Phase Vocoder

Davis Polito

Because I know you guys love formulas

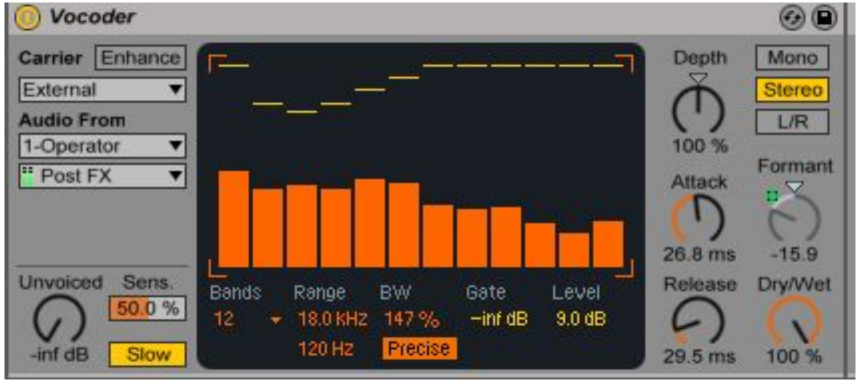
$$PV(x, \alpha, \beta) = \int_{-\infty}^{\infty} |X(\frac{\omega}{\beta}, \alpha t)| \cdot e^{i\phi_{pv}(\omega,t)} d\omega = y(t)$$

$$\phi_{pv}(\omega, t) = \angle X(\frac{\omega}{\beta}, 0) + \int_0^t \frac{d}{dt}[\phi(\omega, t)]\,dt = \angle X(\frac{\omega}{\beta}, 0) + \phi(\omega, t)$$

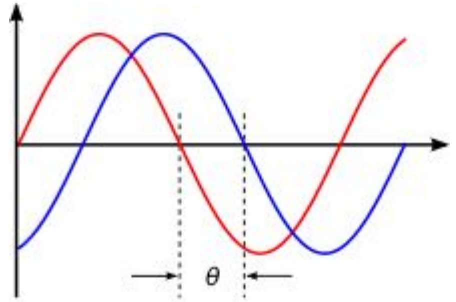
Actually though, what is it?

Vocoder - Modulate Frequency of a voice based on frequency of input

Phase Vocoder - Modulate Frequency and Time information of Input Signal



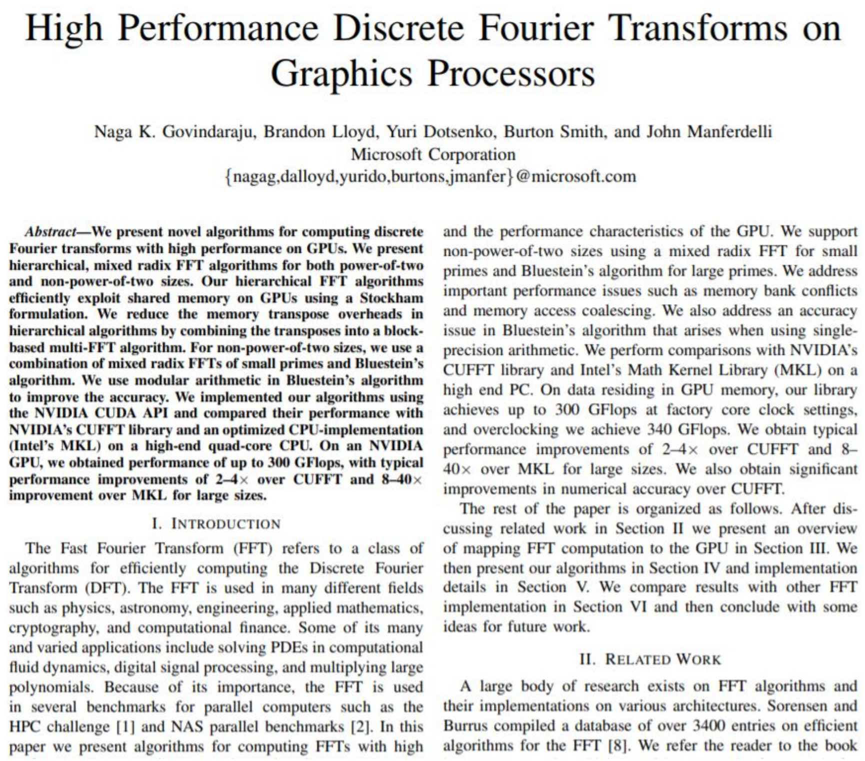
Carrier: External
Audio From: 1-Operator
Post FX
Unvoiced: -inf dB
Sens.: 50.0 %
Slow
Bands: 12
Range: 18.0 kHz, 120 Hz
BW: 147 %
Precise
Gate: −inf dB
Level: 9.0 dB
Depth: 100 %
Attack: 26.8 ms
Release: 29.5 ms
Stereo
Formant: -15.9
Dry/Wet: 100 %



