# GPU Acceleration

Tutorial on High-Performance & Accelerated Computing
at IEEE ICIAfS 2016

Dec. 18, 2016

Hasindu Gamaarachchi

hasindu2008@gmail.com

# Graphics Processing Unit (GPU)

## GPU is the Specialized hardware that renders the graphics output to a display on a computer

- Modern graphics(such as games) include 3D motion, light effects, object transformations, etc.
- CPU is for general purpose calculations, and so cannot efficiently handle them.
- GPU is a hardware specially designed to handle them

# Dedicated Graphics

Some modern dedicated graphics cards


NVIDIA GTX 1080


AMD Radeon R9 280X

# NVIDIA Graphics Processors

## Geforce

For gaming


## Tesla

For high performance computing

# NVIDIA graphics cards

## Quadro

For computer-aided design



## Tegra

Graphics for mobile devices



## NVS

Multi display

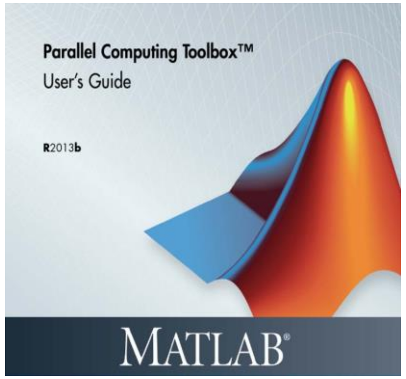
# GPGPU

- GPGPU is General Purpose Computation using Graphics processing Units.


- Bioinformatics
- Computational chemistry and physics
- Numerical analysis
- Machine learning
- Computational finance
- Imaging and computer vision
- Weather and climate
- Medical imaging
- Digital signal processing
- And many more

# How to do GPGPU

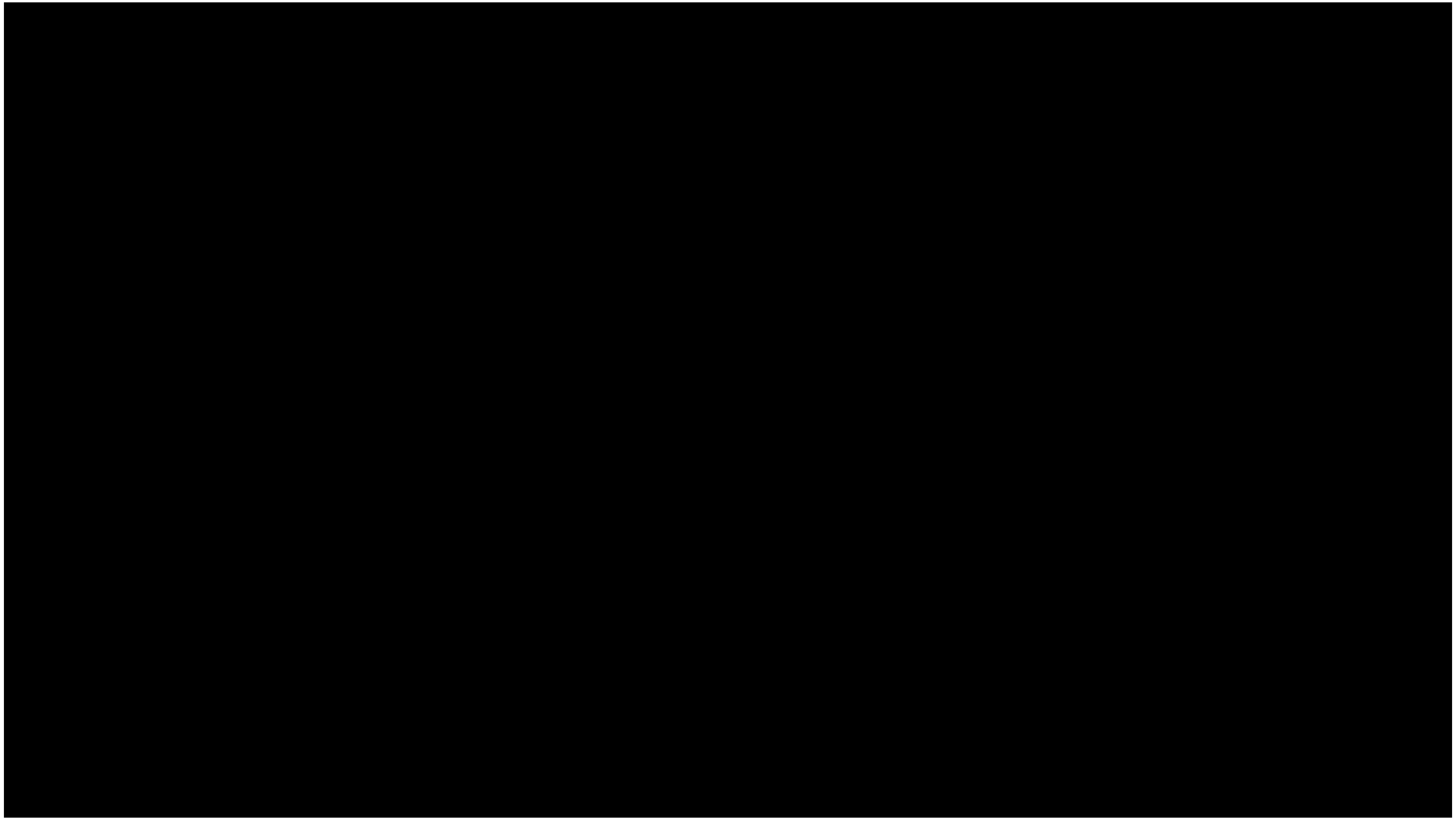
- CUDA C
- OpenCL
- Matlab (Parallel Computing Toolbox)




