In the Proceedings of the 2000 International Conference on Circuits and Systems (ISCAS 2000)

THE CBP PARAMETER — A USEFUL ANNOTATION TO AID BLOCK-DIAGRAM COMPILERS FOR DSP

Shuvra S. Bhattacharyya†, *Praveen K. Murthy*‡

†Dept. of Electrical and Computer Engr., and
Institute for Advanced Computer Studies
University of Maryland, College Park
ssb@eng.umd.edu

‡Angeles Design Systems
San Jose, CA
pmurthy@angeles.com

ABSTRACT

Memory consumption is an important metric during software synthesis from block-diagram specifications of DSP applications. Conventionally, no assumption is made about when during the execution of a functional block (*actor*), the associated data values (*tokens*) are actually consumed and produced. However, we show in this paper that it is possible to concisely and precisely capture key properties pertaining to the relative times at which tokens are produced and consumed by an actor. We show this by introducing the *consumed-before-produced (CBP)* parameter, which provides a general method for characterizing the token transfer of an actor. Good bounds on the CBP parameter can aid a block-diagram compiler in performing more aggressive optimizations for reducing buffer sizes on the edges between actors. We formally define the CBP parameter; derive some useful properties of this parameter; illustrate how the value of the parameter can be derived by examining in detail the multirate FIR filtering operation; and examine CBP parameterizations for several other practical DSP actors.

1. INTRODUCTION

Dataflow is a natural model of computation to use as the underlying model for a block-diagram language for designing digital signal processing (DSP) systems. Functional blocks in a dataflow-based, block-diagram language correspond to actors (vertices) in a dataflow graph, and the connections correspond to directed edges between the actors. These edges not only represent communication channels, conceptually implemented as FIFO queues, but also establish precedence constraints. An actor fires in a dataflow graph by removing tokens from its input edges and producing tokens on its output edges. The stream of tokens produced this way corresponds naturally to a discrete time signal in a DSP system. In this paper, we consider a subset of dataflow called *synchronous dataflow* (SDF) [7]. In SDF, each actor produces and consumes a fixed number of tokens, and these numbers are known at compile time. In addition, each edge has a fixed number of initial tokens, called *delays*. SDF is used in numerous commercial and research-oriented design tools for DSP, such as COSSAP [10] from the Aachen University of Technology (now from Synopsys), GRAPE [6] from K. U. Leuven, Ptolemy [4] from U. C. Berkeley, DSP Canvas from Angeles Design Systems, SPW from Cadence, and ADS from Hewlett Packard.

Given an SDF edge e, we denote the source actor, sink actor, and delay of e by $src(e)$, $snk(e)$, and $del(e)$. Also, $prod(e)$ and $cons(e)$ denote the numbers of tokens produced onto e by $src(e)$ and consumed from e by $snk(e)$. A critical objective when synthesizing software from an SDF specification is minimization of the memory requirements of the target implementation. In this paper, we present a novel method for encapsulating actor properties that enable the application of powerful optimization techniques for buffer memory minimization.

2. THE CBP PARAMETER

We say that a token is *consumed* from a memory buffer when the last access to it from the buffer is completed. Also, for a given invocation I of an SDF actor, a given input edge of the actor, and a given output edge, we represent the number of tokens produced and consumed during the time interval $[0, t]$ by $p_I(t)$ and $c_I(t)$, respectively (time 0 corresponds to the starting time of the actor invocation, and t must be less than or equal to the completion time)[1]. If I is understood from context, we may drop the subscript I, and simply write $p(t)$ and $c(t)$.

Definition 1: *Suppose that A is an actor in an SDF graph, α_i is an input edge of A, and α_o is an output edge of A. The **CBP parameter** of the pair (α_i, α_o) for the given implementation of A, denoted $\mathrm{CBP}_A(\alpha_i, \alpha_o)$, is intended to specify the best (largest) known lower bound on $c_I(t) - p_I(t)$.*

Thus if a CBP parameter has been specified by the actor programmer for (α_i, α_o), then an SDF compiler can assume that $(c_I(t) - p_I(t)) \geq \mathrm{CBP}_A(\alpha_i, \alpha_o)$ for any invocation I, and for all valid t. If no CBP parameter has been specified, a worst-case CBP parameter $\mathrm{CBP}_A(\alpha_i, \alpha_o) = -prod(\alpha_o)$ must be assumed, or the actor source code must be analyzed to try to determine a tighter bound. Such source code analysis is beyond the scope of this paper, and we simply assume the worst case bound $\mathrm{CBP}_A(\alpha_i, \alpha_o) = -prod(\alpha_o)$ when the actor programmer has not specified a CBP value. Note that we always have $\mathrm{CBP}_A(\alpha_i, \alpha_o) \leq 0$ since $c_I(0) = p_I(0) = 0$. Thus, the CBP parameter can be viewed as the lower bound on $c_I(t) - p_I(t)$ that is *closest to 0* among all known lower bounds.

