unit of time, and that the two "idle" periods take α and β time units respectively, with $\alpha \geq 0$ and $\beta \geq 0$. Hence it follows that the average throughput $\theta = \frac{1}{2+\alpha+\beta}$. Next assume that

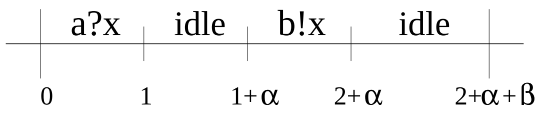


Figure 14: Timing of a single period of the one-place buffer *Buf*.

the buffer becomes occupied (vacant) the moment the input (output) action completes. Then the average occupancy ω is given by

$$(1{+}\alpha)\theta \quad = \quad \omega \quad = \quad 1-(1{+}\beta)\theta$$

Hence $\Omega{\downarrow}_{\Theta=\theta} = \theta$, when $\alpha = 0$, and $\Omega{\uparrow}_{\Theta=\theta} = 1-\theta$, when $\beta = 0$, from which formula 9 follows. $\Box$

Next we investigate under what conditions optimality is preserved by the various construction methods. For serial composition there are no additional constraints.

Theorem 7.2 *Let X and Y be optimal buffers. Then the serial composition $SER(X,Y)$ is also an optimal buffer.* $\Box$

As an immediate consequence of this theorem we have $\heartsuit\mathcal{S} = \mathcal{S}$.

Note that both wagging and multi-wagging can be seen as special cases of a more general construction which involves selecting a set of channels that cut a buffer in two parts in such a way that the input and output port are in distinct parts and subsequently inserting buffers in the selected channels. If both the original buffer and the inserted buffers are optimal and certain conditions regarding the capacities and i/o-distances of the inserted buffers are met, the result will be another optimal buffer.

Lemma 7.1 (Bisection lemma) *Let B be an optimal (κ,δ)-buffer, with structure*

$B \ =$
 proc (**in** a, **out** b)$\cdot$
 $|[$ **chan** $c[0 \mathrel{..} n) \ \rangle\!\rangle \ U(a,c) \parallel V(c,b)]|$

where all c-channels are directed from subsystem U to subsystem V. Moreover, for $0 \le j < n$, let ϕ_j be the fraction of the data that passes from U to V along channel $c[j]$, and let B_j be an optimal (κ_j,δ_j)-buffer. Then

$B' \ =$
 proc (**in** a, **out** b)$\cdot$
 $|[$ **chan** $c[0 \mathrel{..} n), d[0 \mathrel{..} n)$
 $\rangle\!\rangle \ U(a,c) \parallel \Big(\parallel_{0\le j<n} B_j([c[j], d[j]]) \Big) \parallel \ V(d,b)$
 $]|$

is an optimal buffer if and only if there exists δ' and κ' such that $\delta_j = \delta'$ and $\kappa_j = \phi_j\kappa'$, for all $0 \le j < n$. In that case buffer B' is a $(\kappa{+}\kappa', \delta{+}\delta')$-buffer.

Proof. First of all we prove that it is necessary that the capacities and i/o-distances of the inserted buffer satisfy the specified conditions. So let buffer B run at average throughput θ. To begin with note that instead of being passed from subcomponent U to subcomponent V directly, as in buffer B each item in the newly constructed buffer B' has to traverse one of the buffers B_j. If all items require the same traversal time Δt irrespective of the buffer B_j they traverse, then buffer B' will run at the same average throughput θ as B. Furthermore note that the number of items entering (or leaving) buffer B_j in a period Δt equals $\phi_j \theta \Delta t$, i.e. the throughput $\phi_j \theta$ of buffer B_j times the period. So Δt after the arrival of the first data item at buffer B_j the instantaneous occupancy of B_j equals $\phi_j \theta \Delta t$. Thereafter, the same number of data items per time slot enter and leave the buffer on average, so B_j runs at average occupancy $\phi_j \theta \Delta t$.

Now consider the situation in which buffer B' assumes its minimal average occupancy, and let the corresponding traversal time for the inserted buffers be $(\Delta t)_{min}$. Then by theorem 6.1 also buffer B_j assumes its minimal average occupancy. Because B_j is itself an optimal buffer, its minimal occupancy is given by $\phi_j \theta \delta_j$. Hence it follows that $\phi_j \theta \delta_j = \phi_j \theta (\Delta t)_{min}$. Since this holds for any buffer B_j, we derive

$$\forall_{0 \le j < n} \;\; \delta_j = (\Delta t)_{min} \tag{10}$$

Next consider the situation in which B_j assumes it maximal occupancy with corresponding transversal time $(\Delta t)_{max}$. By similar reasoning as above it follows that $\kappa_j = \phi_j \theta (\Delta t)_{max} + \phi_j \theta \delta_j$. Since we already have shown that $\delta_j = (\Delta t)_{min}$ we derive

$$\forall_{0 \le j < n} \;\; \kappa_j = \phi_j \theta ((\Delta t)_{max} + (\Delta t)_{min}) \tag{11}$$

To prove that the conditions are also sufficient we need to show that

$$\Theta(B')(\delta{+}\delta') \le \Omega(B') \le (\kappa{+}\kappa') - \Theta(B')(\delta{+}\delta')$$

This is a trivial consequence of lemma 6.2, formulae 3 and 4, and the fact that $\Theta(B') = \Theta(B)$. $\Box$

Application of the bisection lemma with U instantiated by $Split_l^k$ and V instantiated by $Merge_l^k$ yields

Theorem 7.3 *Let X and Y be optimal buffers such that $\delta(X) = \delta(Y)$ and $(l{-}1)\kappa(X) = \kappa(Y)$. Then for all k, $0 \le k < l$, the wagging composition $WAG_l^k(X,Y)$ is also an optimal buffer.*

Proof. Straight-forward, provided that component

$$\begin{array}{l} SM_l^k \; = \\ \;\; \mathbf{proc}\,(\mathbf{in}\, a, \mathbf{out}\, b)\cdot \\ \;\; |[\, \mathbf{chan}\, g, h \;\rangle\!\rangle\; Split_l^k(a,g,h) \parallel Merge_l^k(g,h,b)]| \end{array}$$

is an optimal buffer. The latter follows from results in [6, 9]. $\Box$

Among other things this theorem implies that tree buffers are optimal.

