Contact Details of Participants*

Name	University	Contact Number	Email
Theekshana Dissanayake	University of *Peradeniya*	0778495156	theekshanadis@eng.pdn.ac.lk.com
Milinda Sandaruwan	University of *Peradeniya*	0776228502	sandaruwanims@gmail.com
Shehan Erange	University of *Peradeniya*	0773082022	stanislousvanderputt1@eng.pdn.ac.lk.com

Please tick the relevant Field*

Aerospace			Multimedia Communications	
Antennas & Propagation			Power Generation & Conversion	
Asset Management			Power Systems & Equipment	
Automotive & Road Transport Sys			Power Trading & Control	
Built Environment Technologies			Project Control	
Communications Networks & Services			Power, Electrical, Machines & Drives	
Control & Automation			Radar Sonar & Navigation	
Electromagnetics			Railway	
Eng for a Sustainable Future			RF & Microwave Engineering	
Functional Safety			Robotics & Mechatronics	
Healthcare Technologies			Satellite Systems and Apps	
History of Technology			Systems Engineering	✓
Management			Tribology	
Manufacturing Enterprise			Vision and Imaging	

*Filling these sections is inevitable.

Special Note to Authors

Pages of the paper should not be more than six pages. The paper you are going to submit is strictly to be followed the following format (All characteristics of word document). **Failure to do so will be a reason to reject the papers.**

GPU Based Parallel Audio Processing System

Theekshana Dissanayake[#1], Milinda Sandaruwan[*2], Shehan Erange[#3]
Department of Computer Engineering, Faculty of Engineering, University of Peradeniya.
Peradeniya, Sri Lanka.
[1]theekshanadis@eng.pdn.ac.lk.com
[2]sandaruwanims@gmail.com
[3]stanislousvanderputt1@eng.pdn.ac.lk.com

***Abstract* -Graphic processing units have been recently used to perform verities of computational tasks including audio processing. It shows promising results for real time applications due to nature of its parallelism. In musical industry audio processing mainly done by hardware related implementations. Also there can be found software implementations that gives real-time performance for certain extend. Equalization can be found as one of major audio processing technique used in sound industry that needs real time performance. This paper discuss how to implement real time equalization GPU and how to extend this technique to parallel signal equalization. These tasks have been tested on NVidia GeForce graphic card and high performance NVidia tesla GPU. Furthermore this paper discuss how much signal parallelism can be achieved for processing audio signals for different sampling rates in real time.**

Keywords-equalization, sampling rate, optimization, occupancy, fourier transformation

INTRODUCTION

Getting use of high-level languages like C/C++ , Graphics Processing Unit (GPU) based applications run the sequential part of the program on the Central Processing Unit (CPU), with single threaded performance while parallel computations done on GPU with multi-threaded performance. This type of computing is called GPU Computing.

The main difference between GPU and CPU is that the CPU consists of a few cores optimized for sequential processing, and the GPU contains smaller but more efficient cores for parallel processing.

Generally GPU was used for games or 3D game rendering. But recently GPU is used for Parallel audio processing, financial modeling, Accelerated Graph Analytics, Circuit Simulation, Chemical Kinetics etc.

Today a typical CPUs are capable of supporting real-time audio applications, but there are many limitations. Requirement of high quality real-time parallel audio processing is ricing for large musical events such as orchestra.

Audio processing technique such as equalization is one of major audio processing technique used in musical industry. When the sampling rates of the audio signals are getting higher because of the high quality requirement (i.e. Direct Stream Digital sampling rate: 2,822,400 Hz) the audio processing system need higher performance to calculate everything in real time. Since Equalization can be done parallel, GPU can give a better performance.

In our research we first evaluate the performance between CPU and GPU using optimized algorithms to conclude that, how much real-time Processing power can be achieved using both CPU and GPU. For that we use single audio sample equalization algorithm implemented using MATLAB and CUDA C.

For the second part of our project we talk about Parallel audio signal processing using GPU. There we talk about how much number of audio signal inputs can be equalize parallel using GPU. We are going to test the CUDA C implementation on High performance Tesla GPU and typical computer NVidia GeForce GPU. In this evaluation we want to conclude that parallel audio signal Processing can be done on a typical computer with high quality sampling rates like **Direct Stream Digital** (328000Hz)**, Digital extreme Definition** (124455Hz) in real time.