Higher CBP values give more flexibility in buffer sharing, as will be demonstrated below, and thus, it is advantageous to specify a tight lower bound as the CBP. Due to the regularity of DSP computations, the computation of tight CBP bounds is often straightforward.

As a simple example of a tight CBP bound, consider the "block addition" actor illustrated in Fig. 1(a). This actor inputs a block of N tokens on each output, and outputs a block of N

1. By *time* here, we mean elapsed time during execution of an implementation of the associated dataflow graph.

tokens such that each i th value in the output block is the sum of the i th values in the input blocks. If the *Motorola DSP56000* code outlined in Fig. 1(b) is used to implement this actor, then it is apparent that the ith token read from each input edge is always consumed before the i th output is computed. As a result, the total number of tokens produced $p(t)$ at any given time (during the execution of a particular invocation of the actor) can never be greater than the number of tokens $c(t)$ consumed until that time from any single input edge. Thus, we are guaranteed that $c_I(t) - p_I(t) \geq 0$ and $\mathrm{CBP}_A(\alpha_i, \alpha_o) = 0$ is a valid choice.

This knowledge that $c_I(t) - p_I(t) \geq 0$ allows us to fully overlay the output buffering for the code segment shown in Fig. 1(b) with either of the two input buffers. For example, if we initialize the output write pointer to the beginning of the input buffer that starts at address *inbuf1*, we are guaranteed by the relation $\mathrm{CBP}_A(\alpha_i, \alpha_o) = 0$ that the output write pointer will never "overtake" the input read pointer associated with the *inbuf1* buffer. Code for the block addition actor that incorporates this input/output overlaying is shown in Fig. 1(c). We refer to this form of buffer sharing — in which an input channel and output channel of the same actor share the same physical buffer — as **buffer merging**.

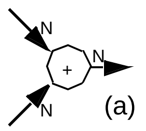
Definition 2: *Since* $\mathrm{CBP}_A(\alpha_i, \alpha_o)$ *always lies in the range*

$$\{-prod(\alpha_o), -prod(\alpha_o)+1, -prod(\alpha_o)+2, \ldots, 0\},$$

the ratio of the absolute value of the CBP parameter to $prod(\alpha_0)$ *is a useful gauge of the degree to which a given actor implementation facilitates the consolidation of an input/output buffer pair. Thus, we define the* ***CBP efficiency*** *of an actor implementation* A *with respect to the ordered pair* (α_i, α_o) *as the sum*

$$1 + \frac{\mathrm{CBP}_A(\alpha_i, \alpha_o)}{prod(\alpha_o)}. \qquad (1)$$

Clearly, the CBP efficiency is always a non-negative rational number that lies in the closed interval $[0, 1]$. For the example of Fig. 1, we have a CBP efficiency of unity, or 100%, since $\mathrm{CBP}_A(\alpha_i, \alpha_o) = 0$. In Section 3, we will see an example of an actor that can have an infinite range of different CBP efficiencies depending on its functional parameters.



(b)

```
move #inbuf1, r0
move #inbuf2, r5
move #outbuf, r4
          move x:(r0)+, x0
          move y:(r5)+, a
do #N, LOOPEND
          add x0, a
          move x:(r0)+, x0
          move y:(r5)+, a
move a, y:(r4)+
LOOPEND:
```

(c)

```
move #inbuf1, r0
move #inbuf2, r5
move r5, r4
          move x:(r0)+, x0
          move y:(r5)+, a
do #N, LOOPEND
          add x0, a
          move x:(r0)+, x0
          move y:(r5)+, a
move a, y:(r4)+
LOOPEND:
```

Fig 1. A block addition actor that illustrates the derivation and exploitation of a tight CBP bound. Here, the sets of indented operations are performed in parallel.

We conclude this section with a simple fact, which is proven in [3], concerning CBP parameters that is useful in deriving CBP parameters for specific actor implementations.

Theorem 1: *Suppose that* A, α_i, *and* α_o *are as in Definition 1. Given an invocation* A_I *of* A, *let* t_i *denote the time (relative to the beginning of* A_I*) at which the* i*th output token of* A_I *is produced, for* $i = \{1, 2, \ldots, prod(\alpha_0)\}$. *Also, define* $t_0 \equiv 0$, *let* T_I *denote the duration of* A_I, *and let* x *be a non-positive integer. Then, if* $c_I(t_i) - p_I(t_i) \geq x$ *for all* $i \in \{0, 1, 2, \ldots, prod(\alpha_0)\}$, *we are guaranteed that* $c_I(t) - p_I(t) \geq x$ *for all* $t \in [0, T_I]$.