But isn’t this already being done?

Yes! but…

What have I done?



High Performance Discrete Fourier Transforms on Graphics Processors

Naga K. Govindaraju, Brandon Lloyd, Yuri Dotsenko, Burton Smith, and John Manferdelli
Microsoft Corporation
{nagag,dalloyd,yurido,burtons,jmanfer}@microsoft.com

***Abstract*—We present novel algorithms for computing discrete Fourier transforms with high performance on GPUs. We present hierarchical, mixed radix FFT algorithms for both power-of-two and non-power-of-two sizes. Our hierarchical FFT algorithms efficiently exploit shared memory on GPUs using a Stockham formulation. We reduce the memory transpose overheads in hierarchical algorithms by combining the transposes into a block-based multi-FFT algorithm. For non-power-of-two sizes, we use a combination of mixed radix FFTs of small primes and Bluestein's algorithm. We use modular arithmetic in Bluestein's algorithm to improve the accuracy. We implemented our algorithms using the NVIDIA CUDA API and compared their performance with NVIDIA's CUFFT library and an optimized CPU-implementation (Intel's MKL) on a high-end quad-core CPU. On an NVIDIA GPU, we obtained performance of up to 300 GFlops, with typical performance improvements of 2–4× over CUFFT and 8–40× improvement over MKL for large sizes.**

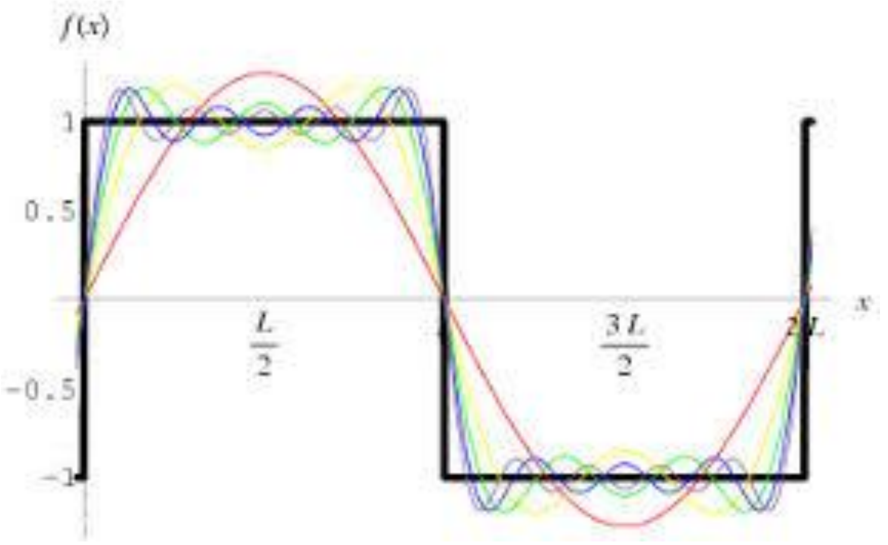
I. Introduction

The Fast Fourier Transform (FFT) refers to a class of algorithms for efficiently computing the Discrete Fourier Transform (DFT). The FFT is used in many different fields such as physics, astronomy, engineering, applied mathematics, cryptography, and computational finance. Some of its many and varied applications include solving PDEs in computational fluid dynamics, digital signal processing, and multiplying large polynomials. Because of its importance, the FFT is used in several benchmarks for parallel computers such as the HPC challenge [1] and NAS parallel benchmarks [2]. In this paper we present algorithms for computing FFTs with high and the performance characteristics of the GPU. We support non-power-of-two sizes using a mixed radix FFT for small primes and Bluestein's algorithm for large primes. We address important performance issues such as memory bank conflicts and memory access coalescing. We also address an accuracy issue in Bluestein's algorithm that arises when using single-precision arithmetic. We perform comparisons with NVIDIA's CUFFT library and Intel's Math Kernel Library (MKL) on a high end PC. On data residing in GPU memory, our library achieves up to 300 GFlops at factory core clock settings, and overclocking we achieve 340 GFlops. We obtain typical performance improvements of 2–4× over CUFFT and 8–40× over MKL for large sizes. We also obtain significant improvements in numerical accuracy over CUFFT.

