# POLYPHASE CHANNELIZATION UTILIZING GENERAL-PURPOSE COMPUTING ON A GPU

Gregory Harrison, Ambrose Sloan, Wilbur Myrick, Joe Hecker, and David Eastin
SAIC, Innovative Technology Office, Chantilly, VA, USA
[harrisongm, sloana, myrickw, heckerjc, eastind]@saic.com

## ABSTRACT

Polyphase channelization is essential for a variety of applications involving bandwidth reduction and signal separation. Implementations of polyphase channelizers based on a general purpose processor (GPP) or field-programmable gate array (FPGA) platform have been investigated in the past. A novel approach to implementing a polyphase channelizer based on a graphics processing unit (GPU) is presented. Current GPUs have been shown to provide up to 500 GFlops for problems that do not have stringent size, weight and power (SWaP) requirements and are well suited for a parallel processing architecture. This paper compares the implementation of a polyphase channelizer based on a NVIDIA® 8800 GTX Graphics Card (NVIDIA Corporation) with one based on a central processing unit (CPU). The practical issues of implementation are presented and the performance measurements are discussed.

## 1. INTRODUCTION

Software defined radios (SDRs) have evolved over the years with more firmware radio-based approaches to accommodate stringent requirements of size, weight and power (SwaP). In recent years, research efforts have led to more software radio server-based architectures that leverage general purpose processors (GPPs) such as the currently deployed Anywave® Base Station product from Vanu Inc.. This paper extends the software radio-based architecture approach by exploring the use of graphics processing units (GPUs) with GPPs in a SDR framework. Rather than discussing general applications of the GPU to different elements of a software defined radio (SDR), we focus on a polyphase implementation of channelization. The motivation for this research is driven by the desire to leverage the highly parallel architecture of the GPU against the computationally intensive function of channelization. GPUs provide additional computational resources that, if implemented properly, could enable software-based radio systems to simultaneously process numerous bandwidth-intensive communication signals in real time, a requirement of cognitive radio networks.

## 2. POLYPHASE CHANNELIZATION

A common approach to performing channelization and resampling is by utilizing a polyphase filterbank implementation (see [2], [3]). For a filterbank of Q equally-spaced channels, each at a rate of P/Q times the input sample rate and using a finite impulse response (FIR) filter of length KQ, a polyphase implementation is more efficient by a factor of :

$$\frac{KPQ(\log(K)+\log(Q))}{K(\log(K)-\log(P))+P\log(Q)}$$

over a simple tune-upsample-filter-downsample approach. (for cases where Q > P, K > P, and where filtering is done using fast Fourier transform [FFT] processing).

The polyphase formulation of channelization also lends itself well to a parallel processing implementation as shown in Figure 1. The core processing stages of a polyphase filterbank can be preformed as large blocks of parallel FFTs, allowing the use of highly-optimized software libraries.

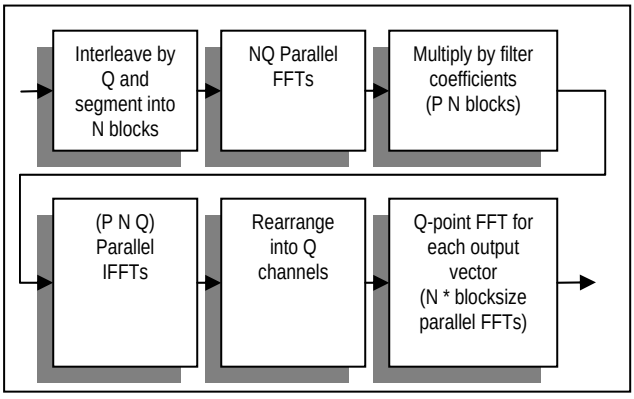


**Figure 1. Parallel Processing Chain for Q-channel Channelization at Oversampled Rate P**

### 2.1 GPU Processing

For all of our GPU development in this paper, we utilized a NVIDIA Corporation's NVIDIA® 8800 GTX (see [5] for specifications) running under 32-bit Red Hat, Inc.'s Red Hat® Enterprise Linux 4. We used the NVIDIA CUDA™ toolkit [6] and the CUFFT 1.1 library [7] to create our GPU channelizer implementation.

We wrote custom GPU kernel code using CUDA to perform our data conversions (integer/signed/endian) and to perform parallel data rearranging and multiplexing between the FFT stages. We also wrote a kernel for performing a tailored parallel binary reduction operation for the resampler application.

### 2.2 CPU Processing

For comparison, we created reference implementations of the channelizer using code targeted to an x86 CPU. We implemented these libraries in C/C++ using the Free Software Foundation's GNU® Compiler Collection (GCC) 3.4.5 under 32-bit Red Hat Enterprise Linux 4 (kernel 2.6.9-34) running on a Dell Precision® 690 workstation (Dell Computer Corporation) with two dual-core Intel Corporation's Intel Xeon® 2.33 GHz processors (5140 Woodcrest, 4MB L2 cache), and 2GB of system memory.