RELATED WORK

On the research paper entitled Real-time adaptive algorithms using a Graphics Processing Unit is written by Jorge Lorente [1], Miguel Ferrer in 2012 discussed about adaptive channel identifier algorithms implementation in CUDA. They mentioned that the parallelism must be there in the algorithms to implement them using CUDA. In their research adaptive algorithms work sample by sample and they had to implement those algorithms using block algorithms. Furthermore their results conclude that GPU is available for real-time adaptive applications.

On the paper HIGH-PERFORMANCE REAL-TIME FIR-FILTERING USING FAST CONVOLUTION ON GRAPHICS HARDWARE by Frank Wefers [2], Jan Berg in 2010 discussed about implementation of filtering on GPU. They mentioned about fast convolution algorithms for audio rendering of complex senses. They talks about the importance of data structures to implementation of fast furrier transformation in GPU. They mentioned about functionalities, provided by CUDAFFT libraries. In our project we also used the functions that is provided by the CUDAFFT libraries. They also did a performance evaluation on data transfers between CPU and GPU. They mentioned that the "Only for larger data sizes, runtimes vs data size scale nearly

linear". Graphs that represent the data clearly indicate that statement. Also they evaluate the BATCHED FFT with CPU and GPU. They mentioned about FFTW library that used to perform furrier transform in CPU. They discussed about the potential for parallelism for higher BATCH sizes. This fact is also related to our project including parallel audio signal equalization. Finally they mentioned that they had achieved 44% computational power using GPU. This fact also proves that GPU computations can achieve higher performance.

OBJECTIVES

Phase 1:

- Compare CPU and GPU performance for optimized audio processing algorithms, implemented using MATLAB and CUDA C to conclude GPU can give a better performance for high quality sampling rates in real time.

Phase 2:

- Implement a parallel audio sample processing algorithm in CUDA C.
- Then evaluate how much parallel audio sources can be equalize using NVIDIA GEFORCE GPU, and to prove rather than buying a expensive analogue audio processing system in real time audio processing can be done in a typical computer or laptop with GPU.

SCOPE

- The real time performance limit is defined as, the algorithm can process one second sample within one second time period.

In this project the main audio processing technique is that used to evaluate performance is equalization.

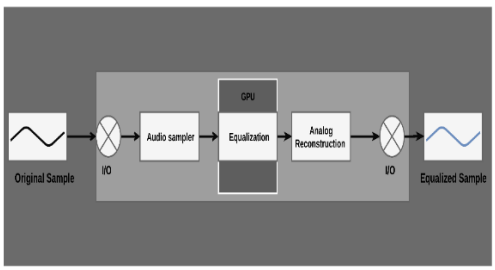

Figure 1: Overview of the equalization process

In this figure, clearly indicate the whole process happening in one second. In our project we assumed that Input and output time (I/O) and time to sample a one second audio and reconstruct the audio is negligible. Our main objective is to speed up the equalization part of the process to get real time performance.

SAMPLING RATE

Analog signals replicate the sound wave as it is but digital audio signal is constructed by getting samples at specified rate called sampling rate using Nyquist sampling theorem.

The quality of the audio signal depends on the sampling rate or bit rate. Sampling rate is defined as 'the number of samples of audio carried per second'. Bit rate defined as 'the number of bits that are conveyed or processed per unit of time'.

Sampling Rate	Audio Quality
44,100 Hz	Audio CD
96,000 Hz	DVD Audio
192,000 Hz	HD DVD
352,800 Hz	Super Audio
2,822,400 Hz	Direct Stream Digital (DSD)
5,644,800 Hz	Double-Rate DSD

Table 1: Audio sampling frequencies/rates

EQUALIZATION

General purpose of audio equalization is to make individual instruments and voice sounds better. It involves the mixing instruments, voice sounds and boosting bands of frequencies with respect to other bands of frequencies. In group of instruments play together with various notes and harmonics of instruments blend together. This blended frequencies may or may not be in human hearing range (20Hz -20000Hz). Every frequency must be treated equally to get a better sound that please the ear. In that scenario equalization comes to picture.

