Accelerating Bit Error Rate Simulation in MATLAB with Graphics Processors

James Lebak, Brian Fanous, Nick Moore
MathWorks
{James.Lebak, Brian.Fanous, Nick.Moore}@mathworks.com

Introduction

Bit error rate simulations are used to estimate the error probability for a communications channel. Typically, many millions of trials must be run in order to have a reasonable estimate of the error probability. The Communications System Toolbox™ in MATLAB® contains tools that allow the user to construct these simulations, but executing the required trials can take a long time. In this paper, we examine techniques for accelerating bit error rate simulations using the Graphics Processing Unit (GPU) support provided by the combination of Parallel Computing Toolbox™ with Communications System Toolbox. We examine two different processing chains, based on subsets of processing chains for digital video broadcasting. We conclude that the best way to perform bit error rate simulation with the GPU is to move as much of the processing chain as possible to the GPU and to process multiple frames simultaneously.

Digital Video Broadcast Standard

The second generation Digital Video Broadcast standard (DVB-S.2) is a specification for digital signal transmission used by DIRECTV in the United States [1]. A Simulink® block diagram that models the transmission and reception of signals in this system is shown in Figure 1. The processing chain is dominated by the Low-Density Parity Check (LDPC) decoder [2, 3]. Our goal is to accelerate the MATLAB version of this processing chain using GPUs, so that a bit error rate simulation can be performed.

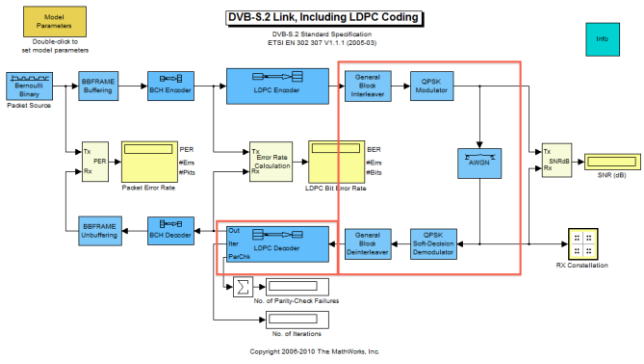

Figure 1. DVB-S.2 Block Diagram. The portion of the simulation enclosed in boxes is used as a benchmark.

The MATLAB version of the chain is implemented using System objects, which are self-contained MATLAB classes with separate setup and step ("execute") methods and code generation capabilities. The System objects in this chain, including the LDPC decoder, are implemented as compiled C code rather than as MATLAB code. MATLAB's Parallel Computing Toolbox enables us to write MATLAB code that executes on NVIDIA GPUs. We can also load and execute kernels that have been written using NVIDIA's C for CUDA language.

In the R2011a release of MATLAB, we implemented the LDPC Decoder using custom CUDA kernels with an internal MATLAB script to direct and coordinate the running of the kernels. For a single frame of data, this GPU-based System object executes roughly 20 times faster than the original System object in MATLAB, when startup overhead is excluded[1]. However, this translates only to a speedup of 5 times in the overall simulation performance. Our goal in this project was to explore techniques that could improve the simulation performance closer to that achieved by the GPU-based LDPC Decoder object.

DVB-S.2 Subset Benchmark

During the R2011b development cycle, we experimented with a processing chain based on a subset of the DVB-S.2 specification. This subset is marked in Figure 1, and includes the LDPC Decoder and five additional system objects: the Block Interleaver and De-interleaver, the Quadrature Phase Shift Key (QPSK) Modulator and Demodulator, and the Additive White Gaussian Noise (AWGN) Channel model. Our goal was to move the entire subset to the GPU. We implemented these five System objects on the GPU using MATLAB code. The results are surprising. When we execute the entire subset on the GPU, the subset runs 12 times faster than the original. If we use only the LDPC Decoder on the GPU, it runs 14 times faster. In this first attempt, moving more of the processing chain to the GPU caused us to slow down.

One cause of the slowdown is the overhead of transferring data between the CPU and the GPU. By default, System objects accept regular MATLAB arrays and output data in the same way. For this experiment, we enabled System objects to read and write GPUArrays, constructs in the Parallel Computing Toolbox that encapsulate data stored on the GPU. Using GPUArrays improves the execution time of the subset by allowing data to stay on the GPU. However, the execution is still no faster than just executing the LDPC Decoder on the GPU and all the other System objects on the CPU.

We gained further insight into the slowdown by comparing the performance of each of the new GPU System objects and their CPU-based counterparts at various input data sizes. We found that when the size is small, the GPU

[1] All performance measurements were obtained using a development version of MATLAB on a 2.5 GHz Intel Core 2 Quad running Windows 7 and using an NVIDIA Tesla C0160 GPU.

System object is slower than the CPU, but the execution time is not very significant. When the size is large, the GPU System object is faster than the CPU object. In MATLAB release R2011b the crossover point, at which the GPU version becomes faster than the CPU, occurs when around 10^5 data points are processed. Unfortunately, the frame size for our DVB-S.2 simulation is only 32400 elements, meaning we cannot expect to get much speedup. To achieve higher performance, we need to be able to increase the number of data points to utilize the GPU more effectively. We are not able to change the frame size. However, we have an essentially unlimited number of frames to process, and no latency requirement for any particular frame. Therefore, an easy way to increase the number of data points is to give each System object the capacity to process multiple frames at each step.

