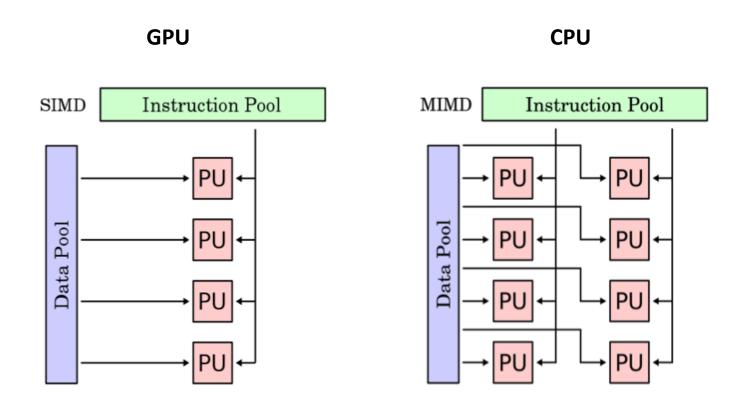
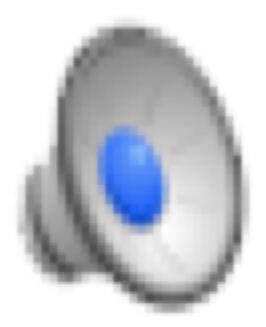
# High Performance Python Lab Term 2 2020/2021

Lecture 7. Python Multithreading, Python Multiprocessing, GPU programming with CuPy

## Flynn's taxonomy



# Why GPUs?



### Chickens vs oxen

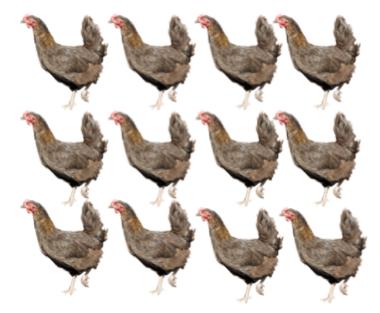


Seymour Roger Cray

"If you were plowing a field, which would you rather use: two strong oxen or 1024 chickens?"