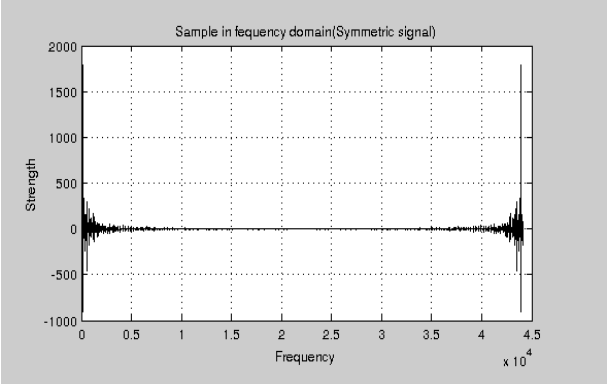

Figure 2: Real part of the signal after applying Fourier transformation

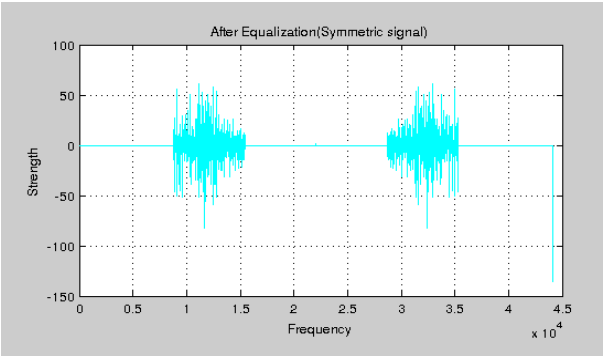

Figure 3: After manipulating strengths of selected frequencies in frequency domain

Figure 4.3 shows the frequencies of sample (x axis) and strength of each frequency (y axis). This graph only plots the real value parts of the audio signal (one second second) after applying Fourier transformation. The result in frequency domain is symmetric because converted analogue signal is a real signal. The figure 4.4 plot shows the waveform after manipulating strengths of frequencies. In this case our equalization pattern is [0 0 0 0 4 8 4 0 0].

Creating decoded audio file:

As explained above we used one second duration audio sample to processed using GPU. Sample refers to set of floating point values gathered after sampling the analogue audio signal. As an example if audio that needed to be processed is 4 seconds and the sampling rate is 44100 Hz then there are 44100*4 floating point values. We only chose 44100 values to process at a given time. Since MATLAB by default only sample audio using 44100 Hz we use some random generated floating point values to evaluate our implementation in higher sampling rates. MATLAB used to get a proper visualize picture about everything. Also in CUDA C implementation we use a text file as the audio sample.

EQUALIZATION USING MATLAB

Optimized algorithm for fast Fourier transformation is needed for the CPU side equalization. Therefore, since MATLAB itself contain optimized ***fft*** () and ***ifft*** () functions to compute Fourier transformation. Therefore MATLAB is used for CPU side equalization implementation. First audio file is needed to be sampled using ***audioread ()*** function. This function is used for both CPU and GPU implementation to convert analogue audio signal to digital signal. This function returns an array (vector) that includes digitalized values, and a sampling rate that set to default as 44100 Hz. After getting one second sample (i.e. if the sampling rate is 44100 Hz that means in 1 second there are 44100 double precision values.) that sample is converted to float array using ***simple()*** function. This procedure was followed because data types of both CPU and GPU implementation must be equal.

Then typical procedure was followed to equalize the digitalized signal. ***fft (), ifft ()*** function used to shift between time domain and frequency domain. Inbuilt functions such as ***tic, toc*** was used for the performance evaluation in MATLAB.

SINGLE AUDIO STREAM EQUALIZATION USING GPU

For a single audio stream, first the sampling values of the audio file was read and loaded in to the memory as an array. The array represents float data. Then the data array copies from host to device to process. In the device memory the data element was converted to the data type as **real**, according to the library. After that the

Data (time domain signal) is converted to the frequency domain signal using ***cufftExecR2C ()*** function in **cuFFT** library. In here the output (answers) represent complex values. Therefore it is important to know about the capacity of the device memory and the data size which we need to process here.

Returned values after calling ***cufftExecR2C ()*** are in ***cufftComplex*** data type. Since fast Fourier transformation of a real analogue signal returns symmetric signal, hence the function returns only (SAMPLE_SIZE/2 + 1) values. It must be noted that this reduce the actual needed memory to half.

After the transformation the strengths of the frequencies are changed according to the equalization pattern and the data was converted again in to the time domain signal using ***cufftExecC2R ()*** function. After coping the data to CPU memory, those values written to a text file that can be used to reconstruct the equalized audio.