Figure 3 shows the results of multiframe processing on the GPU. When we process 8 frames per step, the all-GPU implementation of the subset is about 20% faster than the implementation that only had the LDPC Decoder on the GPU. At this rate, we obtain a speedup of about 17 times that of the original version of the subset. We conclude that at present, processing large amounts of data is crucial to making effective use of the GPU in MATLAB.

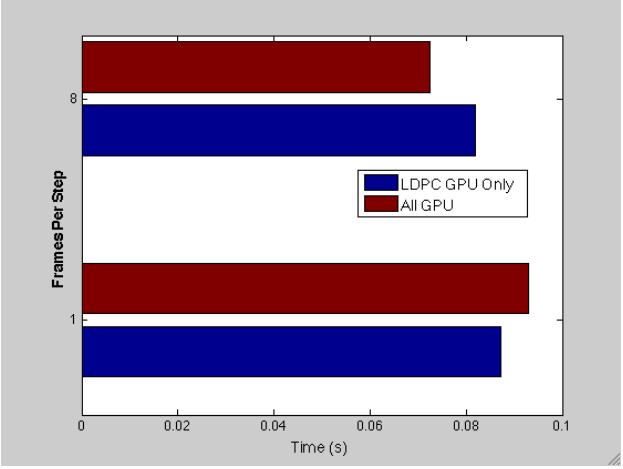

Figure 2. Effect of Multiframe Processing with the GPU.

DVB-S Subset Benchmark

The DVB-S.2 processing chain is dominated by the LDPC Decoder: profiling shows that it occupies 99% of the execution time of the selected subset. We wanted to see whether the GPU could still be an effective tool in a situation where the domination by a single kernel was less pronounced. We picked a subset of the Digital Video Broadcast Satellite standard (DVB-S, [4]) as a benchmark for this purpose. The chain includes the following operations. The percentage of time taken by each operation, as reported by the MATLAB profiler, is listed as well.

- Message generation (1%)
- Convolutional Encoder (1%)
- PSK Modulator (2%)
- AWGN Channel (7%)
- PSK Demodulator (5%)
- Viterbi Decoder (83%)
- Bit-error rate computation (1%)

In this case, we were able to prototype the entire processing chain on the GPU. The Viterbi Decoder was written using CUDA. All other operations were written in MATLAB.

Figure 3 shows the results. Each data frame consists of 2000 double-precision elements, so we expect that a single data frame is too small for us to see much speedup. And indeed this is the case: the overall processing speeds up when the Viterbi decoder is moved to the GPU, and slows down when all the other elements are moved to the GPU and single frames are processed. However, when we process 500 frames at a time on the GPU, we can get much better performance than the original CPU simulation. On our baseline machine the all-GPU multiframe simulation is about 15 times faster than the CPU single frame simulation. For reference, C code generated from this chain of System objects runs about 1.7 times faster than the MATLAB simulation.

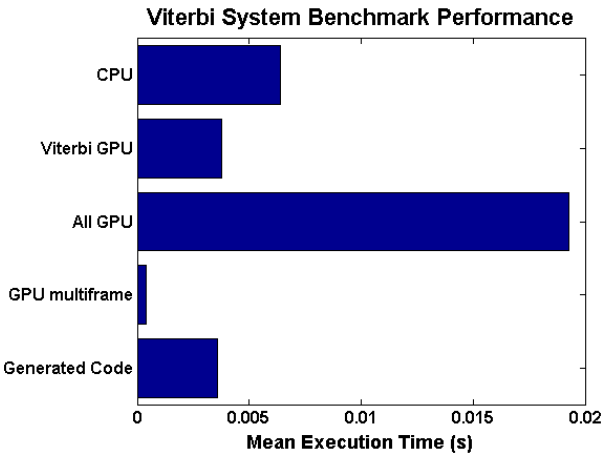

Figure 3. DVB-S Benchmark Results.

Observations and Conclusions

While the GPU can be a powerful tool for acceleration of bit error rate simulation in MATLAB, it is important to use it properly. Custom CUDA kernels are useful for accelerating simulations, but are expensive in terms of development effort. Kernels without a lot of computational density may be more cost-effective if written in MATLAB. A simulation can achieve high throughput, even if individual kernels have low computational density, by processing multiple frames of data simultaneously.

References

[1] DVB-S.2 Standard Specification, ETSI EN 302 307 V1.1.1 (2005-03).

[2] R. G. Gallager, Low-Density Parity-Check Codes, IRE Transactions on Information Theory, Vol. 8, No. 1, January 1962, pp. 21-28.

[3] W. E. Ryan, An introduction to LDPC codes, in Coding and Signal Processing for Magnetic Recoding Systems (Bane Vasic, ed.), CRC Press, 2004.

[4] ETSI Standard EN 300 421 V1.1.2: Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services. (DVB), European Telecommunications Standards Institute, Valbonne, France,1997-08.