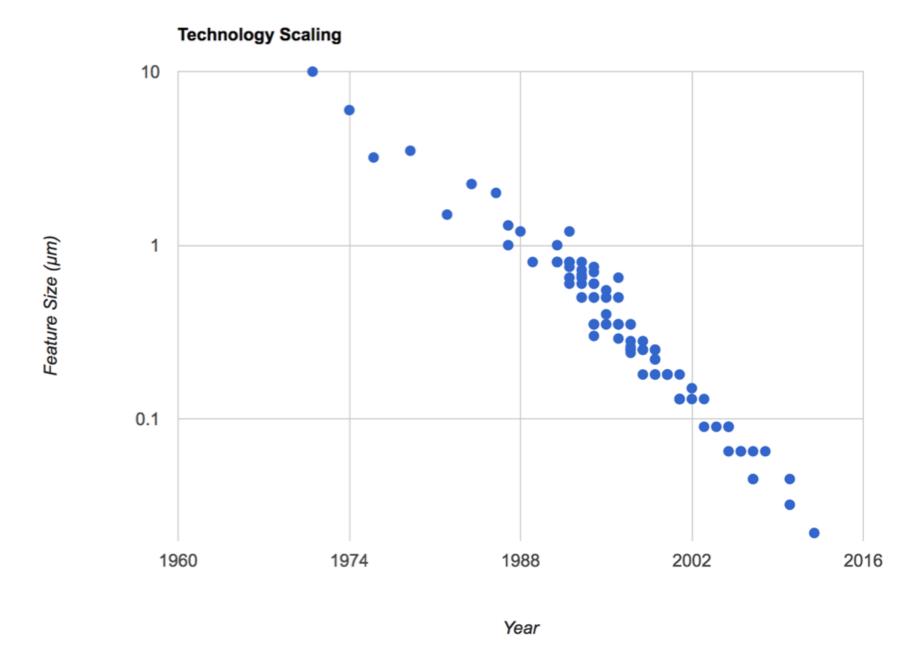


Modern CPU ~10 cores

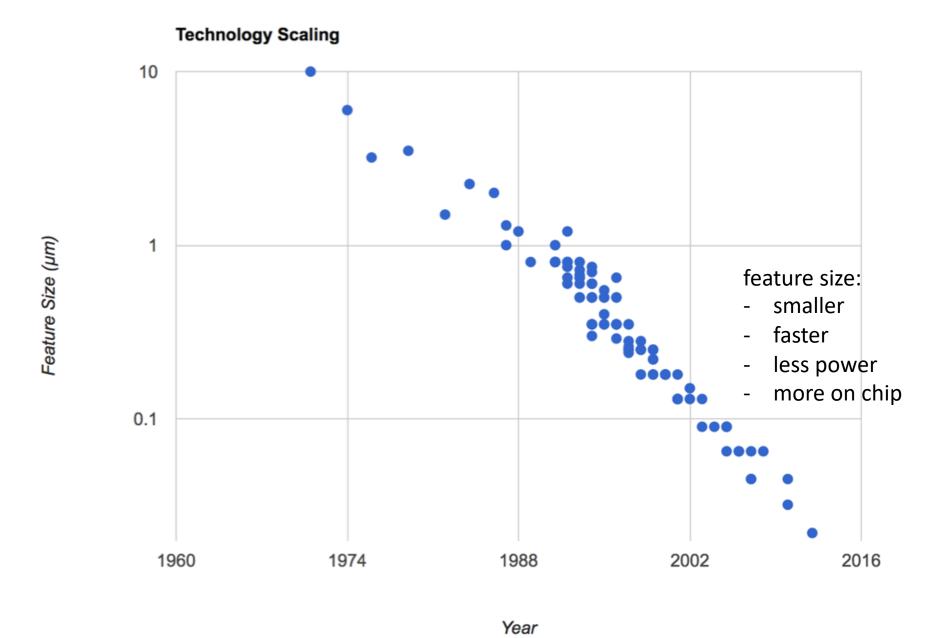


Modern GPU ~5000 cores

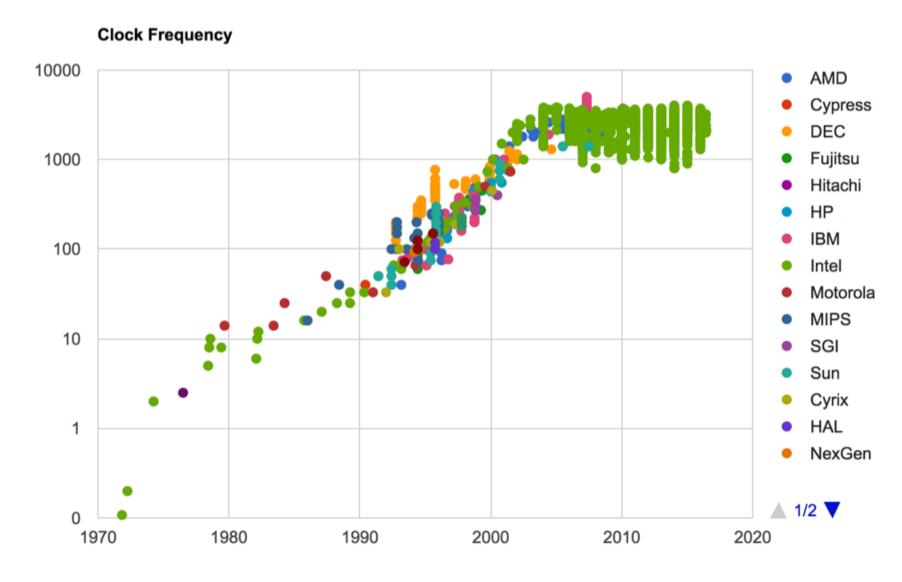
## CPU Database (cpudb.stanford.edu)



## CPU Database (cpudb.stanford.edu)



### CPU Database (cpudb.stanford.edu)



Can't clock further: **HEAT** 

### Power is the new constraint

### Consequence:

- Better to build smaller more efficient processors (in terms of power)
- More of them

### What processors can we build if power is the main constraint?

Modern **CPUs** are great all-around processors capable of doing all sorts of things:

- + flexibility + performance
- power consumption (expensive flops/watt)

Modern **GPUs** have simpler control structure (simpler processing units)

- + simpler processing units (many of those)
- + more power efficient (more flops/watt)
- you have to think how to program those

LatencyThroughput(time)vs(tasks/time)(seconds)(flops)

Latency

(time)

(seconds)

VS

Throughput

(tasks/time)

(flops)

### **Supermarket**

cash desk



cash desk



cash desk





**CPU design** -- design from the point of view of customers (want to get out of supermarket as fast as possible, even though the cashier will not have any tasks to do afterwards, optimize for latency)

**GPU design** – design from the point of view of the supermarket owner (cashiers are getting paid for the whole day – they better have constant work, optimize for throughput)