PARALLEL AUDIO SAMPLE EQUALIZATION IN GPU

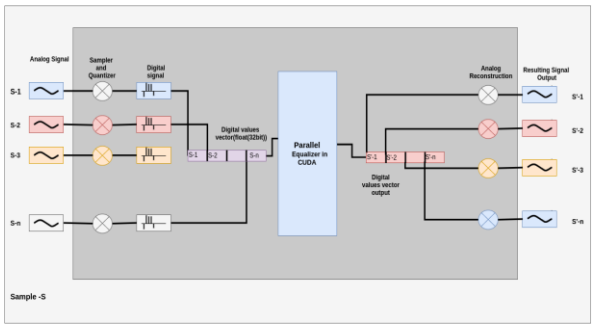

Figure 4: Parallel audio source equalization

This figure describes the whole process of parallel audio sample equalization. As explained before equalization algorithm consist of 3 main steps.

Fast Fourier transformation of audio samples was calculated using **cuFFT** library functions. This parallel sample equalization algorithm differ from single sample equalization. First **cuFFT BATCHED** functions are used to calculate the Fourier transformation of each audio samples. A vector that includes all the sample

values should be crated to do the batched Fourier transformation .Figure above clearly indicate how the vector is formed by concatenating every sample. The length of the array (vector) is given by

Length ( ) = BATCH_SIZE*SAMPLE_SIZE

In this equation SAMPLE_SIZE refers to the number of samples in a batch. As an example digitalized audio signal using sampling rate 44100Hz has sample size of 44100. It must be noted that these **cuFFT** batched functions can only compute batches that have same sampling rates. Same procedure was followed to set up a ***cufftHandle*** variable and make a CUDA plan. Another function ***cufftPlanMany ()*** was used to create the plan. Same function ***cufftExecR2C ()*** with R2C tag can be applied to compute ***fft*** but in this time the plan is different. After ***fft*** returns ***(SAMPLE_SIZE/2 + 1)*BATCH_SIZE*** values, return values are also same as the single sample one (***cufftComplex*** data type).

Function ***equalization ()*** is called on kernel after converting the audio signal to frequency domain. Since all the samples frequency domain values are in the vector, threads are allocated for each index in the vector to get higher performance and Occupancy. Constant block size ***32*32*** and varying 3D grid indexing used for generating threads.

After calling equalization function another ***cufftHandle*** plan was created to get the inverse Fourier transform of all samples.

These are the main steps that we are followed to compute the parallel audio sample equalization. And it must be noted that our inputs are text file as before. Performance evaluation was done using ***cudaEventCreate ( )*** function.

OPTIMIZATION

In this section of the report we are going to talk about things we considered for optimizing our implementation. We used NVIDIA visual Profiler for optimizing our implementation.

- Register usage:

Typically GPU multiprocessor contains larger number of registers. But when some application portion using the multiprocessor, registers divided among concurrent threads. Therefore in our implementation register usage was minimized to 9 avoid register pressure. (This occurs when there are not enough registers to complete the process). (i.e. to get register usage of particular CUDA task apply ***–ptxas – options = -v*** when compiling the code or use NVIDIA visual Profiler). Also developers can use ***–maxrregcount = N*** option to prevent the compiler from allocating larger number of register without proper use of it.

- Block size :

In GPU each multiprocessor support maximum number of block size. Typically in a GPU like NVIDIA tesla it support 1024 block size. Therefore 32*32 constant block size was used to avoid exceeding the maximum block size.

- Occupancy:

Occupancy is a measure of 'how much busy the GPU is'. More we use the resources in GPU more occupancy we can achieve. 3D grid indexing was used to get a higher occupancy. 3D indexing was used because of the limitation in grid dimension in each direction. (x, y ,z). Optimization of the algorithm was done using NVidia visual profiler. 78% of occupancy level was achieved after the optimization.

ACHIEVEMENTS

- Tested devices

Device Name	Multiprocessor count	Maximum threads per block	Maximum grid size	Peak memory bandwidth
Tesla c2075	14	1024	(65535 x 65535 x 65535)	5 Gb/s
GeForce 920M	2	1024	(2147483647 x 65535 x 65535)	500 Mb/s
Intel Core i5 CPU	4 Cores		4 Gb memory	

Table 2: Tested devices

RESULTS

Performance evaluation on CPU and GPU was done using following equations. Performance evaluation only considered computation time of equalization. For GPU implementation it was assumed that data that needed to be processed is already in the RAM.