Likewise application of the bisection lemma, with U instantiated by *Msplit*$_l$ and V instantiated by *Mmerge*$_l$, yields

Theorem 7.4 *Let* $\{X_i \mid 0 \le i < l\}$ *be a collection of optimal* (κ,δ)*-buffers. Then the multi-wagging composition* $MW_l(X_0,\ldots,X_{l-1})$ *is also an optimal buffer with capacity* $l(\kappa{+}2)$ *and i/o-distance* $\delta{+}l{+}1$.

Proof. Straight-forward, provided that component

$$\begin{array}{l} \mathit{MSMM}_l\ = \\ \ \ \mathbf{proc}\,(\mathbf{in}\,a, \mathbf{out}\,b)\cdot \\ \ \ |[\ \mathbf{chan}\,d[0\,..\,l)\ \rangle\!|\ \mathit{Msplit}_l(a,d)\ \|\ \mathit{Mmerge}_l(d,b)]| \end{array}$$

is an optimal buffer. The latter follows from results in [6, 9]. □

Among other thing this theorem implies that the square buffers are optimal. Since these buffers belong to $\mathcal{M}$ but not to $\mathcal{W}$, this indicates that it is relevant to distinguish between the two classes. Another consequence of theorems 7.2, 7.3, and 7.4 is that there exists a lattice similar to that of fig. 8 in which all classes are replaced by their respective subclasses of optimal buffers.

The phenomenon of optimal buffers raises a number of interesting questions. First and foremost there is the question of the existence of these buffers.

In view of the bisection lemma which provides a very general equidistance preserving construction for optimal buffers, and given the fact that in spite of a thorough search we have not been able to construct an optimal buffer that is not equidistant, we formulate the following conjecture.

Conjecture 7.1 *Every maximally elastic buffer is equidistant.* □

A consequence of this conjecture is that we can restrict the search for optimal buffers to those with integral (κ,δ)-coordinates. So for capacity κ and i/o-distance δ both integral, we ask:

Question 7.1 *Given* $E(\kappa) \le \delta \le \kappa$ *does there exist an optimal* (κ,δ)*-buffer, i.e. does* $\heartsuit\mathcal{B}^{\kappa}_{\delta} \neq \emptyset$ *hold?*

Next, given the existence of one or more optimal buffers of a specific i/o-distance and capacity, we are interested in optimal buffers with low structural complexity. As we have seen, we distinguish two ways to achieve that: restriction of the construction methods, and reduction of the set of basic building blocks. Starting with the construction methods we ask:

Question 7.2 *Given the existence of an optimal* (κ,δ)*-buffer, does there exist a "simple" optimal* (κ,δ)*-buffer? Here simple means constructed using only a restricted set of construction methods. So we ask whether* $\heartsuit\mathcal{B}^{\kappa}_{\delta} \cap \mathcal{M} \neq \emptyset$ *or even* $\heartsuit\mathcal{B}^{\kappa}_{\delta} \cap \mathcal{W} \neq \emptyset$ *holds.*

Finally, for those (κ,δ)-coordinates for which simple optimal (κ,δ)-buffers exist, we are interested in the minimal set of the basic building blocks from which such a buffer can be constructed. So we ask:

Question 7.3 *When there exists a simple optimal (κ, δ)-buffer, what is then the smallest set of basic building blocks from which such a buffer can be constructed?*

The latter question depends on the construction methods we consider.

Definition 7.2 (Minimal indices) *Let W_δ^κ be defined as the index of the minimal class of the $\mathcal{W}$-chain that contains an optimal (κ, δ)-buffer, and let M_δ^κ be defined as the index of the minimal class of the $\mathcal{M}$-chain, i.e.*

$$\begin{aligned} W_\delta^\kappa &= \downarrow \{l \mid \heartsuit\mathcal{B}_\delta^\kappa \cap \mathcal{W}_l \neq \emptyset\} \\ M_\delta^\kappa &= \downarrow \{l \mid \heartsuit\mathcal{B}_\delta^\kappa \cap \mathcal{M}_l \neq \emptyset\} \end{aligned}$$

□

With this definition question 7.3 reduces to the computation of the minimal indices M_δ^κ and W_δ^κ . Since $\mathcal{W}_l \subseteq \mathcal{M}_l$, we have $M_\delta^\kappa \leq W_\delta^\kappa$. In particular cases in which $M_\delta^\kappa < W_\delta^\kappa < \infty$ are interesting, because then we can make a trade-off in structural complexity between simpler building blocks or simpler construction methods.

8 Contour functions

In this section we show that the questions relating to the existence and structure of maximally elastic buffers raised in the previous section can be answered by means of so-called *contour* functions. To that end a contour function will be associated with each buffer class in the lattice of fig. 8. This will be done in such a way that these contour functions themselves also constitute a lattice. Given a specific capacity κ and i/o-distance δ, inspection of the contour function of a single class enables us to determine whether that class contains optimal (κ, δ)-buffers. Comparison of contour functions of the classes in a chain allows us to determine the minimal indices.

Note that in this section we will merely define the contour functions and show how they are used. The rationale behind these definitions will be explained in the next section where we are concerned with the computation of contour functions.