3. MULTIRATE FIR FILTERS

A multirate FIR filter actor, shown in Fig. 2(a), performs a sample rate conversion of an arbitrary rational factor u/d along with an FIR ("finite impulse response") filtering operation. Functionally, it is equivalent to the structure shown in Fig. 2(b), which contains a conventional upsampler, downsampler, and an appropriately designed single-rate FIR filter.

The "core" of Fig. 2(b) is the FIR actor, which effectively forms an inner product of a vector of adjacent data samples with a vector of constant coefficients. In this discussion, we consider the class of multirate FIR filter implementations in which the vector of "past" (previously consumed) data samples involved in the FIR inner product is maintained in a separate memory buffer that is internal to the multirate FIR actor. Motivation for this class of implementations is elaborated on in [3].

Fig. 2(c) illustrates the production and consumption activity that occurs in a multirate FIR filter. In this illustration, u and d are taken to be 3 and 2 respectively, and the order of the filter is taken to be 4. The order O_M of the filter is the number of adjacent samples from the input of the FIR block of Fig. 2(b) that are involved in the computation of each output sample.

The first row of symbols (zeros and x_is) shown in Fig. 2(c) represents a stream of data samples processed in a given invocation M_I of the multirate FIR filter M. The zeros shown in the stream are inserted by the logical upsampler block (labeled "$\uparrow$u") in Fig. 2(b). The upsampler effectively interleaves $u-1$ zeros between each pair of input tokens.

Each x_i represents the input token value at offset i relative to the beginning of M_I. Thus, x_0 and x_1 are, respectively, the first and second token values consumed by M_I; x_2 is the first token value consumed by M_{I+1}, the next invocation of M; and x_{-1} is last token value consumed by M_{I-1} (if M_I is the first invocation of M — that is, $I = 1$ — then x_{-1} is part of the ini-

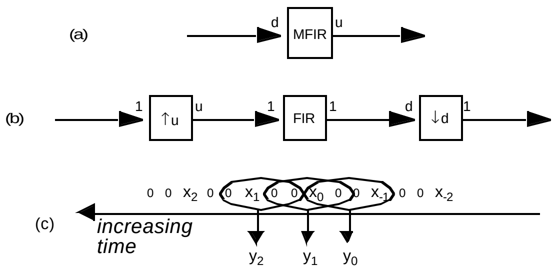


Fig 2. A multirate FIR filter actor that we use to illustrate the derivation of CBP parameters.

tial state of M). Similarly, y_0, y_1, y_2 represent the first, second and third token values produced by M_I.

The three overlapping ovals in Fig. 2(b) group sets of $O_M = 4$ adjacent token values in the upsampled stream with the actor output tokens that are derived from them. Assuming that the output tokens y_0, y_1, y_2 are produced according to their logical ordering (as they usually are) — that is, as long as

$$(i < j) \Rightarrow (y_i \text{ is produced before } y_j) \text{ —} \tag{2}$$

we have the following observations from Fig. 2(c):

$$c(t_0) \geq 1 \text{ , } c(t_1) \geq 1 \text{, and } c(t_2) \geq 2 \text{ ,} \tag{3}$$

where t_0, t_1, t_2 respectively denote the times at which y_0, y_1, y_2 are produced.

It follows that for the multirate FIR filter illustrated in Fig. 2(c) ($u = 3$, $d = 2$), we have that

$$c(t) - p(t) \geq min(\{(1-1), (1-2), (2-3)\}) = -1 \text{ .} \tag{4}$$

Thus, for this example, a CBP parameter of -1 is feasible.

To generalize this analysis, we observe that for arbitrary u , d and O_M, the grouping of values in the upsampled stream with corresponding output tokens has the following characteristics: each pair of adjacent input tokens x_i and x_{i+1} is separated by exactly $(u-1)$ zero-valued samples; each output token is derived from a "window" of O_M adjacent values in the upsampled stream; each successive "window" of O_M samples is shifted d units (token positions) to the left (towards increasing time) with respect to the previous window; and the first window — the window associated with output y_0 — has x_0 as its leftmost sample.

