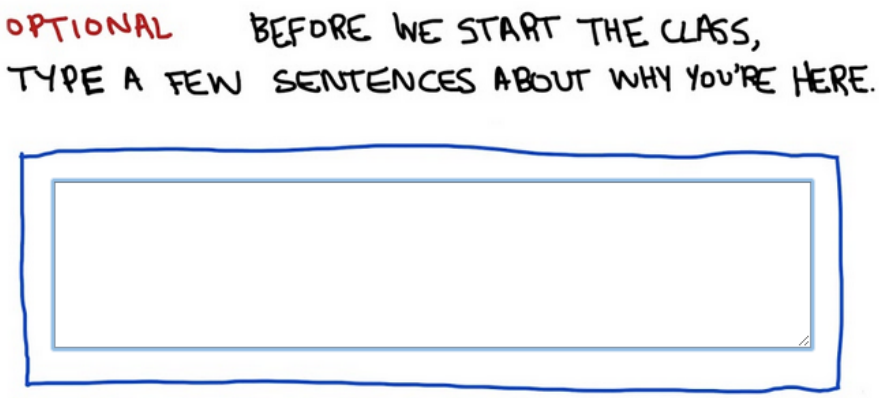
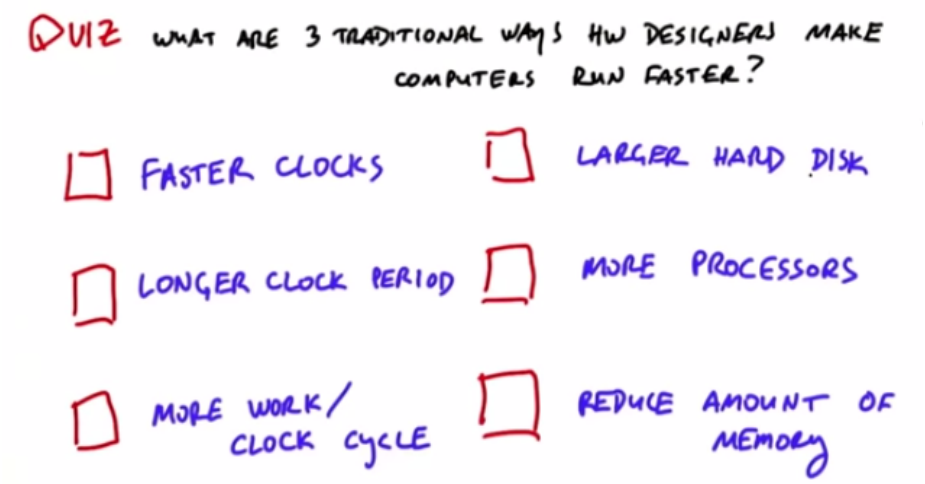
CS344 Introduction to Parallel Programming

Lesson 1: The GPU Programming Model

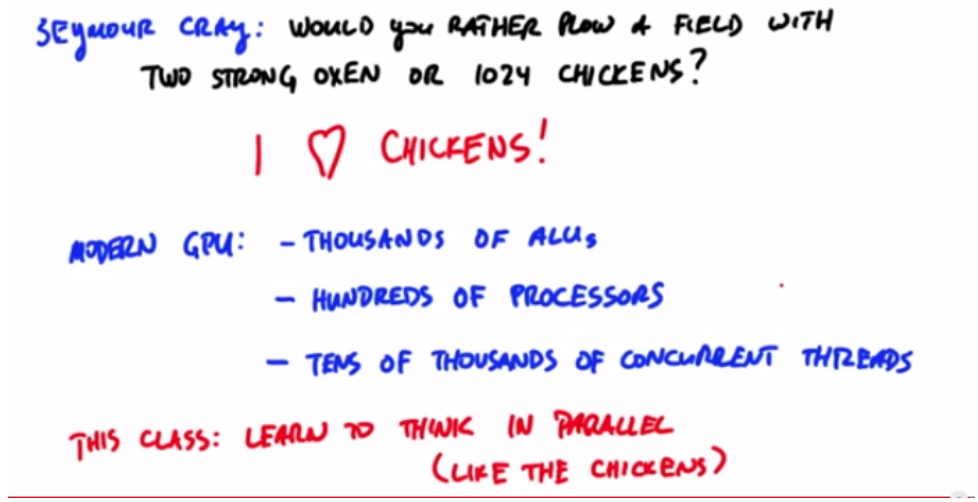
L1-1.3- Why Are You Taking This Class? Quiz 1