Definition 8.1 (Contour function) *Let $\mathcal{X}$ be a class of buffers. Then the contour function $C(\mathcal{X}) : \mathbb{N} \to \mathbb{N} \cup \{+\infty\}$ is defined by*

$$C(\mathcal{X})(\rho) = \downarrow \{\delta \mid \heartsuit\mathcal{B}_\delta^{\rho+\delta} \cap \mathcal{X} \neq \emptyset\}$$

□

Since all buffers of class $\mathcal{S}$ have an i/o-distance equal to their capacity, the contour function of class $\mathcal{S}$ is given by

$$C(\mathcal{S})(\rho) = \begin{cases} 1 & \text{if } \rho = 0 \\ +\infty & \text{if } \rho > 0 \end{cases}$$

For the other buffer classes there is no explicit definition of the contour function. Only an algorithm for the computation of the function values can be given. This is the topic of the next section, however.

To begin with we demonstrate how the existence of an optimal (κ, δ)-buffer for class $\mathcal{X}$ can be established by inspection of $C(\mathcal{X})$.

Theorem 8.1 *Assume* $1 \leq \delta \leq \kappa$ *and let* $\mathcal{X}$ *be a class of buffers from the lattice of buffer classes defined in section 4. Then there exists an optimal* (κ, δ)*-buffer in* X*, if and only if* $C(\mathcal{X})(\kappa{-}\delta) \leq \delta$*.*

Proof. If there exists an optimal (κ, δ)-buffer, then $\heartsuit\mathcal{B}_{\delta}^{\kappa} \cap \mathcal{X} \neq \emptyset$, so by definition $C(\mathcal{X})(\kappa{-}\delta) \leq \delta$. Next consider the only if case. So assume that $C(\mathcal{X})(\kappa{-}\delta) = \gamma$, with $\gamma \leq \delta$. By definition of contour $C(\mathcal{X})$ this means that $\heartsuit\mathcal{B}_{\gamma}^{\kappa-\delta+\gamma} \cap \mathcal{X} \neq \emptyset$. So let $B \in \heartsuit\mathcal{B}_{\gamma}^{\kappa-\delta+\gamma} \cap \mathcal{X}$. If $\gamma = \delta$, then B is an optimal (κ, δ)-buffer. If $\gamma < \delta$, then $Z = SER(B, LBUF_{\delta-\gamma})$ is a (κ, δ)-buffer. Moreover, by theorem 7.2, Z is optimal. Finally Z is a member of $\mathcal{X}$, since $LBUF_{\delta-\gamma} \in \mathcal{S} \subseteq \mathcal{X}$ and $\mathcal{X}$ is closed under serial composition. $\Box$

Since the minimum over a set is at most the minimum over any of its subsets, it follows that C viewed as a function of the buffer class is antitonic, i.e. for all buffer classes $\mathcal{X}$ and $\mathcal{Y}$ function C satisfies

$$\mathcal{Y} \subseteq \mathcal{X} \quad \Rightarrow \quad C(\mathcal{X}) \sqsubseteq C(\mathcal{Y})$$

where the partial order on (contour) functions is defined by

$$C(\mathcal{X}) \sqsubseteq C(\mathcal{Y}) \quad = \quad \forall_{0 \leq \rho}\, C(\mathcal{X})(\rho) \leq C(\mathcal{Y})(\rho)$$

Hence the contour functions can be arranged into a lattice isomorphic to the lattice of buffer classes (see fig. 15). Moreover, note that the contours $C(\mathcal{W})$

$$\begin{array}{ccccccc}
C(\mathcal{B}_1) & = & C(\mathcal{M}_1) & = & C(\mathcal{W}_1) & = & C(\mathcal{S}) \\
\rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \\
C(\mathcal{B}_2) & \sqsubseteq & C(\mathcal{M}_2) & = & C(\mathcal{W}_2) & & \\
\rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \\
C(\mathcal{B}_3) & \sqsubseteq & C(\mathcal{M}_3) & \sqsubseteq & C(\mathcal{W}_3) & & \\
\rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \\
\vdots & & \vdots & & \vdots & & \\
\rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} & & \\
C(\mathcal{B}) & \sqsubseteq & C(\mathcal{M}) & \sqsubseteq & C(\mathcal{W}) & &
\end{array}$$

Figure 15: A lattice of contour functions.

and $C(\mathcal{M})$ are the limits of their corresponding chains, i.e.

$$\lim_{l\to\infty} C(\mathcal{W}_l) = C(\mathcal{W}) \qquad\qquad \lim_{l\to\infty} C(\mathcal{M}_l) = C(\mathcal{M})$$

Not only can contour functions be used to establish the existence of (κ,δ)-buffers, but also the minimal indices W^κ_δ and M^κ_δ can be computed from them, as the following theorem shows.

Theorem 8.2 *For* $1 \le \delta \le \kappa$ *the minimal indices* W^κ_δ *and* M^κ_δ *are given by*

$$\begin{aligned} W^\kappa_\delta &= \downarrow \{l \mid C(\mathcal{W}_l)(\kappa{-}\delta) \le \delta\} \\ M^\kappa_\delta &= \downarrow \{l \mid C(\mathcal{M}_l)(\kappa{-}\delta) \le \delta\} \end{aligned}$$

Proof. Let $l = \downarrow \{k \mid C(\mathcal{W}_k)(\kappa{-}\delta) \le \delta\}$. Then $C(\mathcal{W}_l)(\kappa{-}\delta) \le \delta$ and $C(\mathcal{W}_k)(\kappa{-}\delta) > \delta$, for $k < l$. So by definition 8.1 it follows that $\heartsuit\mathcal{B}^\kappa_\delta \cap \mathcal{W}_l \ne \emptyset$ and $\heartsuit\mathcal{B}^\kappa_\delta \cap \mathcal{W}_k = \emptyset$, for $k < l$. Hence $l = W^\kappa_\delta$. □

We conclude this section with the introduction of one final contour. Since all contours indicate the existence of optimal, hence equidistant, buffers, it follows that the contours belonging to a subclass of $\mathcal{M}$ must lie in the area of the (κ,δ)-space that we have called A_2 in section 5. So let us define $C_=$ as the contour that separates area A_2 from A_3 (see fig. 11), i.e. for all ρ we define

$$C_=(\rho) \quad = \quad E(\rho + \delta) \tag{12}$$

where E is defined by equation 2 in corollary 5.1. Then $C_= \sqsubseteq C(\mathcal{M})$, and the area between these contours is the part of the design space where no simple optimal buffers can be found.