Moscow St. Petersburg



200 km/hour 2 people

Latency =

Throughput =



50 km/hour 40 people

Latency =

Throughput =

Moscow

800 km

St. Petersburg





200 km/hour 2 people

Latency = 4 hours

Throughput = 0.5 people/hour



50 km/hour 40 people

Latency = 16 hours

Throughput = 2.5 people / hour

### Core GPU design tenets

- 1. Lots of simple compute units that trade simple control for more compute;
- 2. Explicitly parallel programming model;
- 3. Optimized for throughput not latency

### **NVIDIA CUDA**

- 1. CUDA is a parallel computing platform and application programming interface (API) that can be used on NVIDIA GPU cards.
- 2. Proprietary with closed source compilers (unlike OpenCL).

#### **CUDA 8.0:**

- CUDART CUDA RunTime library, see docs 丞

- CUSOLVER CUDA based collection of dense and sparse direct solvers, see main 

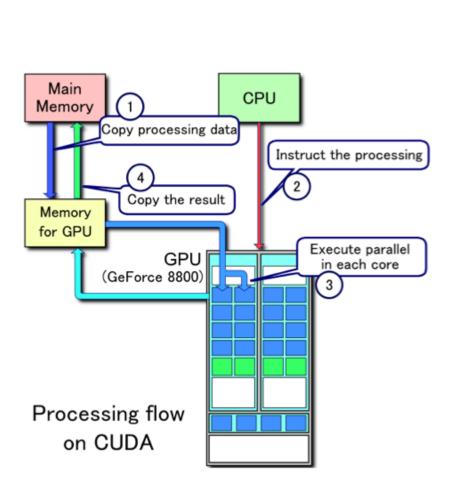
   and docs

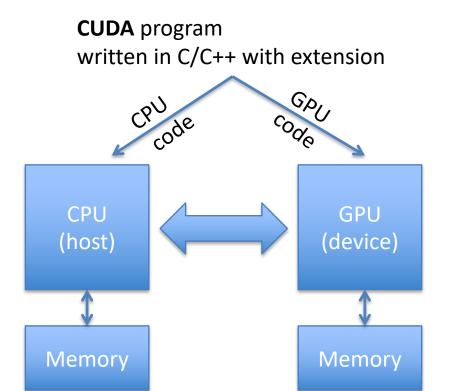
- NVRTC NVIDIA RunTime Compilation library for CUDA C++, see docs ☑

#### CUDA 9.0:

• CUTLASS 1.0 - custom linear algebra algorithms, see CUDA 9.2 News₺, Developer News₺ and dev blog₺

### CUDA workflow and conventions





**CPU** is the master and is responsible for:

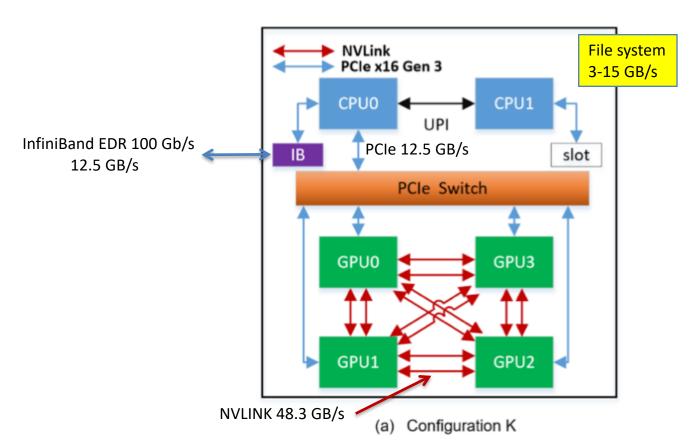
- 1) moving data from CPU mem to GPU mem
- 2) moving data from GPU mem to CPU mem
- 3) allocating memory on the GPU
- 4) launching kernels on GPU ("Host launches kernels on the device")

(1,2 – cudaMemcpy; 3 – cudaMalloc)

### Typical CUDA program

- 1. CPU allocates and initializes storage on CPU (malloc)
- CPU allocates storage on GPU (cudaMalloc)
- CPU copies data to GPU (cudaMemcpy)
- CPU launches kernels on GPU (we'll learn how)
- CPU copies data back from GPU (cudaMemcpy)

Remember: Try to minimize copying between the CPU and GPU!!!



