SOFT354 - Parallel Computing and Distributed Systems

A comparison of the Discrete Fourier Transform algorithm implemented in CUDA and MPI.

Ben Lancaster
January 13, 2018

Abstract

My placement as a Firmware Engineer at Spirent, a world leader in GNSS simulators. Coming from a Computer Science I was up to speed on the programming aspect of firmware, however was lacking in experience of electronic lab equipment. Constant exposure to this new area of technology and equipment has greatly improved my knowledge in GNSS and embedded programming and has influenced my future career choices. I was chiefly responsible for writing a new programmable timer, implementing fan-control strategies and power calibration schemes, writing a driver to read in peripheral devices, provisioning a Linux hypervisor, and designing and implement a Linux USB driver.

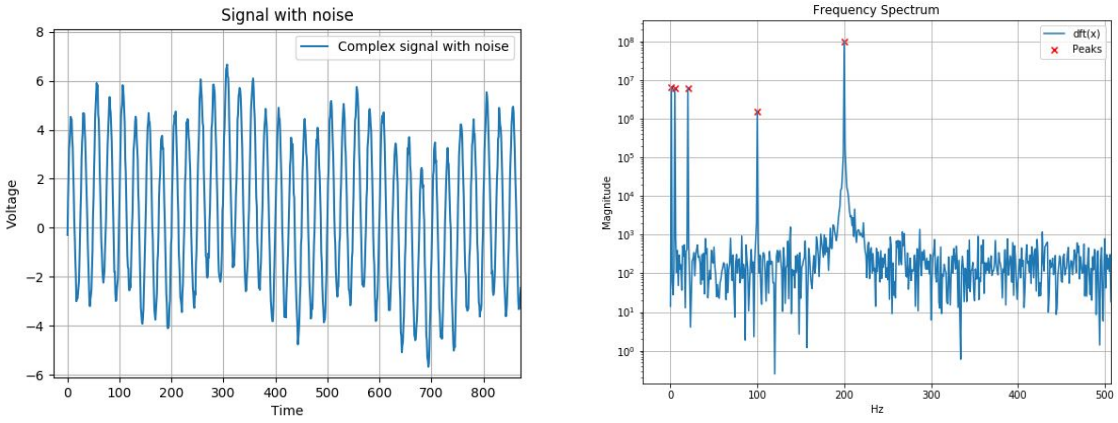
Table of Contents

1 Introduction

This report discusses the implementation and performance of the Discrete Fourier Transform (DFT) algorithm in CUDA and MPI.

DFT is largely used in digital signal and image processing applications.

In this report, I will be implementing the DFT to convert signals in the time domain to the frequency domain.



(a) Time domain **(b)** Frequency domain

Figure 1: Left: A compound signal made up of 1, 5, 20, 100, and 200 Hz sine waves in the time domain. **Right:** Frequency domain representation showing high amplitude peaks for the 1, 5, 20, 100, and 200 Hz waveforms.

2 Implementation

2.1 CUDA

2.1.1 Dynamic Shared Memory

As each operation on each sample requires values from all other samples, global memory accesses will be largest performance bottleneck. To reduce the amount of global memory accesses, I had the first thread of each block copy the complete global memory copy of the samples to a local shared memory buffer, which the DFT kernel would access instead of the global memory.

On Compute Capability 2.0 devices, the maximum shared memory size per block is 48KB, meaning that if the kernel was launched with 1 block, it could access all 48KB of memory. If two blocks were used, each block would have 24KB of memory.

To be able to fit the entire sample set in shared memory, I had to keep the maximum shared memory per block at it's highest by using as few blocks as possible. I used the maximum of 1024 threads per block to reduce the number of required blocks.

Using any number of 8-byte samples below 2048, which would use up to 2 blocks of 1024 threads and 24KB of shared memory, allows all samples for each block to fit into their shared memory.

