Compute Unified Device Architecture (CUDA) Fast Fourier Transform (FFT) benchmarks on the NVIDIA® Tesla™ c2070 graphics processing unit (GPU)

Mathew R.W. Yamasaki, Chris Fallen, Greg Newby
University of Alaska Fairbanks
Arctic Region Supercomputing Center

I. Introduction

In 2006, graphics-processing unit (GPU) creator, developer and manufacturer, NVIDIA Corporation, released the first commodity GPU, the GeForce™ 8800, built upon the Compute Unified Device Architecture (CUDA). This new parallel architecture allowed programmers to pass computationally intensive portions of their applications to the GPU using the *CUDA C* programming language. This new GPU programming language consists of extensions to the C programming language, which simplifies learning and decreases development time.

The discrete Fourier transform (DFT), and its faster implementation, the Fast Fourier Transform (FFT) is used extensively in the field of digital signal processing (DSP). The DFT converts a given signal from the time-frequency domain to the frequency-amplitude domain. This decomposes the input signal into its constituent signals which can then be analyzed. This is similar to the way the human ear distinguishes individual musical notes that make up chords. The DFT is used for everything from magnetic resonance image (MRI) reconstruction for detecting cancers to increasing the resolutions of photographs and videos.

II. Problem

The computational throughput of the GPU is indisputable. However, GPUs may not execute an FFT any faster than an identical one executed on a central processing unit (CPU). Factors such as problem size and data transfer times between the CPU and GPU must be considered when determining if a GPU-accelerated FFT is appropriate.

The increased application of the DFT/FFT to solve increasingly complex problems in physics, engineering, and signal processing is dependent on processing speed, and GPU-acceleration can be a promising and economical technique for improving FFT performance.

Figure 1: The NVIDIA Tesla c2070 GPU (images not to scale).

III. Solution

We used the NVIDIA Tesla c2070 GPU (Figure 1) to benchmark CUDA FFTs for a range of problem sizes in one and two dimensions. Code was written using the CUDA Fast Fourier Transform (CUFFT) library. The basic CUFFT program structure used for benchmarking is shown in Figure 2.

To allow researchers to determine whether a GPU-accelerated FFT is appropriate for their application, benchmarks were also conducted using the Fastest Fourier Transform in the West (FFTW) library on a single Intel® Core™ i7-2600 central processing unit (CPU) based processor. CUFFT is modeled after the FFTW library.

For each FFT size tested in one and two dimensions, we used the average of 20 program executions to make timing comparisons. High-resolution timers with accuracies of ±0.000001 milliseconds (ms) were used.

For testing uniformity, 1D and 2D arrays were populated with complex numbers of the form *a* + *bi* where *a* is represented by 1 and *b* is represented by 0 (1 + 0*i*). Using matrices of 1's for input data allowed us to quickly verify the accuracy of the program without adversely affecting the performance benchmark.

Figure 2: Basic program structure for the 1D CUFFT program.

```
cufftResult flag;

unsigned int signalSize = NX * BATCH;
unsigned int memSize = signalSize * sizeof(cufftComplex);

// Allocate host (CPU) memory for the input signal
Complex* h_signal = (Complex*)malloc(memSize);

// Initialize the input signal
for (unsigned int i = 0; i < NX; i++) {
     h_signal[i].x = 1; //real
     h_signal[i].y = 0; //imag
}

// Allocate device memory for the signal
cufftComplex *d_inSignal, *d_outSignal;
cudaMalloc((void**)&d_inSignal, memSize));
cudaMalloc((void**)&d_outSignal, memSize));

// Transfer the signal from host memory to device memory
CUDA_SAFE_CALL(cudaMemcpy(d_inSignal, h_signal, memSize,
cudaMemcpyHostToDevice));

// Create a 1D FFT plan
cufftHandle plan;
flag = cufftPlan1d(&plan, NX, CUFFT_C2C, BATCH);
if (CUFFT_SUCCESS != flag) {
     printf("Error: cufftPlan1d(CUFFT_C2C) failed\n");
}

// Use the CUFFT plan to transform the signal out of place
for (int i = 0; i < BENCH; i ++) {
     flag = cufftExecC2C(plan, d_inSignal, d_outSignal, CUFFT_FORWARD);
     if (CUFFT_SUCCESS != flag) {
          printf("Error (cufftExecC2C): %s \n",
               cudaGetErrorString(cudaGetLastError()));
          printf("Error code (cufft) = %d\n", flag);
     }
}

// Copy result from device to host
cufftComplex* h_transformed_signal = h_signal;
CUDA_SAFE_CALL(cudaMemcpy(h_transformed_signal, d_outSignal,
     memSize, cudaMemcpyDeviceToHost));
// Destroy the CUFFT plan
cufftDestroy(plan)

// Free host and device memories
free(h_signal);
cudaFree(d_inSignal);
cudaFree(d_outSignal);

cudaThreadExit();
```