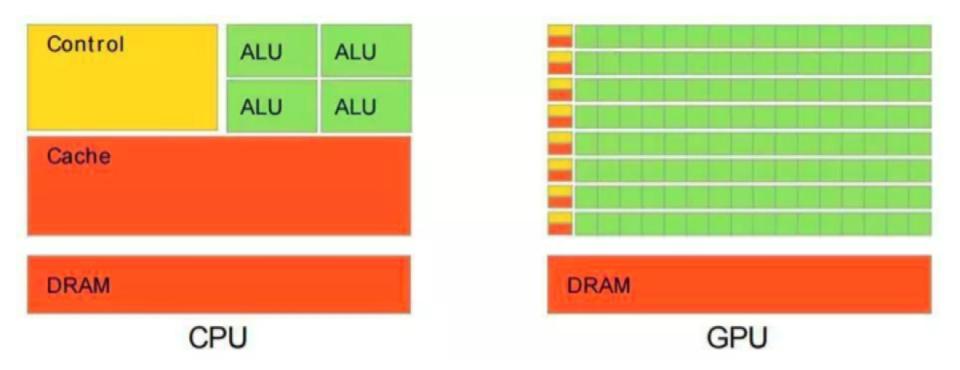
### Core concept of CUDA

#### Come back to this slide after doing examples: make sure you understand this

- 1. Kernels look like serial programs
- 2. Write your program as if it will run on one thread
- 3. The GPU will run that program on many threads (you tell the GPU how many threads to launch)

#### **GPUs are good at:**

- 1. Efficiently launching lots of threads (you can tell the GPU to run 10000000 of threads)
- 2. Running lots of threads in parallel (GPU has a maximum of threads it can run at once)



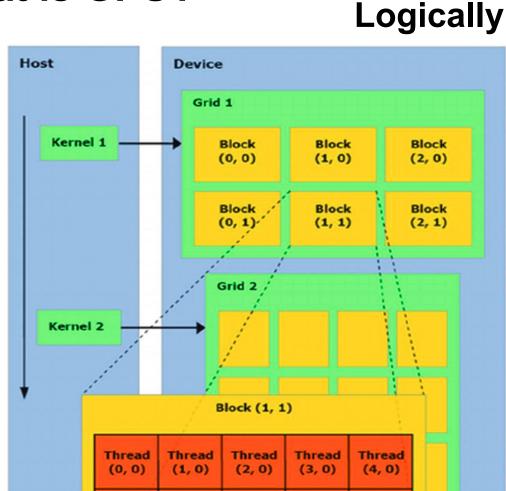
### What is GPU?

**Physically** 

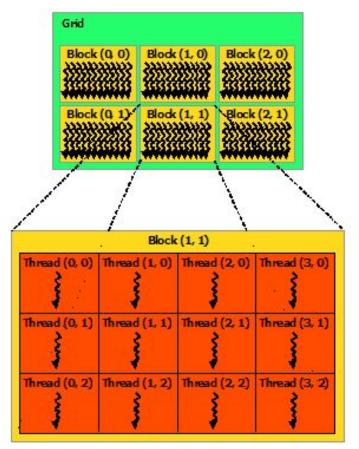
L2 Cache

**GPU Memory** 

**System Memory** 



## **Compute Unified Device Architechture (CUDA)**





- Kernel is a serial program for one thread
- Each thread executes a kernel
- You can tell the GPU how many threads to launch
- Threads are grouped in blocks, blocks -- in grids

## CUDA + Python (CuPy)

CuPy is...

https://cupy.chainer.org/

a library to provide NumPy-compatible features with GPU





```
import numpy as np
X_cpu = np.zeros((10,))
W_cpu = np.zeros((10, 5))
y_cpu = np.dot(x_cpu, W_cpu)
```

```
import cupy as cp
x_gpu = cp.zeros((10,))
W_gpu = cp.zeros((10, 5))
y_gpu = cp.dot(x_gpu, W_gpu)
```

```
y_{cpu} = cp.asnumpy(y_gpu)
```

```
y_gpu = cp.asarray(y_cpu)
```

## CUDA + Python (CuPy)

- Elementwise kernels
- Reduction kernels
- "Raw" kernels (to use this -- a dive into C required)

More Info: https://tinyurl.com/u8xqo9

### **Tasks**

- Saxpy
- CuPy-based bifurcation map
- Histogram
- Image blur

Detailed tasks are in jupyter notebook

### Task 10. Saxpy

# Single precision Alpha X Plus Y (SAXPY)

Part of Basic Linear Algebra Subroutines (BLAS) Library

 $z = \alpha x + y$ 

x, y, z: vector

 $\alpha$ : scalar

# Task 11: CuPy-based bifurcation map