We focus on CUDA C. CUDA C is for NVIDIA GPUs only. But for NVIDIA GPUs this is the most efficient method.

# CPU vs GPU

https://www.youtube.com/watch?v=-P28LKWTzrI



# CPU vs GPU

Intel Xeon Processor E7-8890 v4

- 24 cores
- 48 threads
- 3.4 GHz clock rate
- 60MB cache memory
- 165W TDP

NVIDIA Tesla K40

- 2880 cores
- 30720 threads
- 875MHz clock rate
- 1.536 MB cache memory
- 235W TDP

# CPU vs GPU

| CPU | GPU |
|---|---|
| Small number of cores /threads | Very large number of cores/threads |
| Large clock rate | Less than the clock rate of CPU |
| Large cache | Very small cache |
| Powerful cores (With respect to : pipelining, branch prediction, forwarding, ALU) | Less powerful cores |

Improved latency
Good for serial algorithms

Improved throughput
Good for parallel algorithms

# CPU vs GPU : What each is good at?

```
for (i=0;i<1024;i++){
    num = num % randint();
}
```

- Next result is dependent on the previous result
- Serial algorithm.
- Faster the clock rate is better.
- No use of threads or cores.
- Good for CPU.

```
for (i=0;i<1024;i++){
    C[i] = A[i] + B[i] ;
}
```

- Each operation is independent of the other
- Parallelizable algorithm
- If there are 1024 threads each thread can handle each operation
- Good for GPU

