Radar Pulse Compression Using the NVidia CUDA Framework

Stephen Bash, David Carpman, David Holl
{bash,dcarpman,dholl}@ll.mit.edu
MIT Lincoln Laboratory, Lexington, MA 02420

Introduction

Over the past several years, graphics processing units (GPUs) have gained interest as general purpose highly parallel coprocessors. Early adopters were forced to use traditional 3D graphics application programming interfaces (APIs) in order to access the computational power of the GPU. This process of recasting general purpose problems into graphical terms can be time consuming and create obscure code. The introduction of NVidia's Compute Unified Device Architecture (CUDA) Framework, a C-language development environment for NVidia GPUs, is designed to ease the burden placed on the general purpose GPU programmer. In parallel with the CUDA release, NVidia also released implementations of the BLAS and FFT libraries for the GPU under the names CUBLAS and CUFFT, respectively.

Previous research [1,2] has shown the vast computational power of GPUs for signal processing. Modern radar signal processing is a data parallel operation that benefits from parallel processing architectures. This investigation will focus on the real-world benefit of GPUs for radar pulse compression. First, the performance of 1D and 2D FFTs on a GPU via CUFFT will be compared to a modern day multi-core CPU implementation using FFTW [3]. Subsequently, these performance results will inform the implementation of two surrogate radar pulse compression chains, having differing processing complexity, which will also in turn be benchmarked similar to the FFTs.

FFT Benchmarking Methodology

The benchmarks to be presented are based on the fft_bench code found on the NVidia CUDA Developer forums [4]. For benchmarking purposes fft_bench runs each test point 32 times and chooses the trial with the fastest time to compute the number of FFTs possible per second.

A modified version of fft_bench used during this study runs complex 1D and 2D transforms on the GPU using CUFFT, and 1D and 2D transforms on the CPU using FFTW. The code tests multiple batch sizes (number of same size FFTs to be executed), and both in-place and out-of-place transforms are executed. The FFTW planner was instructed to create multi-threaded plans in patient mode. The timing results do not include planning time for either the GPU or the CPU. The timing results do include the transfer time required to copy data to and from the GPU. GPU results are measured using page-locked host memory to reduce transfer time.

This work is sponsored by the Air Force Research Laboratory under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and not necessarily endorse by the United States Government.

Benchmarking was conducted on two dual-core AMD Opterons at 2.6 GHz with 16 GB of DDR2 RAM. In the same machine, an EVGA GeForce 8800 Ultra Superclocked™ video card at 1.6 GHz with 768 MB of DDR3 RAM was used for GPU benchmarking.

Impact of Transfer Time

As mentioned previously, the benchmarking code includes the time required to copy data from main system memory to the GPU for processing. For numerically lightweight operations such as a 1D FFT, the time required for the transfer dwarfs the FFT computation time. Figure 1 shows the time required to compute complex 1D fixed length FFTs with varying batch sizes and the time required to transfer the data to the GPU.

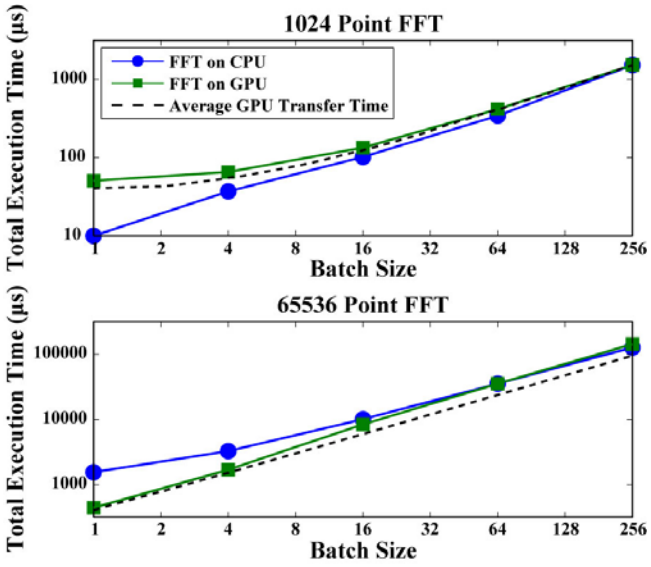

Figure 1: FFT Execution Time including GPU Transfer Time

Figure 1 shows that for short FFTs, the CPU can compute batch sizes up to 256 in the time required to copy the data to the GPU. For larger FFTs there is margin between the time required for transfer and the CPU computation time. When such a margin exists, the GPU can sometimes transfer the data and compute the FFTs faster than the CPU.

The results in Figure 1 show the timing for doing a single FFT on the GPU before returning the data to host memory, and suggest that data transfer time is significant. For data parallel operations the GPU will in general perform better as the number of operations increases for a fixed transfer size. Also, precomputing data that must then be transferred may not be as beneficial on the GPU as on other computing platforms. These are important implementation details to be considered.

FFT Benchmark Results

At each test point (FFT and batch size) the maximum performance is chosen for each computational engine.[1] Portrayed in Figure 2 and elsewhere, the GPU speedup is defined as the ratio of GPU performance to the CPU performance. Values greater than one indicate the GPU is faster, values less than one indicate the CPU is faster.