9 Computation results

In this section we show how contours can be computed and present some interesting results. The basis for the computation of the various contours is given by theorems 7.2, 7.3, and 7.4. Recall that theorem 7.2 states that serial composition of optimal buffers yields another optimal buffer. So in particular serial composition of any optimal buffer with the one-place buffer *Buf* yields another optimal buffer. This way of constructing optimal buffers can succinctly be captured by the following production rule for optimal (κ,δ)-pairs,

$$(\kappa,\delta) \quad\mapsto\quad (\kappa{+}1, \delta{+}1)$$

which should be interpreted as a shorthand of

$$\heartsuit\mathcal{B}^\kappa_\delta \quad\Rightarrow\quad \heartsuit\mathcal{B}^{\kappa+1}_{\delta+1}$$

Switching to a slightly different coordinate system (ρ,δ), where $\rho = \kappa{-}\delta$, this rule becomes even simpler, viz.

$$(\rho,\delta) \quad\mapsto\quad (\rho, \delta{+}1)$$

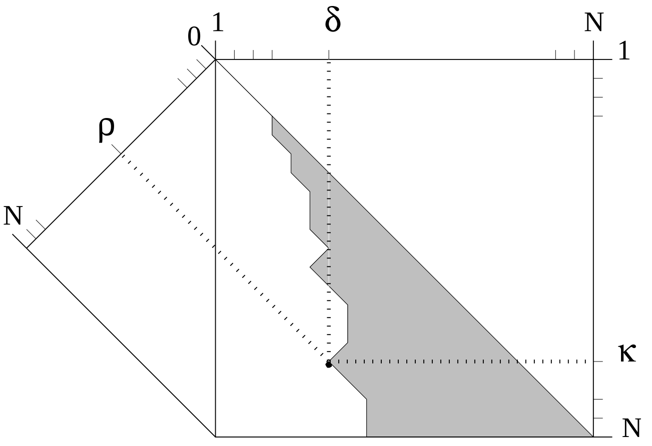


Figure 16: Design space with both coordinate systems. The grey area indicates the part of the design space for which optimal buffers of class $\mathcal{X}$ are known. It is demarcated by two lines: on the right by the diagonal $\delta = \kappa$,or equivalently $\rho = 0$, and by the contour $C(\mathcal{X})$ on the left.

So in the (ρ, δ)-coordinate system it suffices to determine for each ρ the smallest i/o-distance δ for which an optimal buffer exists. Moreover, if we can show that there exists an optimal (ρ, δ)-buffer in class $\mathcal{X}$, then it follows that there also exists an optimal (ρ, δ')-buffer in $\mathcal{X}$, for all $\delta' \geq \delta$.

The contour functions in the previous section have been defined in such a way as to exploit this fact. In particular, if we want to know all optimal (κ, δ)-buffers of class $\mathcal{X}$ in the range $1 \leq \delta \leq \kappa \leq N$, then it suffices to compute the contour $C(\mathcal{X})$ for the argument values $0 \leq \rho < N$, i.e. an initial segment of the contour of length N. Note that the number of (κ, δ)-pairs in the specified range is quadratic in the length of the contour. Figure 16 shows the design space in both coordinate systems.

Just as theorem 7.2 can be captured by a production rule, so can theorems 7.3 and 7.4. In terms of (κ, δ)-coordinates we obtain the production rules

$$\begin{array}{rcl} \begin{array}{r}(\kappa,\delta)\\ ((l{-}1)\kappa,\delta)\end{array} & \mapsto & (l\kappa{+}2, \delta{+}2) \\[1em] (\kappa,\delta) & \mapsto & (l(\kappa{+}2), \delta{+}l{+}1) \end{array}$$

Switching to the (ρ, δ)-coordinate system these become

$$\begin{array}{rcl} \begin{array}{r}(\rho,\delta)\\ ((l{-}1)\rho + (l{-}2)\delta,\delta)\end{array} & \mapsto & (l\rho + (l{-}1)\delta, \delta{+}2) \\[1em] (\rho,\delta) & \mapsto & (l\rho + (l{-}1)(\delta{+}1), \delta{+}l{+}1) \end{array}$$

Note that in the (ρ, δ)-coordinate system all rules share two properties:

1. each newly produced pair has an i/o-distance that is greater than that of the pairs from which it is produced,

2. in case two pairs are needed to produce a new one both these pairs have the same i/o-distance.

Hence given an arbitrary class of buffers closed under serial composition, represented by some contour C, and a set of construction methods, represented by a set of production rules R, the contour of the closure of that class under the construction methods can be computed by the following procedure:

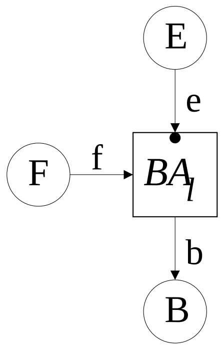
> For increasing i/o-distance δ, starting with contour C and $\delta = 1$, check for each production rule in R whether it can be applied to current set of buffers with i/o-distance δ, and if so check whether application of the rule yields a buffer with coordinates (ρ', δ') below the current contour, i.e. $\delta' < C(\rho')$. If this is indeed the case, then adjust the current contour, thereby implicitly adjusting the current set of buffers.

For a given i/o-distance the order in which the production rules are applied is irrelevant as long as all possibilities for that i/o-distance are considered. Moreover, when the contours of a chain are computed by increasing order, it is not hard to extend this procedure in such a way that also the minimal indices are obtained.

We have used the procedure sketched above to compute an initial segment of 1000 values of the contours of several buffer classes. The results are presented in the original (κ, δ)-coordinates; the (ρ, δ)-coordinates are merely used for computational purposes.

