No. 3 August 14, 1990

Capsim Application Note

Memory Management in Capsim Simulations

Introduction

In this application note, we will describe how memory usage grows in Capsim simulations that use multi-rate sampling and vector processing. Techniques for managing memory usage in long simulations will be introduced.

Buffers in Capsim

Capsim uses buffers to connect blocks and to pass samples between blocks. These buffers are actually linked lists and the elements are called cells. Each cell represents either a *char,integer, float* or any other data type. When a star generates samples, it does so by placing them on its output buffer through the call *it_out(i)* followed by *sampleOut(0)* = value. In this case, the pointer to buffer i is incremented by a cell and now points to an empty but existing cell. The statement, *sampleOut(0)=value*, copies *value* into the cell. *sampleOut* is the buffer name. Now, initially Capsim allocates 128 cells for each buffer. Therefore, a simple increment of the buffer pointer is sufficient. However, if there are no more cells available, usually because the star connected to the buffer did not consume all samples in the buffer, Capsim will allocate another 128 cells. Hence, the size of the buffer grows. Under many circumstances, this is will take place since the star connected to the source will consume all samples available on its input buffer before returning control to Capsim.

There are, however, many situations in multi-rate simulations, where, for very long simulations, instability may result. In this case, a buffer may overflow in the sense that it requires more memory than the Capsim kernel will allow it to have. A typical case is illustrated in Fig. 1. The data source produces, say, 128 samples each time it is visited. Normally all sources produce 128 samples at a time. We will make this assumption in the following discussion. The *coder* processes all 128 samples on its input and produces N*128 samples (let N=8). These samples are input to an adder. Another source, the *gauss* star, is also connected to the adder. When it is visited, it produces 128 samples. Now, the adder star looks at its input buffers and determines the minimum number of samples available. In this case, buffer 0 from the *coder*