CPU Time = [FFT_Time + EQ_Time + IFFT_Time] (s)

GPU Time = [mem_copy_H2D +FFT_Time + EQ_Time + IFFT_Time+ mem_copy_D2H] (s)
{FFT - fast Fourier transformation, H2D H-Host CPU D-Device GPU}

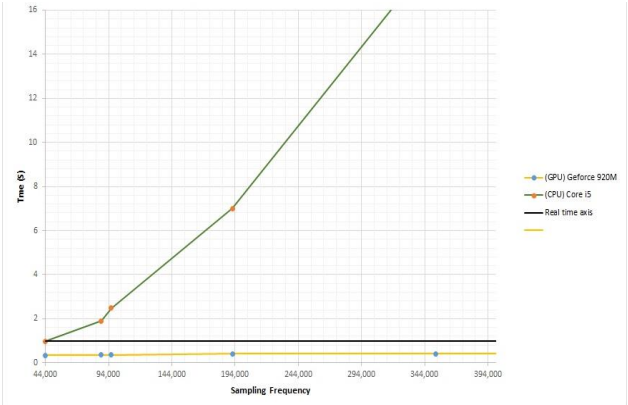

Figure 5: Processing time for an audio stream vs sampling frequencies

Figure 5 shows the time taken by the CPU and GPU in seconds (y axis) to process a given audio sampling rate (x axis). These audio sample rates are commonly used in musical industry. There are 3 plots where each is for different tested device as given in the legend. This graph can be used to conclude that the CPU can't process high quality audio sampling rate in real time. The real time performance limit is defined as, the algorithm can process one second samples within one second time period. This graph clearly indicate that CPU has lack of performance when sampling rate is rising.

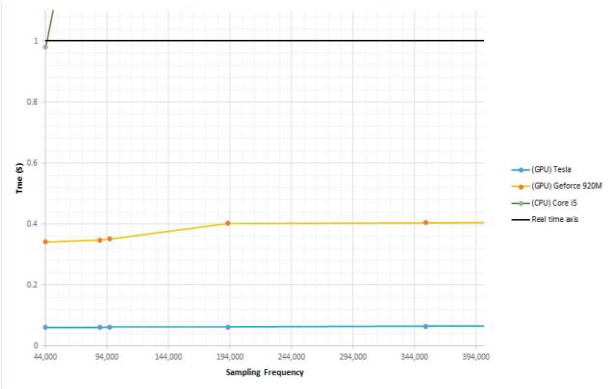

Figure 6: Performance evaluation between NVIDIA Tesla and NVIDIA GEFORCE

Figure 6 Shows more abstract view of 2 GPUs, NVIDIA Tesla and NVIDIA GEFORCE 920M. It indicate Tesla GPU has 5 x performance comparing to GEFORCE.

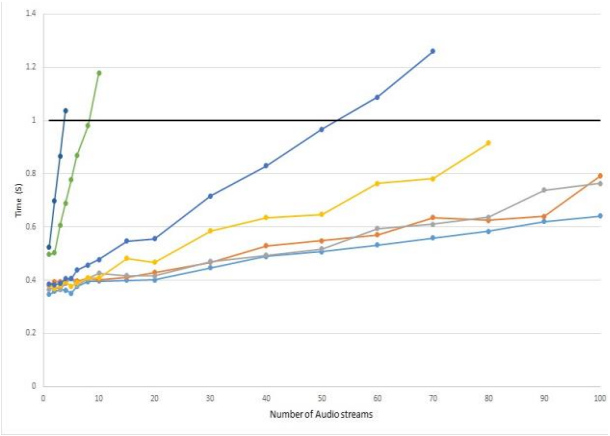

Figure 7: Processing time for GEFORCE 920

Figure 7 shows the time taken by GPU in seconds (y axis) to process a given batch size (i.e. number of sources) (x axis).There are 7 plots where each plot is for different sampling rates as given in the legend. This graph can be used to determine the maximum batch size (number of sources) that can be processed in real time using GFEFORCE GPU. According to this 50 parallel sources of sampling rate 352,800 Hz can be equalize in real time using a typical laptop GPU less than 1 second.