Initialize the GPU

Transfer input data to the GPU

Create and execute the FFT plan on the GPU

Transfer results from the GPU to CPU

IV. Results

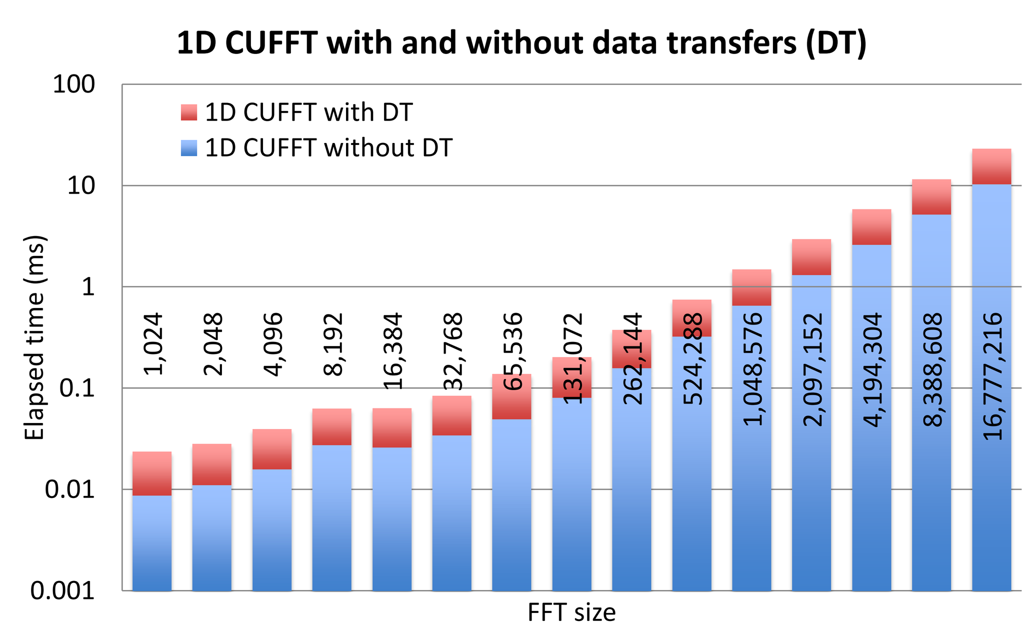

Figure 3: Benchmark results show minimal but consistent overhead for all sizes of 1D FFTs.

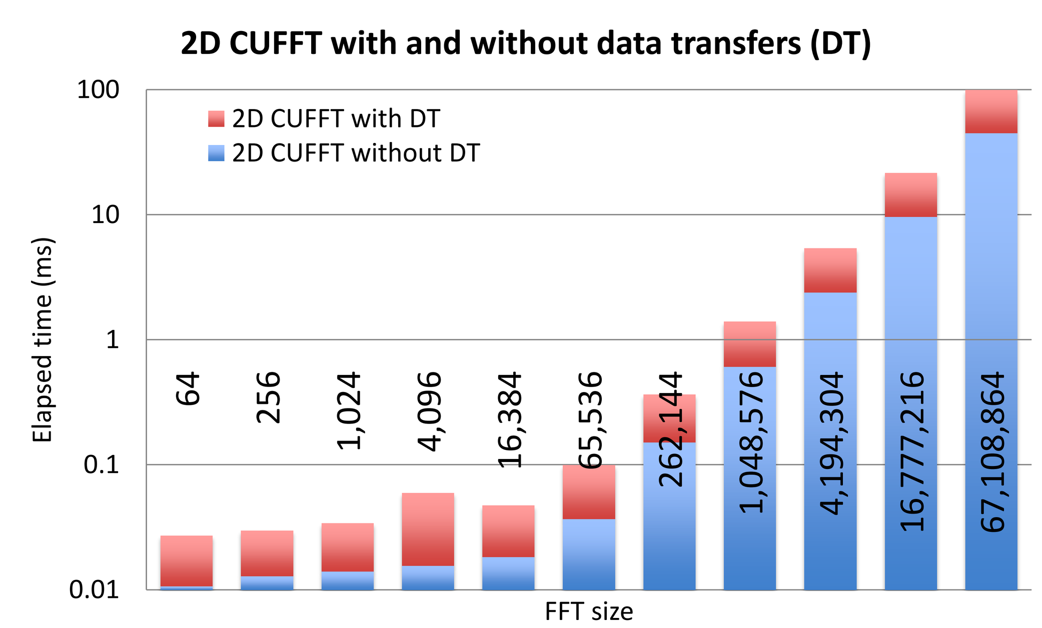

Figure 4: Benchmark results show significant data transfer overhead for smaller (65,536 and below) 2D FFT sizes.