If using more than 2048 8-byte samples, resulting in using 3 or more blocks, results in the shared memory per block overflowing. The kernel handles this by pausing blocks that do not have enough shared memory ready, and running blocks that do have all their shared memory ready.

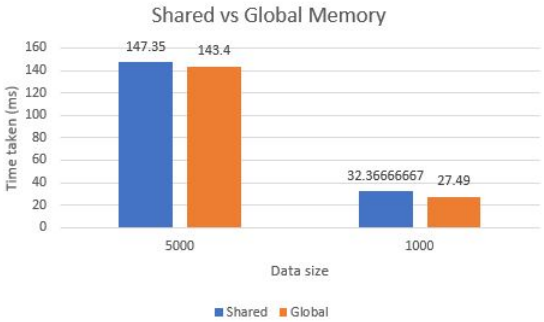


Figure 2: Average kernel execution time using shared vs. global memory.

Even though we are accessing faster memory, it's size limitations causes resource availability delays when using larger data sets, which results in higher latency and lower performance.

2.2 MPI

With DFT, each sample must have operate on all other samples. In my implementation, I must first copy the complete sample array to each node.

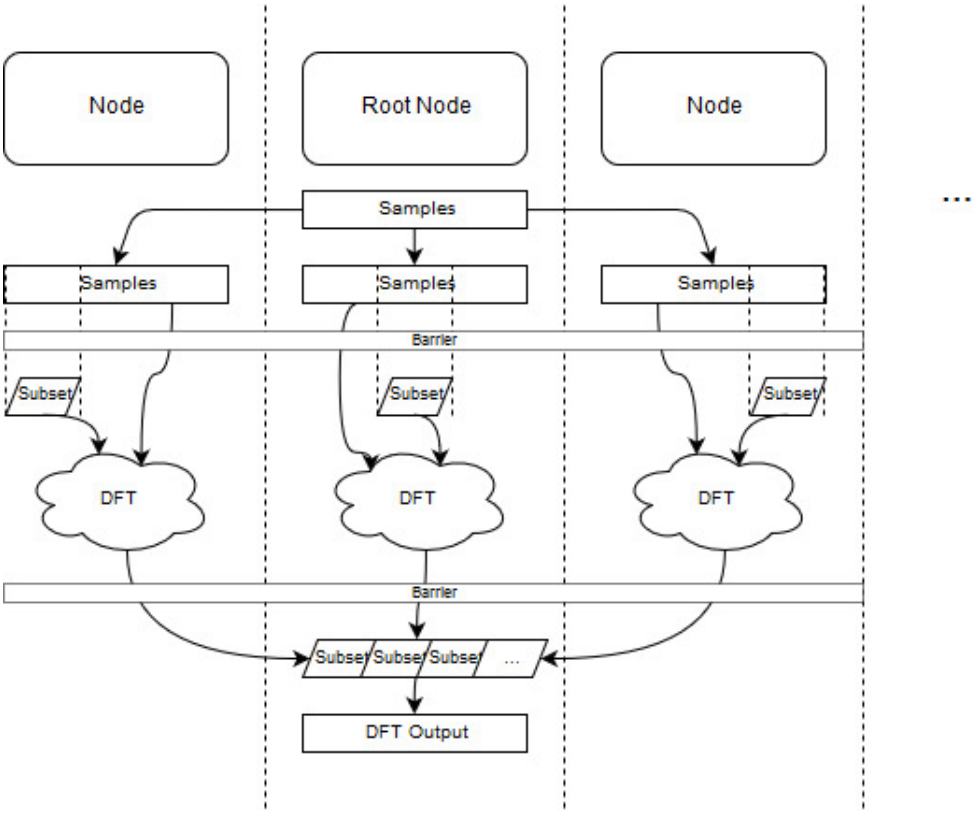


Figure 3: Control flow diagram for the MPI DFT algorithm.

3 Evaluation

3.1 Measuring Performance