Given these characteristics along with the assumption of (2), it is easily seen that if $c(t)$ and $p(t)$ denote the total number of tokens consumed during the first t time units (during the interval $[0, t]$) in the execution of a given invocation M^* of a multirate FIR filter actor, then we have

$$(p(t)-1)d \leq (c(t)-1)u + u - 1 \text{ ,} \tag{5}$$

which is equivalent to

$$p(t) < 1 + \frac{c(t)u}{d} \text{ .} \tag{6}$$

Now suppose, as above, that $t_0, t_1, \ldots, t_{u-1}$ denote the times at which outputs $y_0, y_1, \ldots, y_{u-1}$, respectively, are produced. Then, clearly for all i,

$$p(t_i) = (i+1) \text{ ,} \tag{7}$$

and combining this with (6) yields

$$i + 1 < 1 + \frac{c(t_i)}{d}u \text{ ,} \tag{8}$$

which is equivalent to

$$c(t_i) > \frac{di}{u} \text{ .} \tag{9}$$

Thus, combining (9) and (7), we have

$$c(t_i) - p(t_i) > \frac{di}{u} - (i+1) \text{ .} \tag{10}$$

From (10) and the restriction that

$$i \in \{0, 1, \ldots, (u-1)\} \tag{11}$$

(since M^* produces u output tokens), it follows that

$$c(t_i) - p(t_i) \geq 0 \text{ whenever } d \geq u \text{ .} \tag{12}$$

On the other hand, if $u > d$, then the LHS of (9) attains its minimum value over the range (11) when $i = (u-1)$. Thus, for $u > d$, we have

$$c(t_i) - p(t_i) > \frac{d(u-1)}{u} - u \text{ ,} \tag{13}$$

which is equivalent to

$$c(t_i) - p(t_i) > (d-u) - \frac{d}{u} \text{ .} \tag{14}$$

Since $u > d$, and both $c(t_i)$ and $p(t_i)$ must be integers, it follows that

$$c(t_i) - p(t_i) \geq (d-u) \text{ whenever } u > d \text{ .} \tag{15}$$

From Theorem 1, we can extend the conclusions of (12) and (15) to arbitrary values of t. That is, throughout any invocation of M^*, we have that

$$(d \geq u) \Rightarrow (c(t) - p(t) \geq 0) \text{ and}$$
$$(u > d) \Rightarrow (c(t) - p(t) \geq (d-u)) \text{ .} \tag{16}$$

In summary, we have established the following result.

Theorem 2: *If in-place computation is not used and output tokens are produced according to their logical ordering, then a rational, multirate FIR filter can be implemented with the CBP parameter derived from the following relations:*

$$\text{CBP} = \begin{cases} 0 \text{ if } (u \leq d) \\ (d-u) \text{ if } (u > d) \end{cases} \text{ ,} \tag{17}$$

where u/d is the reduced form of the output-to-input sample-rate conversion ratio.

4. OTHER USEFUL ACTORS

To emphasize that CBP parameters may vary widely depending on the particular actors under consideration, and to juxtapose CBP results for a variety of practical examples, Table 1 summarizes CBP efficiencies that we have derived for several actors. This includes the examples that we have discussed in Sections 2 and 3, as well a number of additional actors that can be found in the Ptolemy [4] DSP libraries. For actors that have multiple inputs or multiple outputs, we have listed the *maximum* CBP efficiency over all input/output combinations.

5. RELATED WORK

Buffer sharing has been studied extensively in general-purpose contexts (e.g., see [5]). Our CBP-based buffer merging approach differs in that it is streamlined to exploit the structure and access patterns commonly found in DSP library modules.

The CBP parameter plays a role that is somewhat similar to the array index distances derived in the in-place memory management strategies of Cathedral [13], which applies to nested loop constructs in Silage. The CBP-based buffer merging approach presented in this paper is different from the approach of [13] in that it is specifically targeted to the high regularity and modularity present in SDF graph implementations (at the expense of decreased generality). In particular, the overlapping of SDF input/output buffers by systematically applying CBP analysis does not emerge in any straightforward way from the more general techniques developed in [13]. Our form of buffer merging is especially well-suited for incorporation with the SDF vectorization techniques (for minimizing context-switch overhead) developed at the Aachen University of Technology [10] since the absence of nested loops in the vectorized schedules [11] allows for more flexible merging of input/output buffers [8]. Buffer merging is also compatible with buffer access enhancements such as polyphase filter implementation [12], and cyclo-static dataflow specification [2].

Lower bounds on memory requirements of SDF specifications have been derived [1], assuming that each buffer is assigned to separate storage. Exploring the incorporation of buffer merging opportunities, as enabled by the CBP parameter, into this analysis is a useful direction for further investigation.