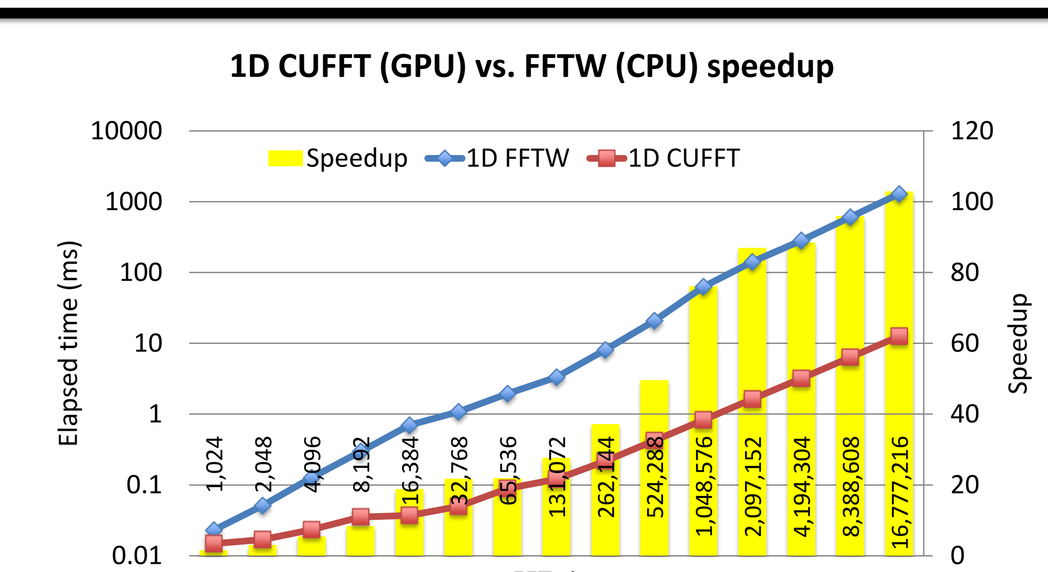

Figure 5: Results show a large speedup increase after 524,288 elements.

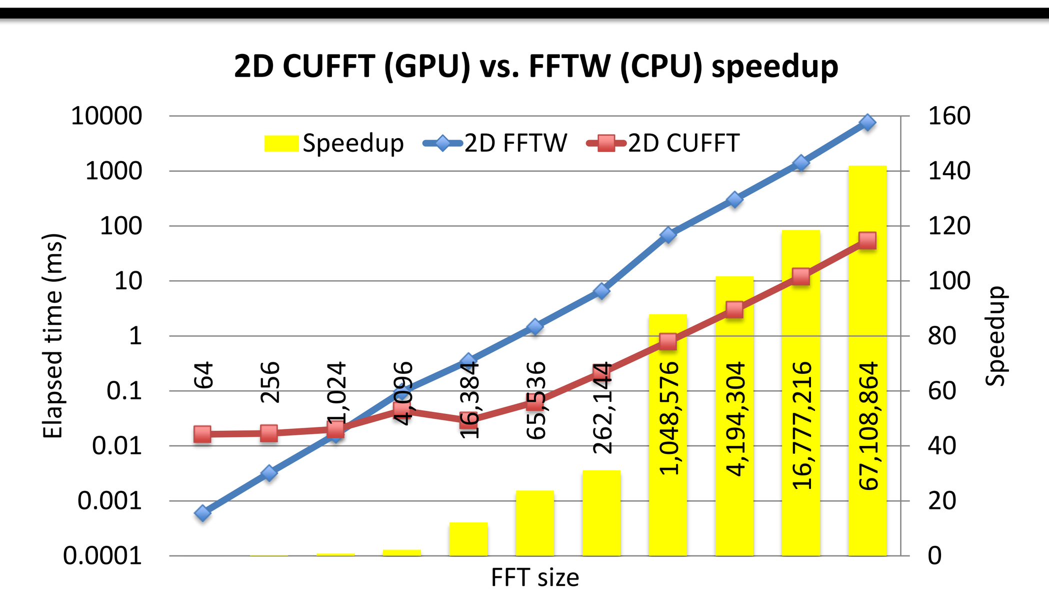

Figure 6: CUFFT is over 140 times faster than FFTW a 2D FFT of 67,108,864 elements.

V. Conclusion

The data transfer times for 1D FFTs (Figure 3) decrease as the size of the transform increases. This inverse relationship, which also occurs with 2D FFTs (Figure 4), is the result of CUDA's massively parallel architecture, and resulting latency produced by data transfers.

Based on the 1D speedup results (Figure 5), there is minimal advantage of CUFFT over the CPU FFT for transform sizes below 8,192.

For 2D transform sizes under 65,536 elements, there is little speedup of CUFFT. There is a very large speedup increase for benchmark sizes greater than 262,144 (Figure 6).

The overall results indicate that GPU-accelerated FFTs are better suited for 1D and 2D transform sizes greater than 1,000,000 elements.

The Tesla c2070's performance increases with progressively larger FFTs, indicating that massive and efficient parallelization of CUDA dramatically increases throughput for large problem sizes. This gives researchers and scientists portable supercomputing capabilities at a fraction of the cost and power consumption.

Acknowledgements

Sincere thanks to the JEOM program for making this internship possible, especially Pam Gilliam for her professionalism and patience. Mathew Yamasaki is also sincerely grateful to Dr. Newby and Dr. Fallen at UAF ARSC for their patience, guidance, and encouragement, without which this experience would never have been possible. Many thanks also to all the supporting staff at ARSC.