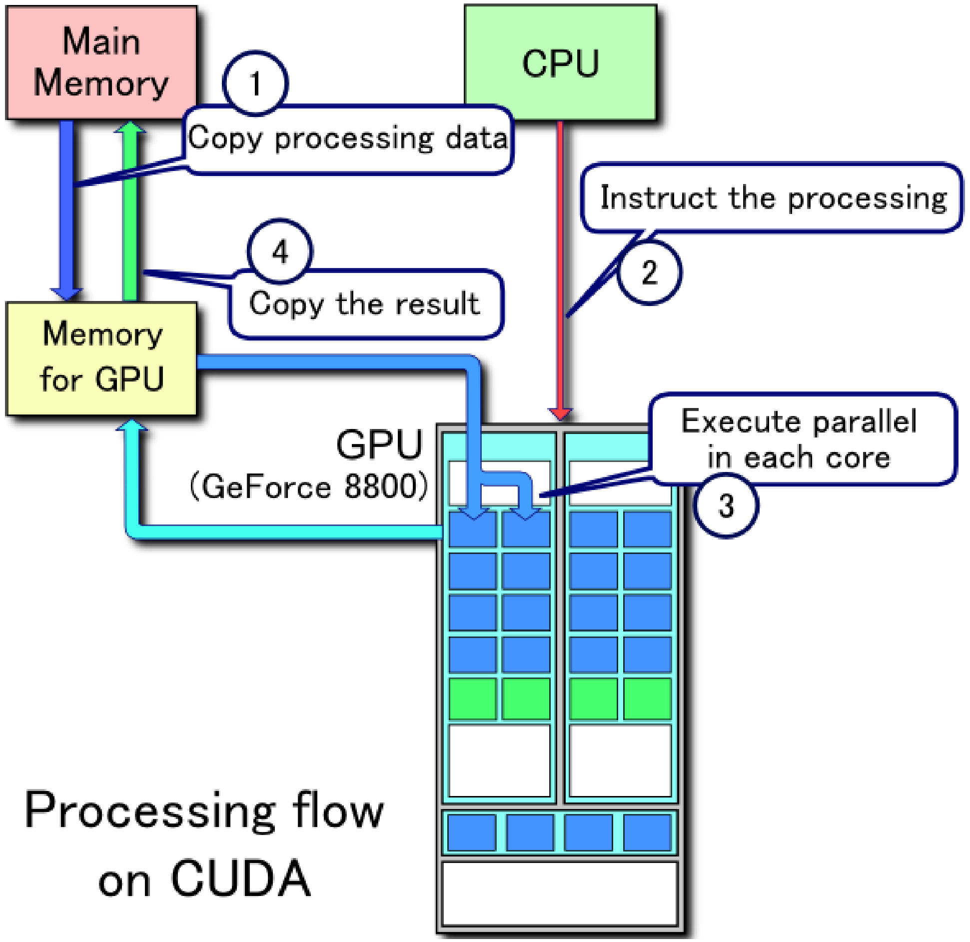
# CUDA

- CUDA stands for Compute Unified Device Architecture

.

- Parallel computing platform and programming model invented by NVIDIA.

- Enables using NVIDIA GPUs for GPGPU.




Processing flow on CUDA

# CUDA Toolkit

Provides a development environment for  C and C++ developers building GPU-accelerated applications.

Contains

- compiler for NVIDIA GPUs
- CUDA libraries
- Debugging tools
- Visual Profiler
- Nsight integrated development environment

Available for Windows, Linux and Mac OS

# CUDA C

- For programming the GPU, C language has been extended by NVIDIA to include GPU specific features. This is called CUDA C.

- Include special syntax

- Include high level functions for GPU tasks such as GPU memory allocation, memory copying from CPU to GPU and vice versa

# CUDA C

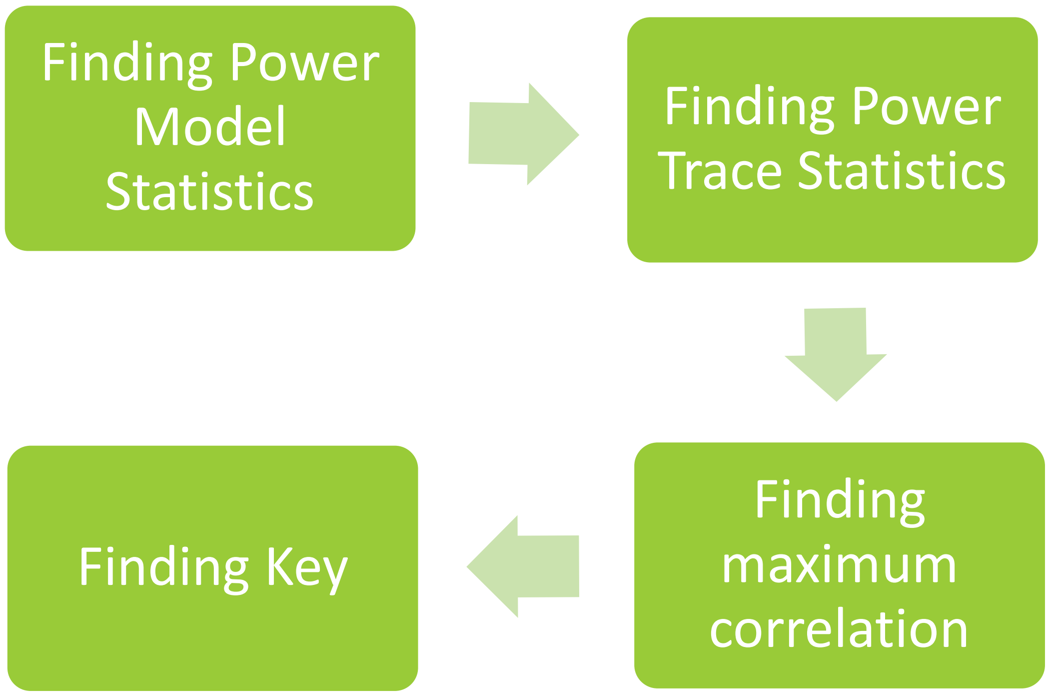
## Standard C Code

```
void addVector(int  *C, int *A, int *B)
{
      for(int i=0;i<SIZE;i++){
            C[i]=A[i]+B[i];
      }

}


addVector(C,A,B)
```

## CUDA C Code

```
__global__  void addVector(int *C, int *A, int *B )
{
      int tid = blockDim.x * blockIdx.x + threadIdx.x;
      if(tid<SIZE)  C[tid]=A[tid]+B[tid];

}

cudaMemcpy(A, A_RAM,sizeof(int)*SIZE,cudaMemcpyHostToDevice);
cudaMemcpy(B,B_RAM,sizeof(int)*SIZE,cudaMemcpyHostToDevice);

int numBlocks = ceil(SIZE/(float)256); int threadsPerBlock = 256;
addVector<<<numBlocks,threadsPerBlock>>>(C, A, B);

cudaMemcpy(C_RAM,C,sizeof(int)*SIZE,cudaMemcpyDeviceToHost);
```