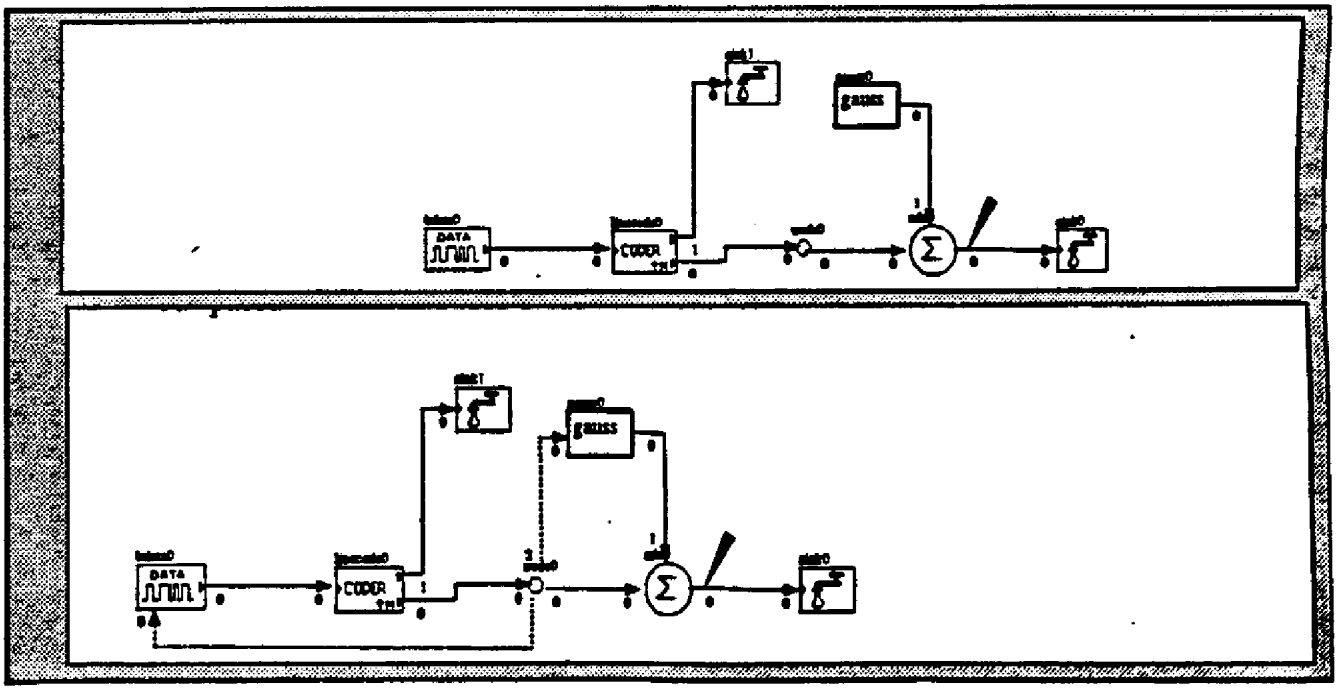

Figure 1. Buffer growth control using pacers

star contains 8*128 samples , while the buffer from *gauss* contains 128 samples. Thus, the adder will process only 128 samples. It then outputs the 128 samples to the *sink* star which simply absorbs them. Buffer 0 at the adder input now has 7*128 samples. At this stage, the *data* star is visited again by the scheduler and 128 more samples are generated. The coder process all 128 samples and outputs 8*128 samples on its output buffer. But this buffer already has 7*128 samples. So Capsim allocates another 8*128 cells. The adder can only process 128 of these since the *gauss* star only produces 128 samples. Obviously, we have an unstable situation in which the buffer at the adder input will eventually overflow if the number of samples generated by the *data* star is long enough.

Pacers

The problem is that the *gauss* star does not generate enough samples at each visit. This problem can be corrected by pacing the *gauss* star. That is, we control how many samples it produces and when to produce them. The concept of pacing is illustrated in Fig. 1 (b). Note the buffer connecting the coder output to the gauss star. This is a pacer connection and Capsim draws the connection using a dashed line. When a pacer is connected to a source star, the star looks at the pace buffer and reads in all the samples on its input buffer. It ignores the value of the samples, but it counts their number. It then generates that many samples on its output buffer. Whence, when the coder produces 8*128 samples, the *gauss* star also produces 128*8 samples. The *add* star processes all 8*128 samples and sends them to the sink star. Thus the maximum number of samples on an output buffer is now bounded by 8*128, no matter how many samples the *data* star generates. The other pacer between the coder and the source may also be used to control buffer growth. The reason is that the coder produces 128 samples for every input sample. Thus, if we the let the *data* star produce only one sample per visit, then buffer growth can be controlled in this manner. This requires that the oversampling rate be specified to the *data* star. This is done through the pace rate parameter, which is the inverse of the oversampling ratio. However, pacing the *gauss* star solves the problem in this case.

As was noted, each source star with pacer capability has a number of parameters associated with the pacer. These include the pace rate and the number of samples to generate on the first visit. The pace rate controls the generation of samples. For example, if the pace rate is 0.01 then, for every 100 samples on the pace buffer, one sample is generated by the source. Each source star also has a parameter which specifies the total number of samples to generate. This can be set to -1 so that samples are generated indefinitely (this is used when the star is paced and another source determines the total number of samples to generate). Thus, in Fig.1, the *data* star may have the number of samples set to, say, 100000, while the gauss star is set to -1 since it is paced.

Monitoring Buffer Growth

Capsim provides the user with a file called *buffer.dat* which stores information on the buffer size for all connections in the universe and all galaxies. This file is created every time a simulation is run. The information in the file can be used by the user to determine problems and to identify the stars that need pacing.

Table 1. Buffer Monitoring, no pacers

gauss0:0 1	plot0:0 1	node0:0 28	node0:0 48
bdata0:0 1	node0:0 9	node0:0 29	node0:0 49
linecode0:0 1	node0:0 10	node0:0 30	node0:0 50
linecode0:1 1	node0:0 11	node0:0 31	node0:0 51
linecode0:0 2	node0:0 12	node0:0 32	node0:0 52
linecode0:0 3	node0:0 13	node0:0 33	node0:0 53
linecode0:0 4	node0:0 14	node0:0 34	node0:0 54
linecode0:0 5	node0:0 15	node0:0 35	node0:0 55
linecode0:0 6	node0:0 16	node0:0 36	node0:0 56
linecode0:0 7	node0:0 17	node0:0 37	
linecode0:0 8	node0:0 18	node0:0 38	
node0:0 1	node0:0 19	node0:0 39	
node0:0 2	node0:0 20	node0:0 40	
node0:0 3	node0:0 21	node0:0 41	
node0:0 4	node0:0 22	node0:0 42	
node0:0 5	node0:0 23	node0:0 43	
node0:0 6	node0:0 24	node0:0 44	
node0:0 7	node0:0 25	node0:0 45	
node0:0 8	node0:0 26	node0:0 46	
add0:0 1	node0:0 27	node0:0 47	

Table 1 shows the results of the topology in Fig. 1. (a) where buffer overflow occurs. The number by each connection is the number of segments (i.e. 128 cells) that have been allocated for that connection's buffer. Note that the buffer, *node0:0*, which connects to the adder input from coder, grows to 56 segments.

When pacers are used, as in Fig. 1 (b), buffer growth is bounded. The results are shown in Table 2. Note that the maximum number of segments is 2. That is 2x128 cells.