Figure 19 shows the contours of the first six classes of the $\mathcal{W}$-chain. As one can see, contour $C(\mathcal{W}_2)$ is roughly logarithmic, which means that for the vast majority of (κ, δ)-pairs there is already an optimal buffer in class $\mathcal{W}_2$. To be precise, there are only 2477 (κ, δ)-pairs in area A_2 for which there is no optimal (κ, δ)-buffer in $\mathcal{W}_2$. This is approximately 5‰ of the total number of (κ, δ)-pairs in that area. Going from class $\mathcal{W}_2$ to class $\mathcal{W}_3$ the amount of additional optimal buffers is still substantial, but thereafter only occasionally new optimal buffers appear. Moreover, the pattern of appearance, i.e. the shape of the difference curves, is quite irregular, both with respect to the number of additional optimal buffers and with respect to the capacity at which the first new optimal buffer is found. For class $\mathcal{W}_3$ the first new optimal buffer is a buffer with (κ, δ)-coordinates $(17, 7)$. This is the diamond buffer displayed in fig. 6.

Figure 20 shows the contours of the first ten classes of the $\mathcal{M}$-chain. Because the contours of class $\mathcal{M}_6$, $\mathcal{M}_8$, $\mathcal{M}_9$, and $\mathcal{M}_{10}$ are identical to those of their immediate predecessors (at least for values of ρ below 1000), only six distinct contours are displayed. For the $\mathcal{M}$-chain roughly the same observations can be made as for the $\mathcal{W}$-chain. Going from class $\mathcal{M}_2$ to class $\mathcal{M}_3$ the improvement is substantial, but thereafter only occasionally new buffers appear and the pattern seems irregular. Although difficult to infer by inspection of the plots, it can be verified that the contours of the $\mathcal{M}$-chain indeed lie below the corresponding contours of the $\mathcal{W}$-chain. The smallest buffer in $\mathcal{M}_3$ is an interesting one. It is the square buffer $SBUF_3$ with (κ, δ)-coordinates $(9, 5)$.



```
$BAM_l$ =
 proc (in $e, f,$ out $b$)·
 |[ var $x$
 ▷ ( $(e?x; b!x)^l$
    ; $(f?x; b!x)^l$
    $)^*$
 ]|
```

Figure 17: A block-wise alternating merge component.

Finally fig. 21 shows the results for the classes $\mathcal{W}$ and $\mathcal{M}$. These contours have been obtained by computing the contours of a sufficiently long initial segment of the chains for which they are the limiting cases. In addition contour $C_=$ defined in formula 12 is drawn. The area between this contour and contour $\mathcal{M}$ is "unknown territory". It contains the (κ, δ)-pairs for which it is not known whether there exists an optimal (κ, δ)-buffer. Of course, when they exist, these buffers must have complex structures, since they do not belong to $\mathcal{M}$.

The smallest (κ, δ)-pair for which we do not know whether $\mathcal{B}$ contains an optimal buffer is the pair $(13, 6)$. Such a buffer can be constructed, however, if one is willing to extend the set of basic building blocks. The building block we need is a component that merges two streams in a block-wise alternating fashion. In strict alternation, it copies data from its two input streams to the output stream in blocks of length l. Its diagram and program text are given in fig. 17. With this additional building block an optimal $(13, 6)$-buffer can be constructed as a variation of the square buffer $SBUF_3$, The resulting design is displayed in fig. 18. Note that the one-place buffers that in $SBUF_3$ connect the multi-split to the multi-merge component, have been replaced by $Split_2^0$ components. As a consequence we now need two multi-merge components and an additional BAM_3 component to merge the output streams of the multi-merge components into a single output stream. Although we shall not do so here, it can be shown by scheduling of communication events that this buffer is indeed optimal.

Having obtained this new optimal buffer we can of course apply the bisection lemma to see if we can generate more new optimal buffers. In particular there exists a cut containing 6 channels that bisects the $(13, 6)$-buffer. Since each of these channel carries the same fraction of data items, application of the bisection lemma with this cut yields the production rule

$$(\kappa, \delta) \quad \mapsto \quad (6\kappa{+}13, \delta{+}6)$$

which can be used to show the existence of an optimal (19,7)-buffer.

Similar constructions can be applied to find optimal buffers for other (κ, δ)-pairs in the unknown territory. We shall not pursue this line of investigation, however, because the unknown territory is already very small. For contours

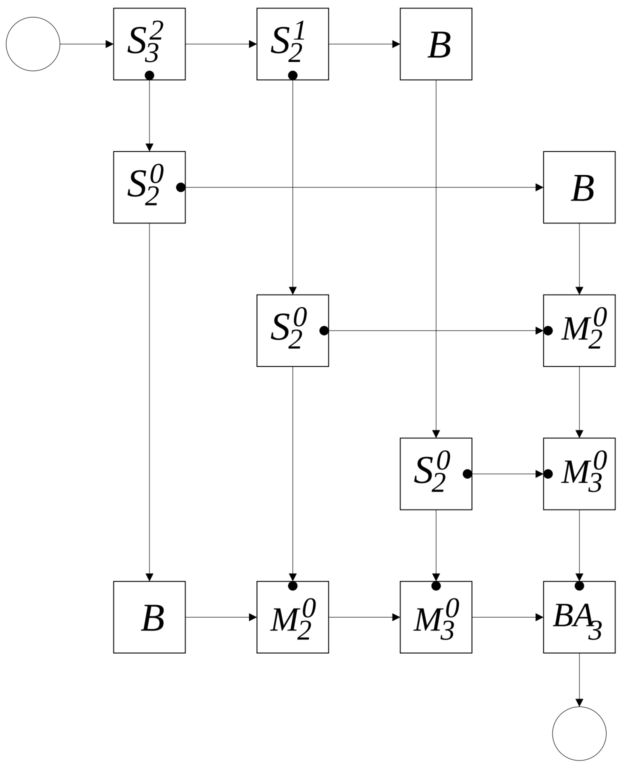


Figure 18: Optimal $(13, 6)$-buffer.

truncated at length 1000 it contains only 1478 (κ, δ)-pairs, which is roughly 3‰ of the entire area A_2. As we increase the contour length the proportion of the unknown territory becomes vanishingly small.