# CPA Algorithm




- Algorithm of huge computational complexity
- Needs to process lot of power traces
- Hence takes a lot of time on a CPU
- But highly parallelizable

# Phase 1 : Finding Power Model Statistics

## Key in AES

128 bits

| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 8 | Byte 9 | Byte 10 | Byte 11 | Byte 12 | Byte 13 | Byte 14 | Byte 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- Consists of 16 bytes (let's call byte position)
- each byte can take values from 0 to 255 (let's call subkey)
- There are 256x16 combinations

**In CPU**

*For each byte position from 0 to 15*

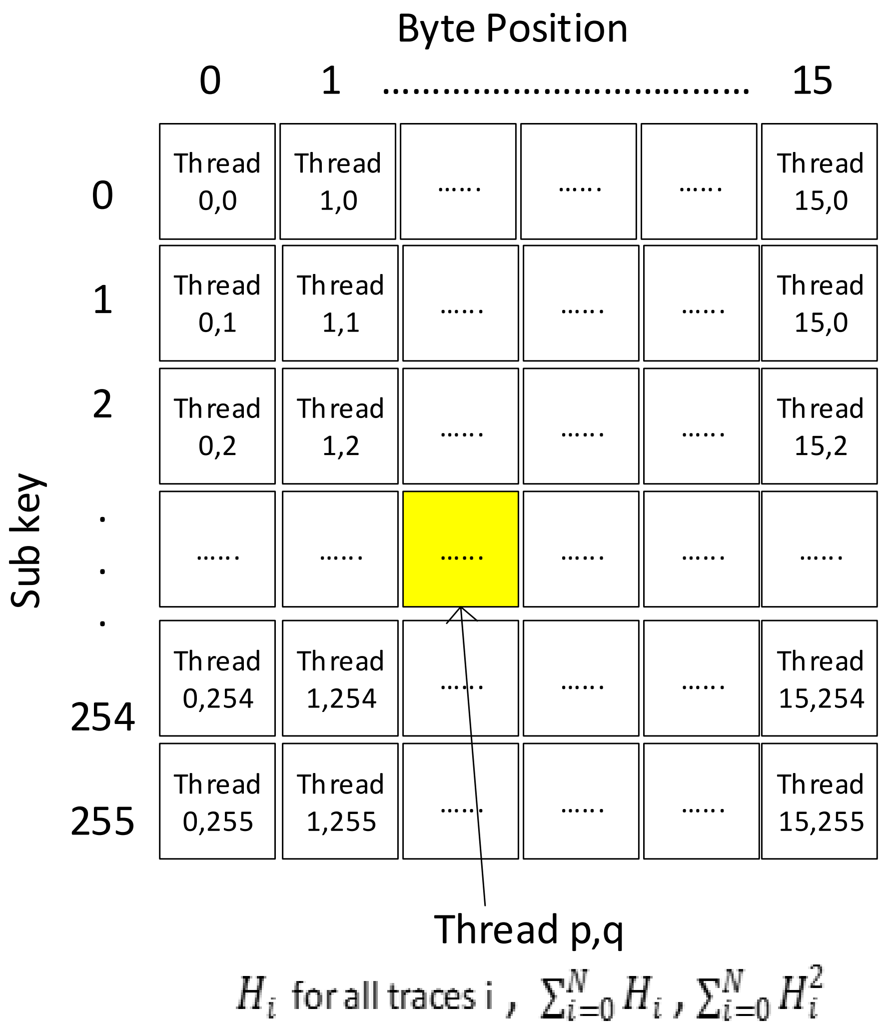
*For each subkey from 0 to 255*

$H_i$ *for all traces i,* $\sum_{i=0}^{N} H_i$, $\sum_{i=0}^{N} H_i^2$

# Phase 1 in GPU

- Assign a thread for each combination
- Thread structure is 2D

CUDA kernel with 2D thread indexing model



# Phase 2 : Finding power trace statistics

- A power trace has *m* number of samples ( m is large as 100 000)
- Must calculate power trace statistics for each sample point for all sub keys for all byte positions