Table 2. Pacers

gauss0:0 1
linecode0:0 1
node0:0 1
node0:1 1
node0:2 1
gauss0:0 2
add0:0 1
plot0:0 1

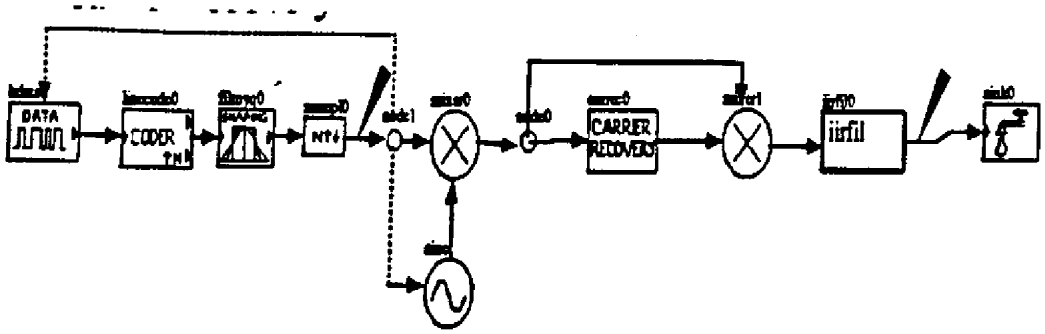

Figure 5. Carrier Recovery System Block Diagram

The carrier recovery block is a galaxy which is shown in Fig. 6. Based on the previous discussion, it is obvious that a serious buffer overflow problem exists at the mixer input. This is demonstrated in Table 4 . Note that Capsim also provides information on the buffers inside the galaxy. Now, by using pacers, shown by the dashed lines, we are able to control buffer growth, as demonstrated in Table 5 .

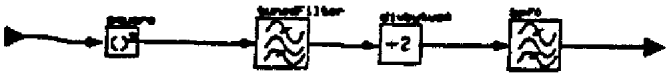

Figure 6. Carrier recovery galaxy

Table 4. No Pacers

bdata0:0 1	resmpl0:0 9	node0:1 1
linecode0:0 1	...	carrec0/square:0 1
linecode0:0 2	resmpl0:0 75	carrec0/tunedFilter:0 1
linecode0:0 3	plot0:0 1	carrec0/divbytwo0:0 1
linecode0:0 4	plot0:0 2	carrec0:0 1
linecode0:0 5	...	mixer1:0 1
linecode0:0 6	plot0:0 73	iirfil0:0 1
linecode0:0 7	plot0:0 74	spectrum0:0 1
linecode0:0 8	plot0:0 75	node1:0 76
filtnyq0:0 1	plot0:0 76	node1:0 77
filtnyq0:0 2	node1:0 1	node1:0 78
...	node1:0 2	node1:0 79
filtnyq0:0 8	...	node1:0 80
resmpl0:0 1	node1:0 75	...
resmpl0:0 2	sine1:0 1	node1:0 124
...	mixer0:0 1	
resmpl0:0 8	node0:0 1	

Comparing Table 4 and 5 we see that the buffer node1:0 connected to the mixer no longer grows beyond 75 segments. If more points were simulated with no pacers, this buffer will go unstable. Note that in this simulation, 100 bits were generated.

Table 5. With Pacers

bdata0:0 1	resmpl0:0 9	mixer0:0 1
linecode0:0 1	...	node0:0 1
linecode0:0 2	resmpl0:0 75	node0:1 1
linecode0:0 3	plot0:0 1	carrec0/square:0 1
linecode0:0 4	plot0:0 2	carrec0/tunedFilter:0 1
linecode0:0 5	...	carrec0/divbytwo0:0 1
linecode0:0 6	plot0:0 73	carrec0:0 1
linecode0:0 7	plot0:0 74	mixer1:0 1
linecode0:0 8	plot0:0 75	iirfil0:0 1
filtnyq0:0 1	plot0:0 76	spectrum0:0 1
filtnyq0:0 2	node1:0 1	
...	node1:0 2	
filtnyq0:0 8	...	
resmpl0:0 1	node1:0 75	
resmpl0:0 2	sine1:0 1	
...		
resmpl0:0 8		

Summary

This application note highlighted issues related to buffer growth and possible overflow when multi-rate sampling simulations are run in Capsim. Methods to control buffer growth were introduced which include pacing sources. Moreover, a feature in Capsim for monitoring the size of buffers in a simulation was introduced. To deal with all aspects of buffer growth requires a detailed explanation of the scheduling algorithm in Capsim, among other topics. We will leave this to a future application note.

Acknowledgement

The author would like to acknowledge Bill Hughes from GE Corporate Research for his report and insight into buffer control in Capsim.

Prepared by Saman Ardalan
Center for Communications and Signal Processing
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27695-7914