Summarizing, we find that for almost all (κ, δ)-pairs there exists an optimal buffer and that in the overwhelming majority of the cases there is an optimal buffer with a very simple structure, i.e. an optimal buffer that belongs to class $\mathcal{W}_2$. Moreover, all these buffers are equidistant by construction.

10 Conclusions

This paper is concerned with the high-level design of maximally elastic buffers. We call the designs high-level, because the buffers are described by small programs written in a high-level language. Although we have not chosen any particular language, it is a trivial exercise to reformulate the designs in existing hardware description languages such as Tangram [1] or Balsa [4], from which there is an automated route to silicon. Also specification and performance metrics are given by quantities defined in terms of these program texts. In particular we have avoided detailed timing considerations. Nevertheless, we have obtained accurate predictions of the elasticity of our designs, on which high-level design decisions can be made.

We have classified buffers according to their structural complexity. For this purpose a taxonomy based on allowed building block and construction methods. The construction methods appear to be well-suited, because there exists a class, viz. class $\mathcal{M}$, that on the one hand is sufficiently large to contain buffers that perform optimal in terms of metrics like elasticity and throughput for all possible specifications, but on the other hand excludes most buffers that for some reason are less desirable. Furthermore, the taxonomy is fine-grained enough to distinguish a lot of structural detail within class $\mathcal{M}$.

The classification effort shows that for almost any combination of desired capacity and i/o-distance a maximally elastic buffer with a very simple structure is available. Indeed the tree buffers, which provide the highest capacity for a fixed i/o-distance and therefore are the most elastic buffers of all, are also amongst the most simple buffers. Occasionally designs incorporating square buffers perform better. So for practical designs it is best to consider only a few families of optimal buffers that have the property that the elasticity goes to 1 as the capacity increases. Furthermore it turns out that all maximally elastic buffers found are equidistant, which we suspect to be a fundamental property.

Although the classification is not complete, the part of the design space for which it is not known whether the classes of the taxonomy contain maximally elastic buffers is vanishingly small. From a theoretical point of view it would be satisfying to explore this unknown territory further, but it is certainly not of practical interest.

It should be stressed that we have not merely presented a taxonomy, but in fact a framework to create taxonomies. This framework is open-ended in the sense that we can easily extend the given taxonomy by the introduction of additional basic building blocks and additional construction methods. As an example we have shown that adding block-alternating mergers makes it possible to construct new optimal buffers, but there are other possibilities as well. E.g. one can consider basic building blocks with storage capacity 2 and i/o-distance 1. This will result in smaller contours, i.e. for the same capacity optimal buffers with smaller i/o-distance can be constructed. Moreover, since such building blocks are capable of performing input and output events simultaneously, it also becomes possible to construct buffers that can run at throughput 1. Another possible extension is the introduction of three-way splitters and mergers. This enables the design of cubic buffers. An initial investigation into these extensions has been done in [9], where also the diamond buffer has been presented for the first time.

11 Acknowledgements

I would like to thank Peter Hilbers and Johan Lukkien for many fruitful discussions and their comments on draft versions of this paper.

References

[1] K. v. Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In

Proc. European Conference on Design Automation (EDAC), pages 384–389, 1991.

[2] E. Brunvand. Low latency self-timed flow-through FIFOs. In W. J. Dally, J. W. Poulton, and A. T. Ishii, editors, *Advanced Research in VLSI*, pages 76–90. IEEE Computer Society Press, 1995.

[3] J. Ebergen. Squaring the FIFO in GasP. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 194–205. IEEE Computer Society Press, Mar. 2001.

[4] D. Edwards and A. Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, 45(1):12–18, 2002.

[5] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[6] R. H. Mak. Design and performance analysis of buffers: a constructive approach. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 137–148, Apr. 2002.

[7] C. E. Molnar, I. W. Jones, B. Coates, and J. Lexau. A FIFO ring oscillator performance experiment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 279–289. IEEE Computer Society Press, Apr. 1997.

[8] C. E. Molnar, I. W. Jones, W. S. Coates, J. K. Lexau, S. M. Fairbanks, and I. E. Sutherland. Two FIFO ring performance experiments. *Proceedings of the IEEE*, 87(2):297–307, Feb. 1999.

[9] A. M. Smets. Design of optimal buffers. Master's thesis, Technische Universiteit Eindhoven, 2002.