6. SUMMARY

The CBP parameter provides a concise and precise method for encapsulating a library-developer's knowledge of DSP software functionality in a manner that is valuable for synthesis tools. Our previous work has demonstrated the ability to systematically exploit pre-specified CBP parameters to significantly reduce memory requirements in software implementations [8]. For example, we observed an improvement exceeding 20% over previous methods for a practical, sample-rate conversion application. By focusing on the multirate FIR filter, we have demonstrated analysis techniques that can be used to derive tight CBP parameters from an understanding of a DSP library function or analysis of code that implements the function. We have also reported general expressions for the CBP efficiencies of several additional practical DSP building blocks, which were obtained by analyzing implementations in the DSP libraries provided within the Ptolemy design environment [4]. A useful direction for further study is the investigation of tools to help automate the derivation of tight CBP parameters.

ACKNOWLEDGEMENTS

A portion of this research was sponsored by the US National Science Foundation (MIP9734275) and Northrop Grumman Corp. We are also grateful to the anonymous reviewers for their useful comments.

7. REFERENCES

[1] M. Ade, R. Lauwereins, and J. A. Peperstraete, "Data Memory Minimisation for Synchronous Data Flow Graphs Emulated on DSP-FPGA Targets," *Design Automation Conference*, 1997.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-Static Dataflow," *IEEE Transactions on Signal Processing*, vol. 44, pp. 397-408, 1996.

[3] S. S. Bhattacharyya, P. K. Murthy, *The CBP Parameter — a useful Annotation to Aid SDF Compilers,* Tech. Rpt. UMIACS-TR-99-56, Univ. of Maryland, Sep. 1999.

[4] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, "Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems", *Intl. Journal of Computer Simulation*, Jan. 1995.

[5] J. Fabri, *Automatic Storage Optimization*, UMI Rsch. Press, 1982.

[6] R. Lauwereins, M. Engels, J. A. Peperstraete, E. Steegmans, J. V. Ginderdeuren, "GRAPE: A CASE Tool for Digital Signal Parallel Processing," *IEEE ASSP Magazine*, vol. 7, 1990.

[7] E. A. Lee, D. G. Messerschmitt, "Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing," *IEEE Trans. on Computers*, Feb., 1987.

[8] P. K. Murthy, S. S. Bhattacharyya, "A Buffer Merging Technique for Reducing Memory Requirements of Synchronous Dataflow Specifications," in *Proc. Intl. Symp. on System Synthesis*, November, 1999.

[9] P. K. Murthy, S. S. Bhattacharyya, *Shared Memory Implementations of Synchronous Dataflow Specifications using Lifetime Analysis Techniques*, Tech. Rpt. UMIACS-TR-99-32, Univ. of Maryland, June 1999.

[10] S.Ritz, M.Pankert, H.Meyr. "Optimum vectorization of scalable synchronous dataflow graphs." *Intl. Conf. on Application Specific Array Processors*, 1993.

[11] S. Ritz, M. Willems, H. Meyr, "Scheduling for Optimum Data Memory Compaction in Block Diagram Oriented Software Synthesis," *Intl. Conf. on Acoustics, Speech, and Signal Processing*, 1995.

[12] P.P. Vaidyanathan. *Multirate Systems and Filter Banks*. Prentice Hall, 1993.

[13] I.Verbauwhede, F.Catthoor, J.Vandewalle, H.De Man. "In-place memory management of algebraic algorithms on application specific ICs." *J. of VLSI Signal Processing*, pp. 193–200, 1991.

Table 1. CBP efficiencies for several useful actors. Further details can be found in [3].

Actor	Relevant parameters	(Max.) CBP efficiency
Block addition	Block size N.	1 (100%)
Multirate FIR filter	Sample rate conversion ratio a/b (in reduced form).	1 if $a < b$; b/a if $a > b$.
Chop	Production param. N_w; consumption param. N_r; offset Δ; *past-inputs* (boolean).	1 if $\Delta \geq 0$; $1 + \Delta/N_w$ if $((\Delta < 0)$ **and** $(\textit{past-inputs} = \text{false}))$; $1 + \frac{min(\{N_r - N_w, 0\})}{N_w}$ if $((\Delta < 0)$ **and** $(\textit{past-inputs} = \text{true}))$.
Autocorrelation	Inputs to average N_{avg}; lags to estimate N_{lag}.	$\frac{1 + N_{\text{lag}}}{2N_{\text{lag}}}$
Block lattice	Block size N_B; filter order N_o.	1
Commutator	Number of input ports k; block size N_B.	$\frac{1}{k}$
Distributor	Number of output ports k; block size N_B.	1