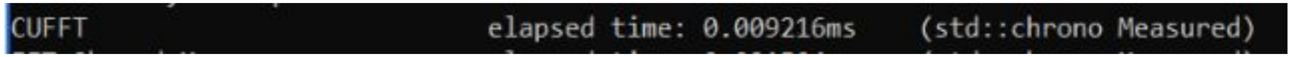
The rest of the paper is organized as follows. After discussing related work in Section II we present an overview of mapping FFT computation to the GPU in Section III. We then present our algorithms in Section IV and implementation details in Section V. We compare results with other FFT implementation in Section VI and then conclude with some ideas for future work.

II. Related Work

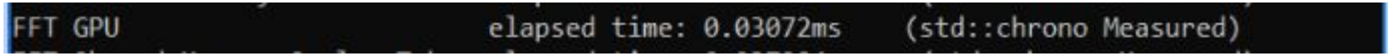
A large body of research exists on FFT algorithms and their implementations on various architectures. Sorensen and Burrus compiled a database of over 3400 entries on efficient algorithms for the FFT [8]. We refer the reader to the book



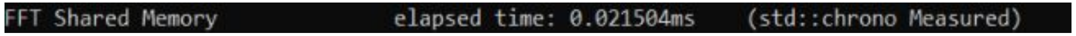
Here’s Some Data



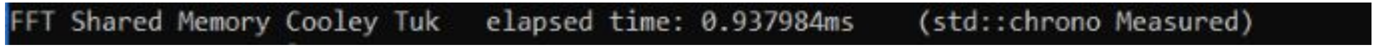
```
CUFFT                           elapsed time: 0.009216ms    (std::chrono Measured)
```



```
FFT GPU                         elapsed time: 0.03072ms    (std::chrono Measured)
```

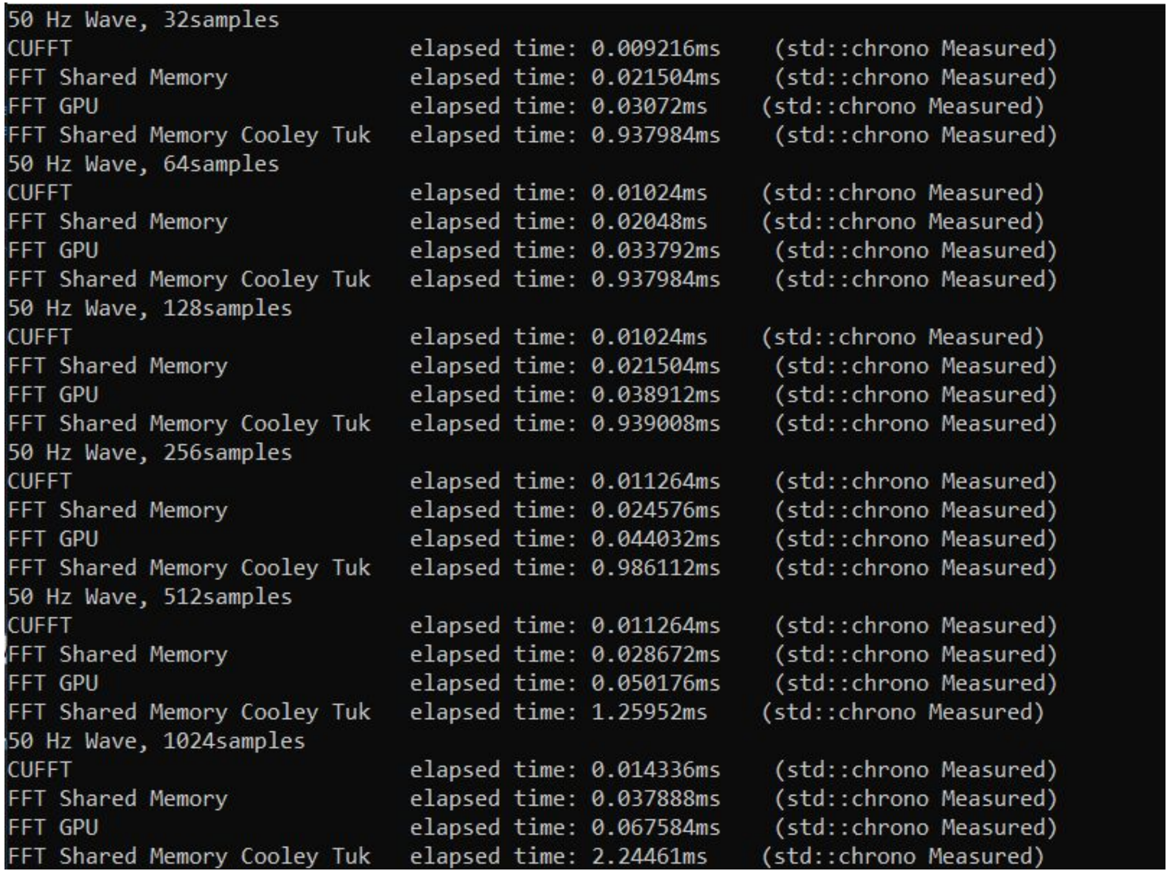


```
FFT Shared Memory               elapsed time: 0.021504ms    (std::chrono Measured)
```



```
FFT Shared Memory Cooley Tuk    elapsed time: 0.937984ms    (std::chrono Measured)
```