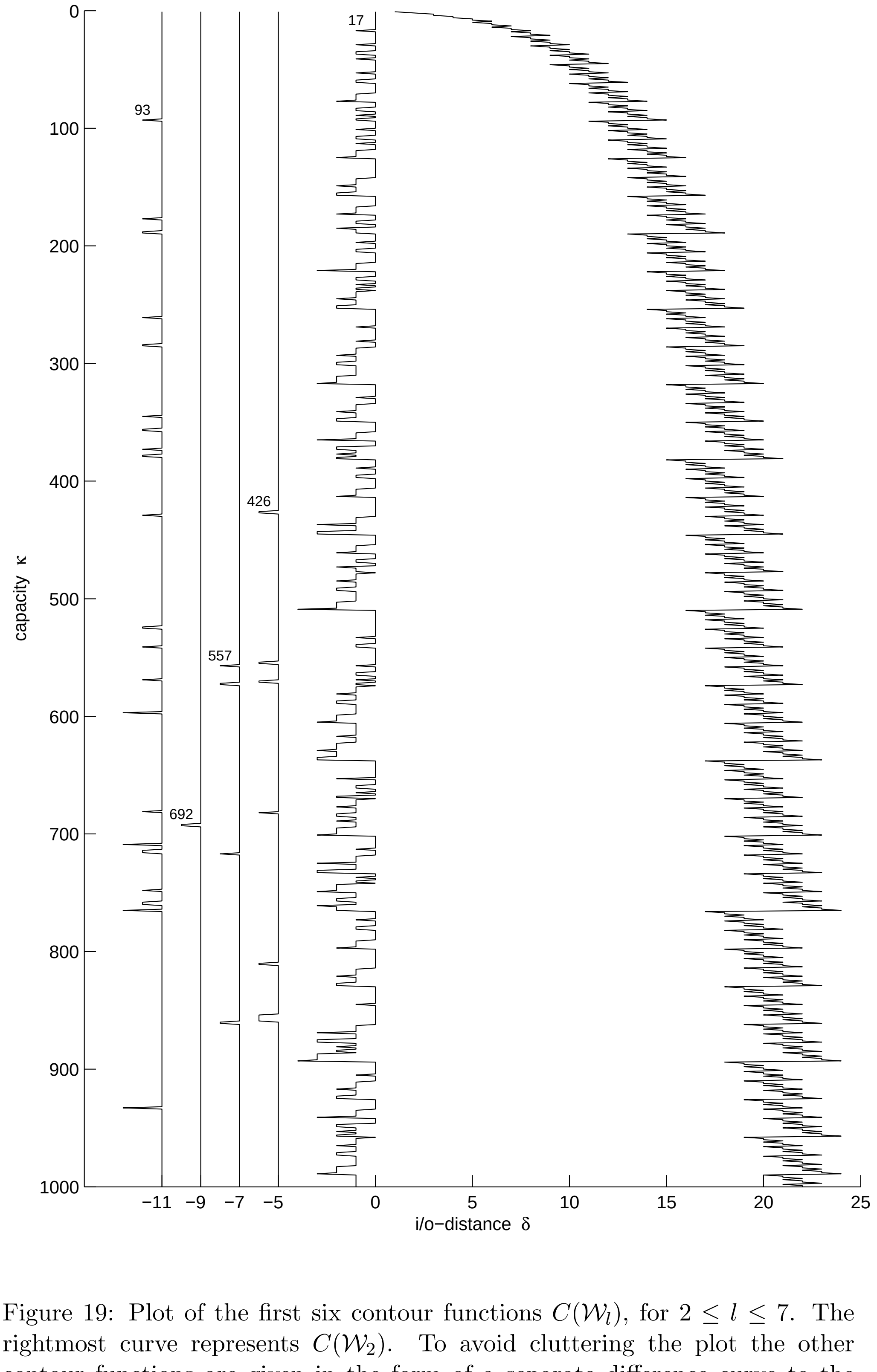


Figure 19: Plot of the first six contour functions $C(\mathcal{W}_l)$, for $2 \leq l \leq 7$. The rightmost curve represents $C(\mathcal{W}_2)$. To avoid cluttering the plot the other contour functions are given in the form of a separate difference curve to the left of the δ=0 axis. From right to left the curves $C(\mathcal{W}_3)-C(\mathcal{W}_2)$ up to $C(\mathcal{W}_7)-C(\mathcal{W}_6)-11$ are plotted. We see that $C(\mathcal{W}_3)$ differs substantially from $C(\mathcal{W}_2)$, but thereafter the differences are minute, until $C(\mathcal{W}_7)$.