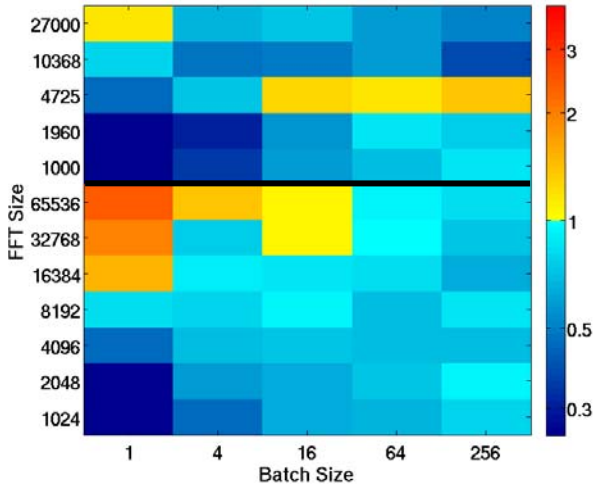

Figure 2: 1D FFT GPU Speedup vs. FFT and Batch Size

Figure 2: 1D FFT GPU Speedup vs. FFT and Batch Size expands on the results in Figure 1. For 1D FFTs, the GPU shows an advantage on large power-of-two sized with the peak speedup around 2.5.

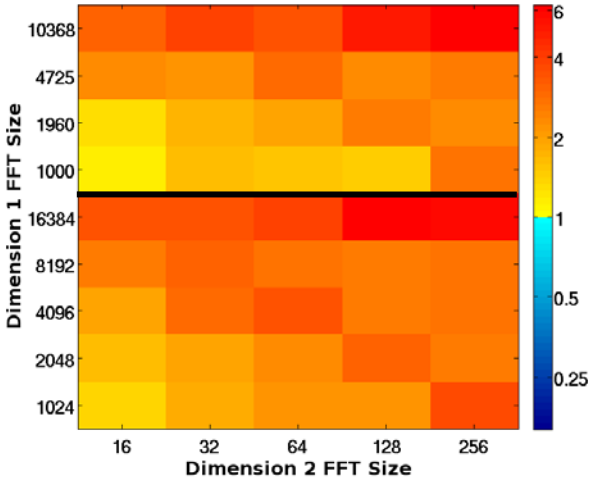

Figure 3: 2D FFT GPU Speedup vs. FFT Size

In Figure 3, 2D FFTs show a consistent advantage for the GPU, with the maximum speedup over 6. The 2D FFT requires more operations than its 1D counterpart, and as such, the cost of data transfer is less of a performance penalty.

Pulse Compression

Pulse compression in radar is a combination of waveforms and processing that allows a long pulse to be transmitted while retaining the range resolution of a short pulse [5]. In what follows fast time refers to intra-pulse timing (sample to sample), while slow time refers to inter-pulse timing (the time between the same sample in two consecutive pulses).

The first pulse compression processor (Figure 4) is a traditional convolution theorem implementation. The total operation count of this processing chain is similar to the operation count for a large 2D FFT, so GPU speedup similar to the 2D FFT is expected.

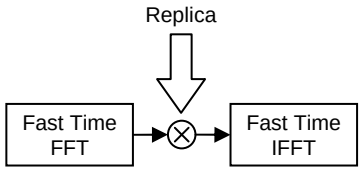

Figure 4: Traditional Pulse Compression

The second pulse compression implementation is shown in Figure 5. It adds complexity to enable spatially diverse processing of Doppler intolerant orthogonal waveforms: a slow-time FFT and Doppler correction prior to the fast-time FFT, and the convolution with multiple orthogonal replicas. The additional steps incur a large number of GPU-efficient operations, so it is expected that this processing chain should see significant speedup versus the CPU implementation.

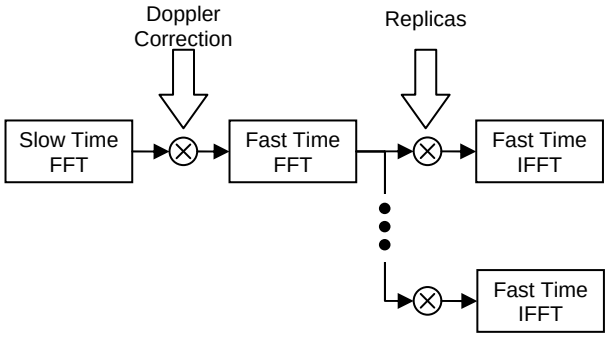

Figure 5: Doppler Intolerant Pulse Compression

References

[1] I. Buck, "GeForce 8800 & NVIDIA CUDA: A New Architecture for Computing on the GPU," *Supercomputing 2006 Workshop*, Tampa, FL, November 13, 2006.

[2] M. McGraw-Herdeg, D. Enright, and B. Michel, "Benchmarking the NVIDIA 8800GTX with CUDA Development Platform," *HPEC 2007 Proceedings*, Lexington, MA, September 19, 2007.

[3] M. Frigo and S. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE* 93 (2), 216–231 (2005).

[4] fft_bench code available at: http://forums.nvidia.com/index.php?showtopic=42482

[5] M. Skolnik, Radar Handbook, Second Edition, McGraw Hill Publishing, Boston, MA, 1990.

[1] This may result in comparing an in-place GPU FFT vs. an out-of-place CPU FFT.