All The Data



```
50 Hz Wave, 32samples
CUFFT                         elapsed time: 0.009216ms    (std::chrono Measured)
FFT Shared Memory             elapsed time: 0.021504ms    (std::chrono Measured)
FFT GPU                       elapsed time: 0.03072ms    (std::chrono Measured)
FFT Shared Memory Cooley Tuk  elapsed time: 0.937984ms    (std::chrono Measured)
50 Hz Wave, 64samples
CUFFT                         elapsed time: 0.01024ms    (std::chrono Measured)
FFT Shared Memory             elapsed time: 0.02048ms    (std::chrono Measured)
FFT GPU                       elapsed time: 0.033792ms    (std::chrono Measured)
FFT Shared Memory Cooley Tuk  elapsed time: 0.937984ms    (std::chrono Measured)
50 Hz Wave, 128samples
CUFFT                         elapsed time: 0.01024ms    (std::chrono Measured)
FFT Shared Memory             elapsed time: 0.021504ms    (std::chrono Measured)
FFT GPU                       elapsed time: 0.038912ms    (std::chrono Measured)
FFT Shared Memory Cooley Tuk  elapsed time: 0.939008ms    (std::chrono Measured)
50 Hz Wave, 256samples
CUFFT                         elapsed time: 0.011264ms    (std::chrono Measured)
FFT Shared Memory             elapsed time: 0.024576ms    (std::chrono Measured)
FFT GPU                       elapsed time: 0.044032ms    (std::chrono Measured)
FFT Shared Memory Cooley Tuk  elapsed time: 0.986112ms    (std::chrono Measured)
50 Hz Wave, 512samples
CUFFT                         elapsed time: 0.011264ms    (std::chrono Measured)
FFT Shared Memory             elapsed time: 0.028672ms    (std::chrono Measured)
FFT GPU                       elapsed time: 0.050176ms    (std::chrono Measured)
FFT Shared Memory Cooley Tuk  elapsed time: 1.25952ms    (std::chrono Measured)
50 Hz Wave, 1024samples
CUFFT                         elapsed time: 0.014336ms    (std::chrono Measured)
FFT Shared Memory             elapsed time: 0.037888ms    (std::chrono Measured)
FFT GPU                       elapsed time: 0.067584ms    (std::chrono Measured)
FFT Shared Memory Cooley Tuk  elapsed time: 2.24461ms    (std::chrono Measured)
```

Timeline

Milestone 2:

Basic Phase Vocoder Algorithm running on Jetson Nano

Milestone 3:

Real Time interaction as well as basic pitch movement (trying out different Phase Vocoding Strategies)

Final Project:

Pitch Shifting(autotune)/Time Stretching/Prizmizer

Multi-Input Vocoder/Spectral Mixer?