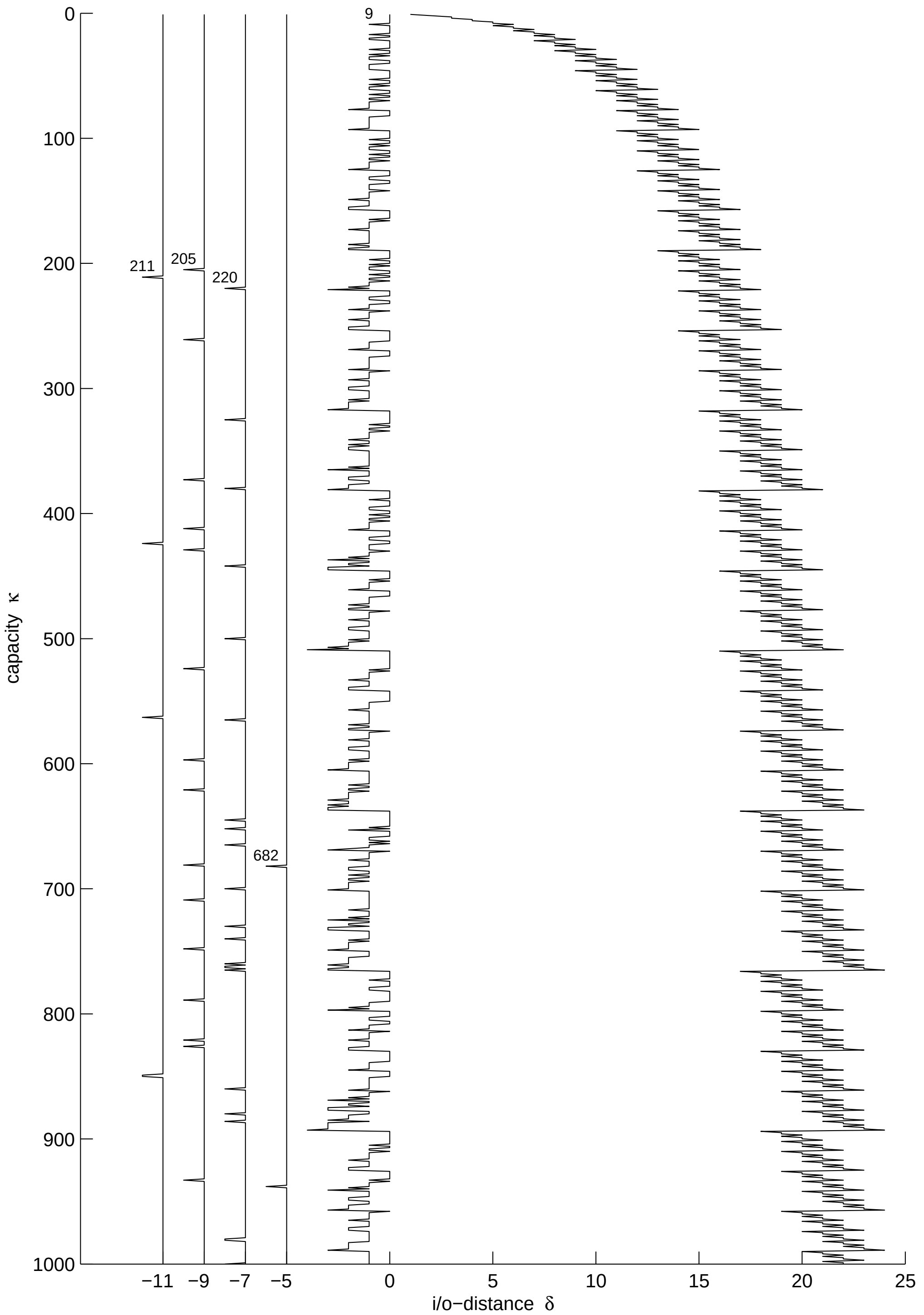


Figure 20: Plot of the contour functions $C(\mathcal{M}_l)$, for $l = 2, 3, 4, 5, 7, 11$. The rightmost curve represents $C(\mathcal{M}_2) = C(\mathcal{W}_2)$. The other contour functions are given in the form of a separate difference curve to the left of the δ=0 axis. From right to left the curves $C(\mathcal{M}_3)-C(\mathcal{M}_2)$, $C(\mathcal{M}_4)-C(\mathcal{M}_3)-5$, $C(\mathcal{M}_5)-C(\mathcal{M}_4)-7$, $C(\mathcal{M}_7)-C(\mathcal{M}_5)-9$, and $C(\mathcal{M}_{11})-C(\mathcal{M}_7)-11$ are plotted.

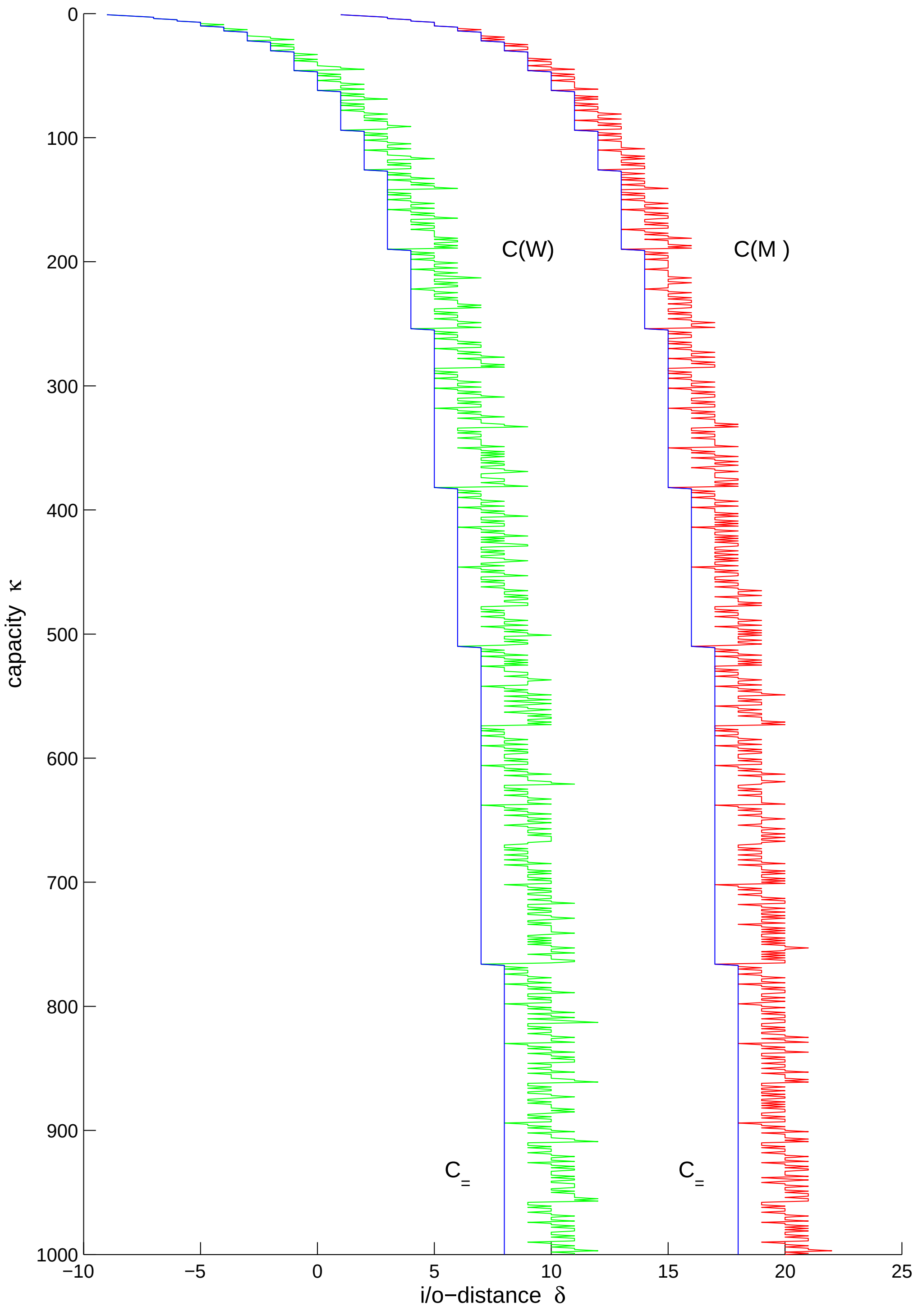


Figure 21: Plot of the contour functions $C(\mathcal{M})$ and $C(\mathcal{W})$, the latter shifted by -10. Also contour $C_=$ to the left of which no equidistant buffers exist is drawn. Note that the gap between this contour and the other two is very small indeed. For any capacity the gap involves at most 4 i/o-distances.