**In CPU**

*For each byte position from 0 to 15*

*For each subkey from 0 to 255*

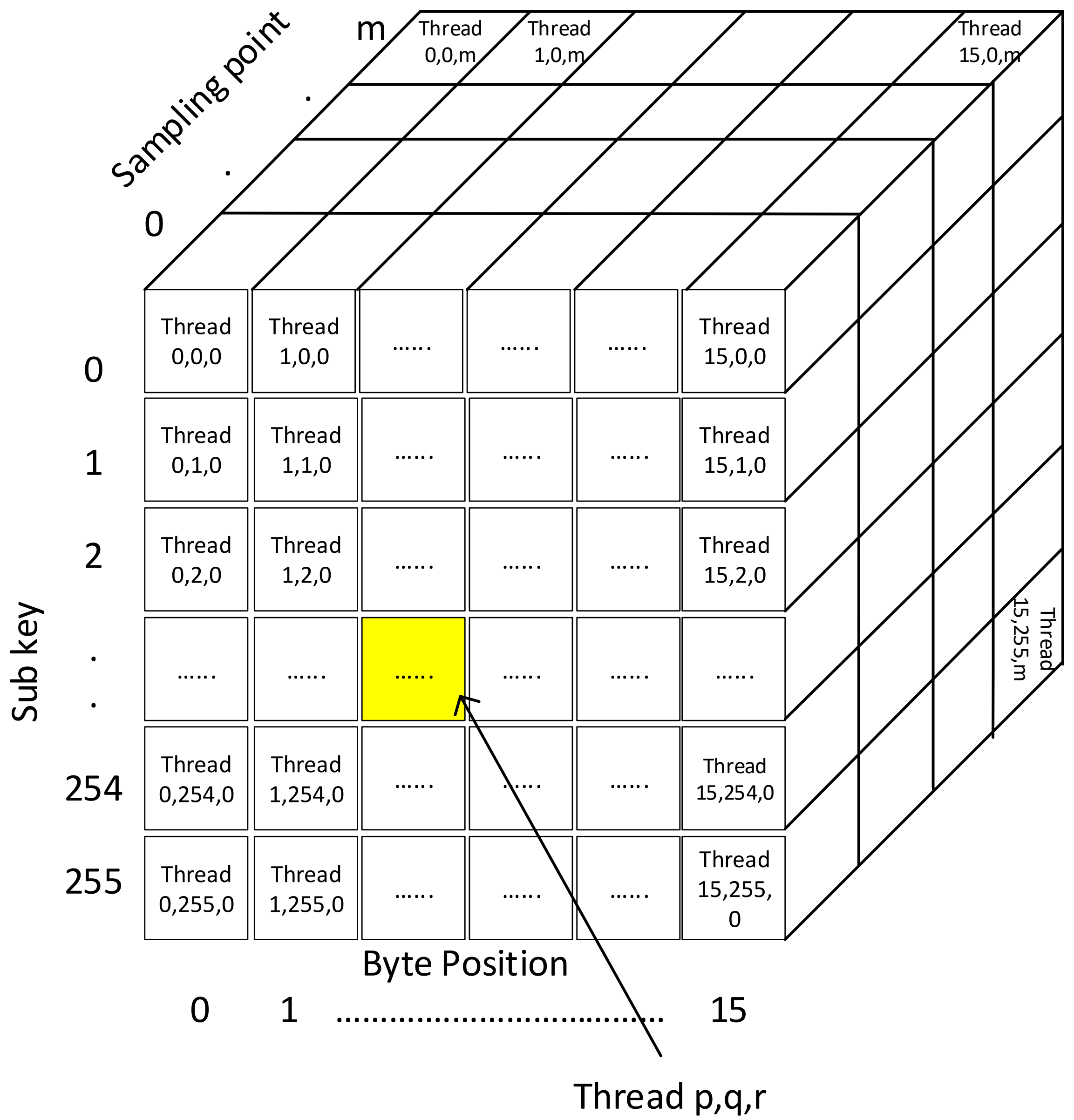
*For each samplepoint from 0 to m-1*

$$\sum_{i=0}^{N} W_{i,j}, \sum_{i=0}^{N} W_{i,j}^{2}, \sum_{i=0}^{N} W_{i,j} H_i$$

# Phase 2 in GPU

- A thread is assigned to each combination
- Thread structure is a 3D

This is the most compute intensive phase
Therefore CUDA kernel with 3D thread indexing model




$\sum_{i=0}^{N} W_{i,j}$ , $\sum_{i=0}^{N} W_{i,j}^2$ , $\sum_{i=0}^{N} W_{i,j} H_i$

# Phase 3 : Finding maximum correlation

- For each combination of sub key and byte position find maximum Pearson correlation coefficient.