We used single-precision, SSE-enabled, multithreaded FFTW (version 3.1.2) libraries built from source to perform our CPU FFT computations. Whenever appropriate, we used parallel FFTW plans to maximize our CPU FFT performance. We configured FFTW to use four threads for generating its FFT plans in order to match the number of cores on our system.

## 3. TESTING GPU KERNAL

As our baseline test case, we used a 16-bit, signed complex data file stored on disk, containing $2^{28}$ samples (1GB file size). We ran a range of channelization tests, with the number of channels processed ranging from 2 to 1024 channels.

Because our processing is heavily input/output (I/O) bound by the file writing process, our timing results presented here represent no writing of the resulting data to disk, thereby highlighting the relative performance advantage of the GPU. These numbers do, however, include reading data from the file and transferring data to and from the GPU device.

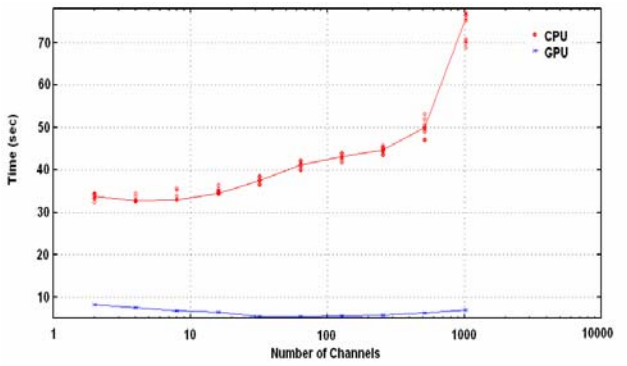


**Figure 2. Channelization Time for 1GB 16-bit Complex Signed Data File, Filter Length = (number channels * 63)**

From Figure 2, we can see that the GPU implementation was typically on the order of 4-10 times faster than the associated CPU polyphase implementation across a wide range of channelization levels.

Figure 3 shows timing results for a range of decimation rates for the polyphase resampler. In this example, the resample ratio was simple decimation by a factor of 1/N. These results show that the GPU processing advantage over the CPU grows as the decimation rate increases, with a 10x advantage for a decimation rate of 1024. Even for lower decimation rates, however, the GPU outperforms the CPU by at least a factor of two.

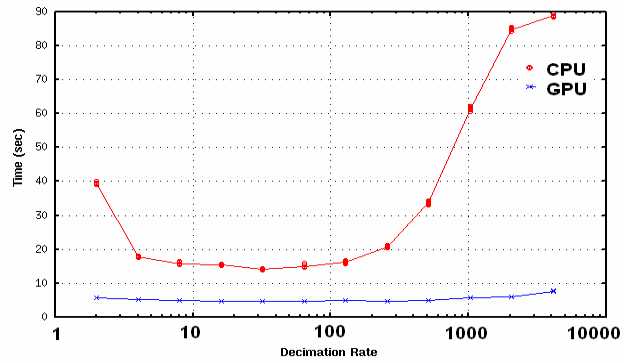


**Figure 3. Resample Time for 1GB 16-bit Complex, Signed Data File, Filter Length = (decimation rate * 63)**

## 4. DEVELOPING A GPU-BASED SDR

After successfully implementing and testing the GPU channelization kernel on a Dell Precision 690 workstation, we wanted to test the kernel code on a SDR platform. To satisfy this requirement, we developed a GPU-based SDR testing platform utilizing a microATX motherboard. This testing platform allows the greatest

flexibility in interfacing the GPU with a variety of peripherals while keeping the form factor as small as possible. The testing platform consists of the Universal Software Radio Peripheral (USRP) and GNU Radio software on a microATX motherboard interfaced to a NVIDIA 8800 GTX GPU. Although one could have selected a laptop with GPUs as the SDR testing platform, we wanted the ability to test different types of GPUs as well as maintain code compatibility with the baseline testing code. A block diagram of the microATX motherboard used for the GPU-based SDR testing platform is shown in Figure 4.