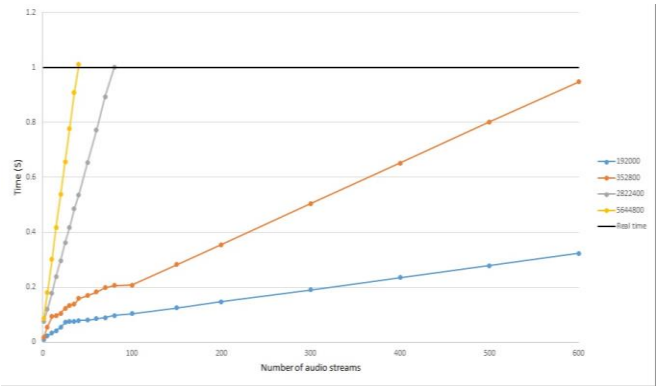

Figure 8: Processing time of Tesla vs number of audio streams

Figure 8 shows the time taken by GPU in seconds (y axis) to process a given batch size (i.e. number of sources) (x axis).There are 4 plots where each is for different sampling rates as given in the legend. This graph can be used to determine the maximum batch size that can be processed in real time using Tesla GPU. According to this 40 parallel sources of sampling rate 5,644,800Hz can be equalize real time using GPU. In this case 79 % occupancy was achieved on the GPU and 900MB of data was transferred and processed using the GPU.

CONCLUSION

Sampling Rate (Hz)	Time in CPU (s)	Time in NVidia G-Force GPU (s)	Approximate Speedup
44100	0.98	0.34	×3
88200	2.49	0.35	×7
192000	6.99	0.40	×17
352000	18.6	0.41	×47

Table 3: Speed up archived using GPU

According to table 3, GPU gives better performance to higher sampling rates. Also CPU can be used to process audio signals with lower sampling rates like 44100Hz. But for the real time performance, it is better to use a GPU. By doing the equalization part in GPU speedup the performance **×47**. Also since GPU has higher data transfer rates, time for copy data to GPU and get back the data from GPU is negligible.

According to the figure 7 typical NVIDIA GEFORCE graphic card can process high quality audio samples where audios sampled using super audio (table 4.2) sampling rate. According to figure 7 GPU can process 50 parallel audio source signals in real time. By considering real world applications this shows promising results for parallel audio processing system. Time for sample the audio and reconstruct the audio must be considered in typical scenario. By considering those time values (assume time for both operations takes 0.5 seconds) this system at least can process 20 parallel sources in real time. Also it must be noted that NVIDIA GEFORCE graphic card is not suited for audio qualities like DSD and Double-rate DSD. High performance GPU

like Tesla should be used for achieve real time performance for those sampling rates.

REFERENCES

[1]http://ieeexplore.ieee.org/document/6868998/?reload=true

[2]http://dafx10.iem.at/proceedings/papers/WefersBerg_DAFx10_P76.pdf

[3]https://en.wikipedia.org/wiki/Sampling (signal_processing) #Sampling_rate
Accessed on 4th of December 2016 at 10.20 p.m.

[4] Sampling Theory for Digital Audio by Dan Lavry, Lavry Engineering, Inc.

[5] INTRODUCTION TO Signal Processing Sophocles J. Orfanidis Rutgers University

[6] 1999-2002 by Sanjeev R. Kulkarni.Lecture Notes for ELE201 Introduction to Electrical Signals and Systems.

[7] Audio Equalizer Ohio State University Department of Electrical and Computer Engineering March 2009 by Betty Lise Anderson

[8] CUDA FFT Documentation
http://docs.nvidia.com/cuda/cufft/#axzz4RtDvBg2r
Accessed on 6th of December 2016 at 01.18 p.m.

[9] CUDA occupancy
http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#axzz4RtDvBg2r
Accessed on 4th of December 2016 at 10.20 p.m.

[10] MATLAB FFT ()
https://in.mathworks.com/help/matlab/ref/fft.html
Accessed on 6th of December 2016 at 01.18 p.m.

[11] Programming Massively Parallel Processors

[12] Tutorial on GPU computing With an introduction to CUDA Felipe A. Cruz Tutorial on GPU computing With an introduction to CUDA University of Bristol, Bristol, United Kingdom.