**In CPU**

*For each byte position from 0 to 15*

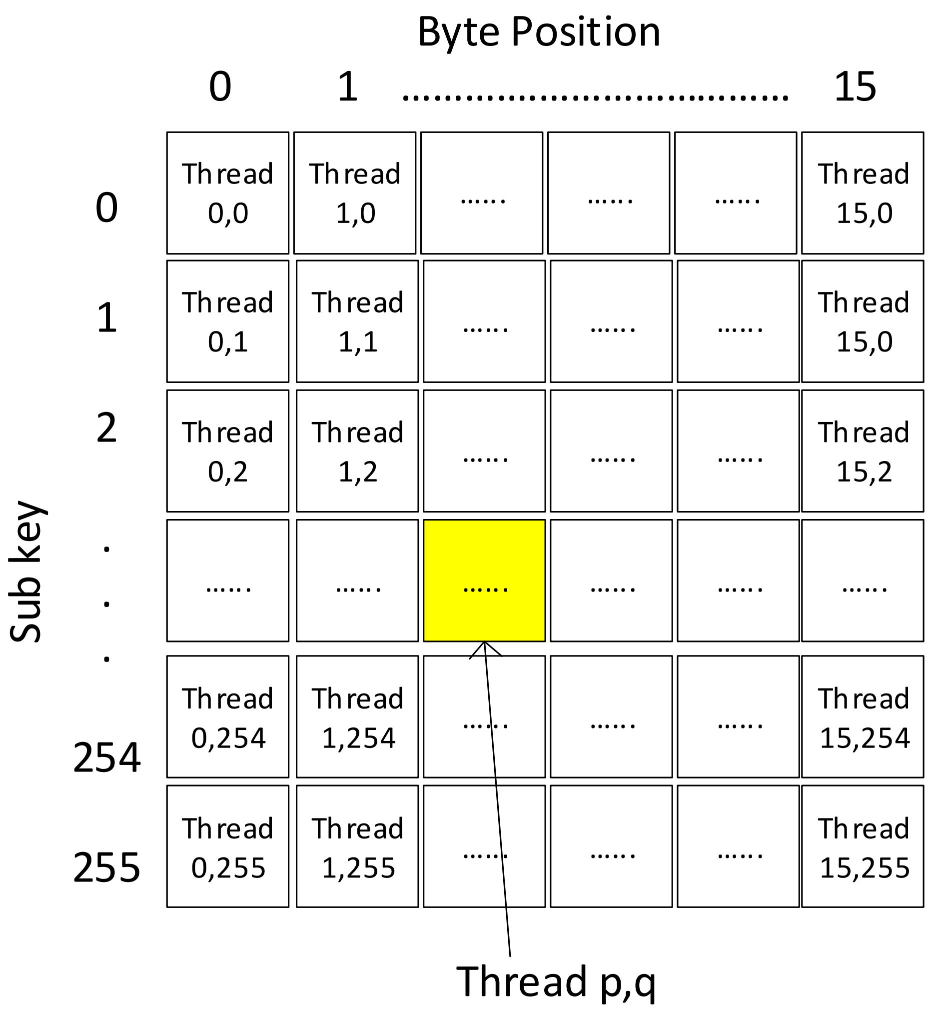
*For each subkey from 0 to 255*

$$\hat{\rho} = \frac{N\sum_{i=0}^{N} W_{i,j}H_i - \sum_{i=0}^{N} W_{i,j}\ \sum_{i=0}^{N} H_i}{\sqrt{N\sum_{i=0}^{N} W_{i,j}^2 - \left(\sum_{i=0}^{N} W_{i,j}\right)^2}\sqrt{N\sum_{i=0}^{N} H_i^2 - \left(\sum_{i=0}^{N} H_i\right)^2}}$$

# Phase 3 in GPU

- Thread model is 2D

CUDA kernel with 2D thread indexing model




$$\hat{\rho} = \frac{N\sum_{i=0}^{N} W_{i,j}H_i - \sum_{i=0}^{N} W_{i,j}\ \sum_{i=0}^{N} H_i}{\sqrt{N\sum_{i=0}^{N} W_{i,j}^2 - \left(\sum_{i=0}^{N} W_{i,j}\right)^2}\ \sqrt{N\sum_{i=0}^{N} H_i^2 - \left(\sum_{i=0}^{N} H_i\right)^2}}$$

# Phase 4 : Deriving the round key

- For each byte position, sub key with maximum correlation is selected
- 1D thread model : Not much complexity. So done on CPU