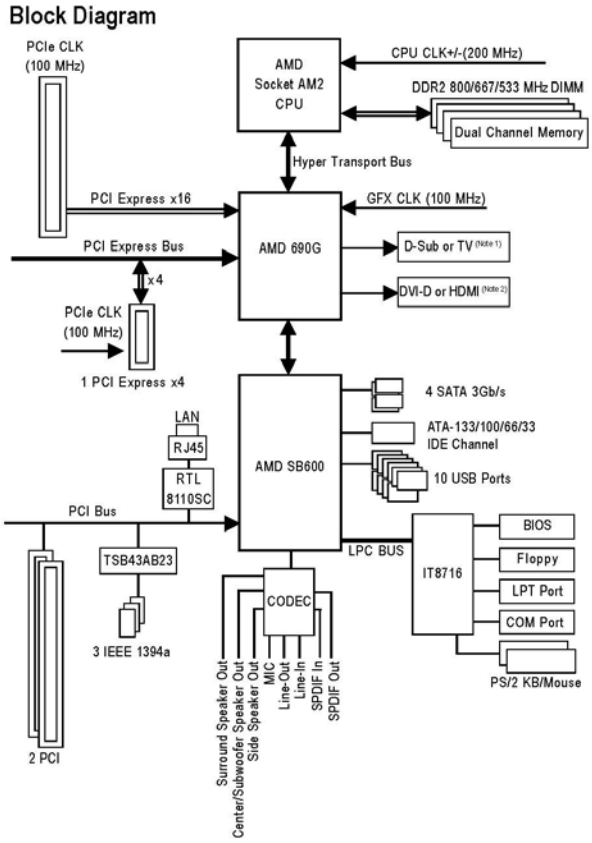


**Figure 4: Block diagram of motherboard used to create a GPU based SDR**

A simple experiment to test the GPU based SDR using the channelization kernel was to collect AM data using the USRP and GNU radio. Rather than developing GNU radio code to interface the USRP directly to the GPU, we decided to save the data to disk before processing it with the GPU. This scenario matches closely with the test that was previously done on the Dell Precision 690 workstation. The USRP was tuned to 1 MHz and decimated to 1 Msps so the entire AM band could be channelized with the GPU configured to generate 100 bands (each band has a 10 kHz bandwidth). Figure 5 illustrates an energy map of the bands generated from the GPU.

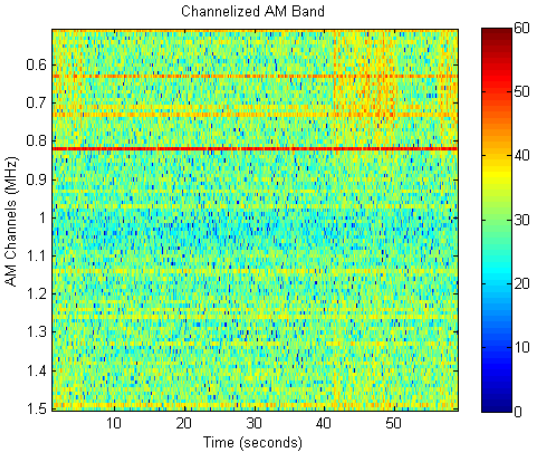


**Figure 5: Energy map of AM Band channelized using GPU-based channelization kernel**

The AM frequency band at 820 kHz (strongest band in energy map) was selected and demodulated after the channelization. The time to process this 1 minute data file through the GPU was approximately 4 seconds compared to approximately 9 seconds through the CPU. This indicates a 2.5x improvement, which is comparable to the baseline test case for 100 channels. This also indicates the potential to have real-time AM band processing if data is streamed from the USRP to the GPU directly.

## 5. SUMMARY AND CONCLUSIONS

Based on our results, using a GPU for polyphase channelization can provide a significant improvement in processing time. This advantage is especially pronounced for higher channelization and decimation rates. An AM channelization example was given, but this approach can easily be extended to a variety of signal processing routines involving multi-band partitioning as a preprocessing step. To further take advantage of the GPU's parallelism, we plan to extend the processing of the channelized data to include filtering, equalization, and demodulation while on the GPU. We have implemented GPU approaches with some of these algorithms already and need only apply them in parallel to the channelized streams.

## 6. REFERENCES

[1] *Use Cases for Cognitive Applications in Public Safety Communications Systems – Vol. 1 Review of the 7 July Bombing of the London Underground,*

http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-07-P-0019-V1_0_0.pdf

[2] *Business Model for Wireless PCS*, SDR Forum, 2003, http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-03-P-0001-V1_0_0_Business_Case.pdf

[3] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[4] F. J. Harris, C. Dick, and M. Rice, "Digital Receivers and Transmitters Using Polyphase Filter Banks for Wireless Communications," *IEEE Transactions on Microwave Theory and Techniques*, Vol. 51, No. 4, April 2003.

[5] http://www.nvidia.com/page/geforce_8800.html

[6] *CUDA Programming Guide, Version 1.1*, http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf

[7] *CUDA CUFFT Library User's Manual, Version 1.1*, http://developer.download.nvidia.com/compute/cuda/1_1/CUFFT_Library_1.1.pdf

[8] *FFTW3 User's Guide*, http://www.fftw.org/fftw3.pdf