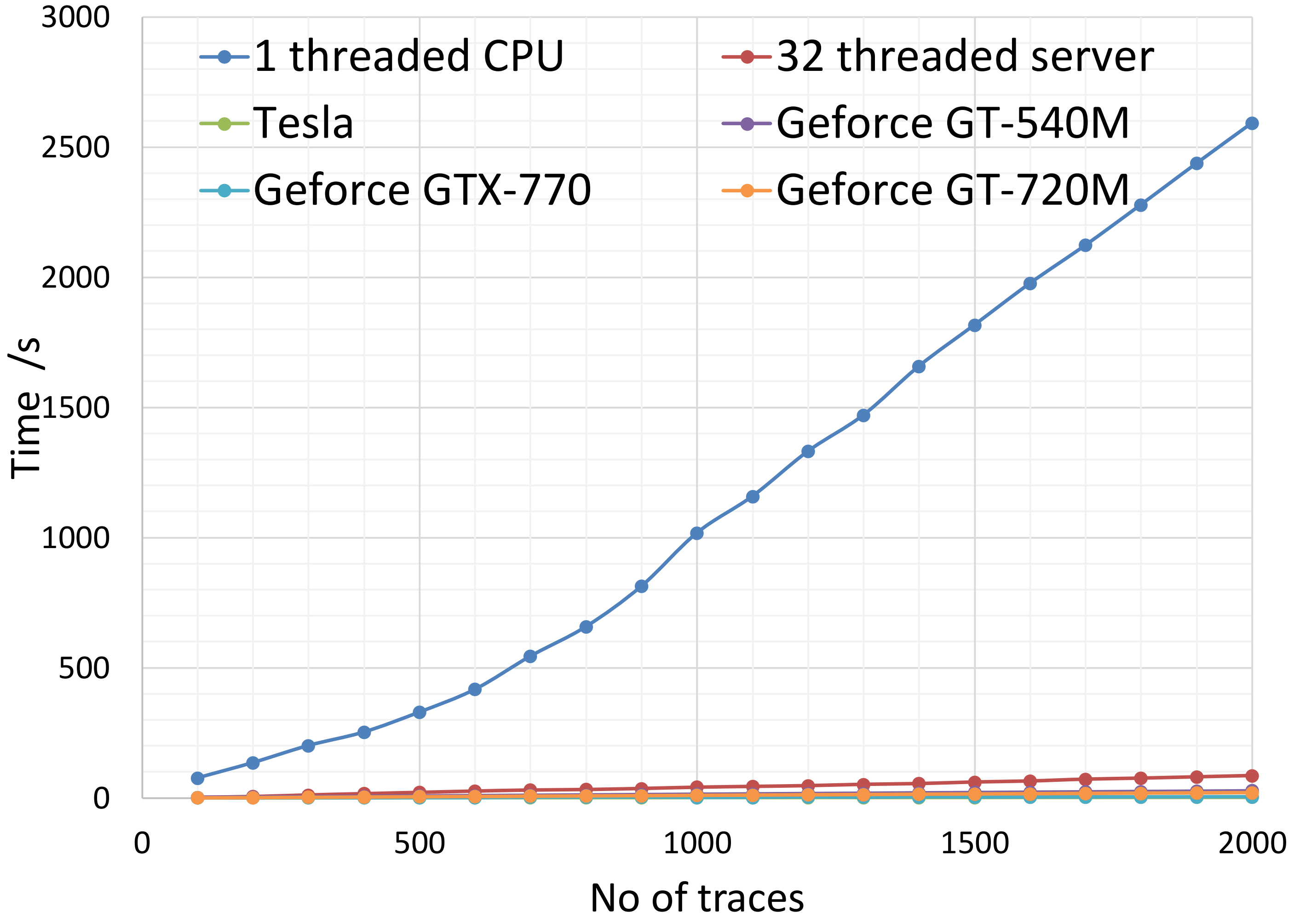
| **Byte 0** | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** | **Byte 7** | **Byte 8** | **Byte 9** | **Byte 10** | **Byte 11** | **Byte 12** | **Byte 13** | **Byte 14** | **Byte 15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ | $\hat{\rho}0$ |
| $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ | $\hat{\rho}1$ |
| … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … |
| $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ | $\hat{\rho}255$ |
| max | max | max | max | max | max | max | max | max | max | max | max | max | max | max | max |

# Optimization we achieved …

| CUDA kernel | Consumed time as a percentage | Performance Limiter |
| --- | --- | --- |
| **Phase1 (Power model statistics)** | 00.0% | Instruction and Memory Latency |
| **Phase2 (power trace statistics)** | 85.8% | Computation |
| **Phase3 (maximum correlation)** | 14.2% | Computation |

- Phase 2 is most compute intensive phase
- Limitation is caused by computation which is mostly a hardware limitation

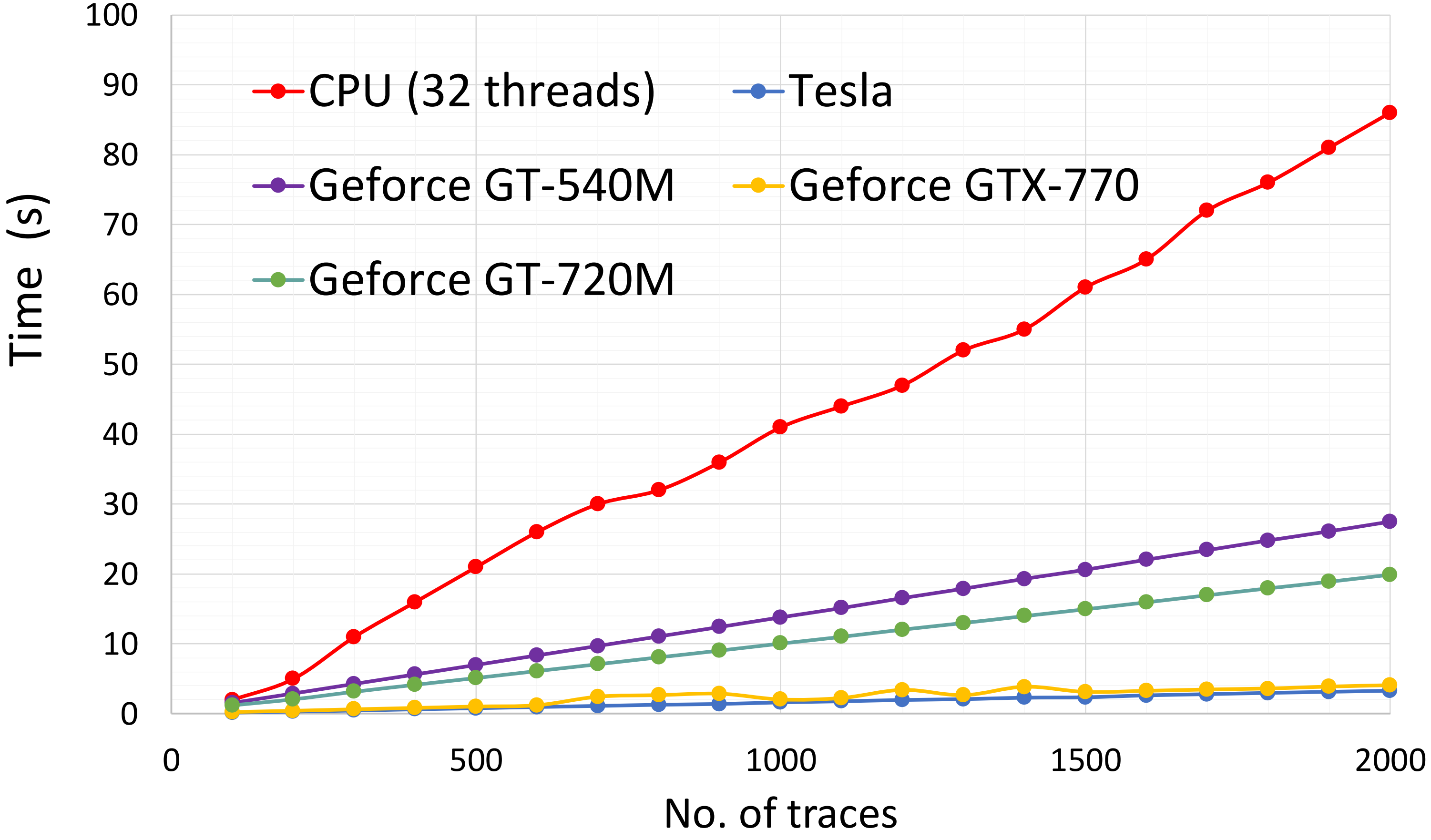
# Results




CPA on NVIDIA Tesla C2075 is,

- More than 1300 faster than a single threaded CPU

# Results




CPA on NVIDIA Tesla C2075 is,

- More than 60 times faster than a 32 threaded high performance server

# More info

**CUDA**

https://feels.pdn.ac.lk/course/view.php?id=476

https://feels.pdn.ac.lk/course/view.php?id=426

Login as guest

**Accelerating Correlation Power Analysis using Graphics Processing Units (GPUs)**

*Hasindu Gamaarachchi, Roshan Ragel, and Darshana Jayasinghe,*
*Information and Automation for Sustainability (ICIAfS), 2014 7th International Conference on, 22-24 Dec. 2014*

# More info

Download the slides from :

http://tesla.ce.pdn.ac.lk/iciafs/gpu.pdf

CPA implementations :

https://github.com/hasindu2008/PowerAnalysis/tree/master/4.analysis

# NVIDIA Research Center in Peradeniya

https://tesla.ce.pdn.ac.lk/

**NVIDIA Tesla K40**

- 2880 cores
- 30720 threads


**NVIDIA Tesla C2075**

- 448 cores
- 21504 threads


**Temporary account for 1 week**

```
SSH server : tesla.ce.pdn.ac.lk
User : iciafs
Password : Ic1Af32016
```

Contact : roshanr@pdn.ac.lk
hasindu2008@gmail.com

# Questions?