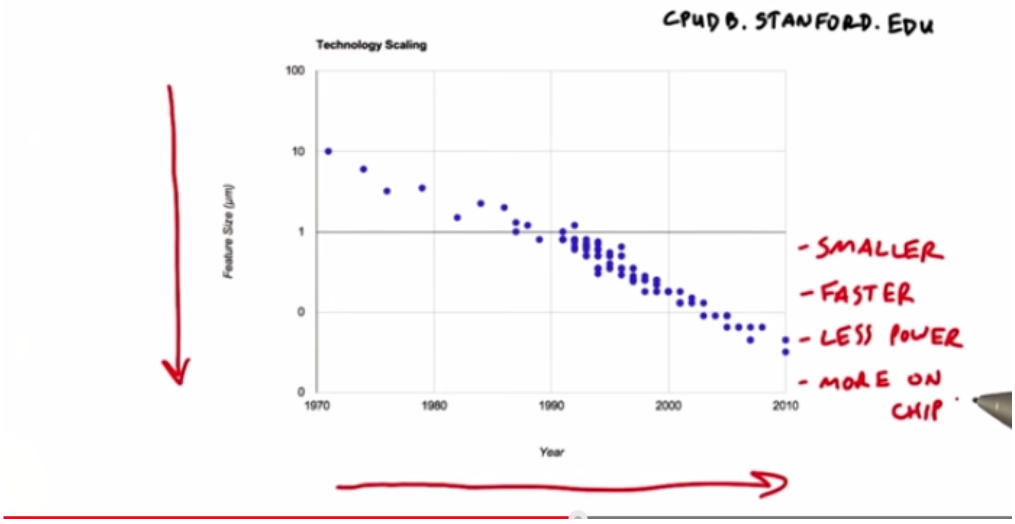
L1-1.5-How To Make Computers Run Faster Quiz 2

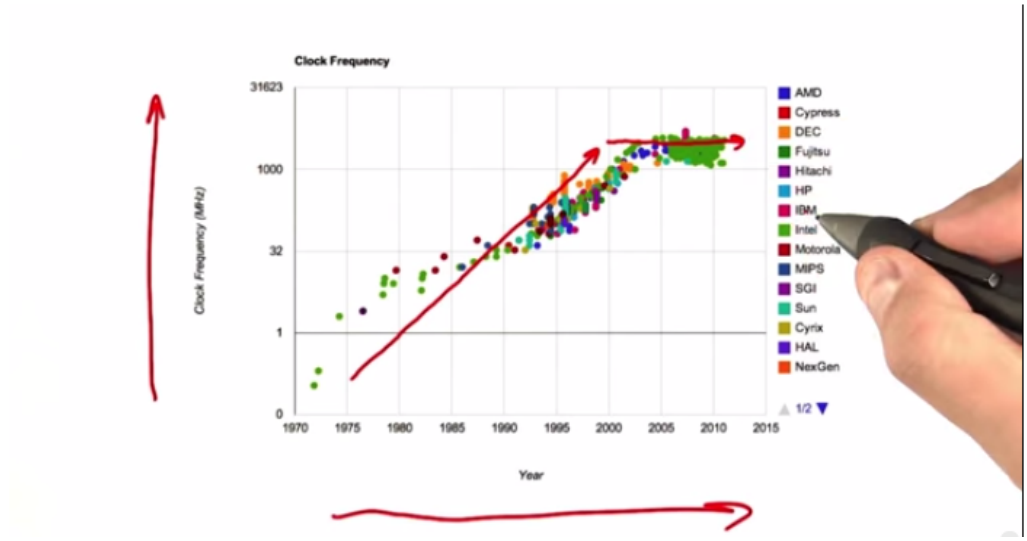


L1-1.6- Chickens or Oxen

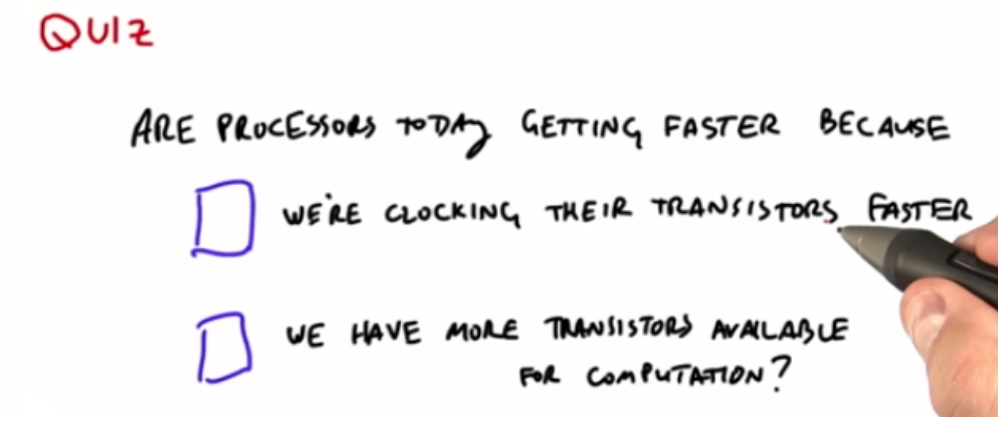


L1-1.7- CPU Speed Remaining Flat

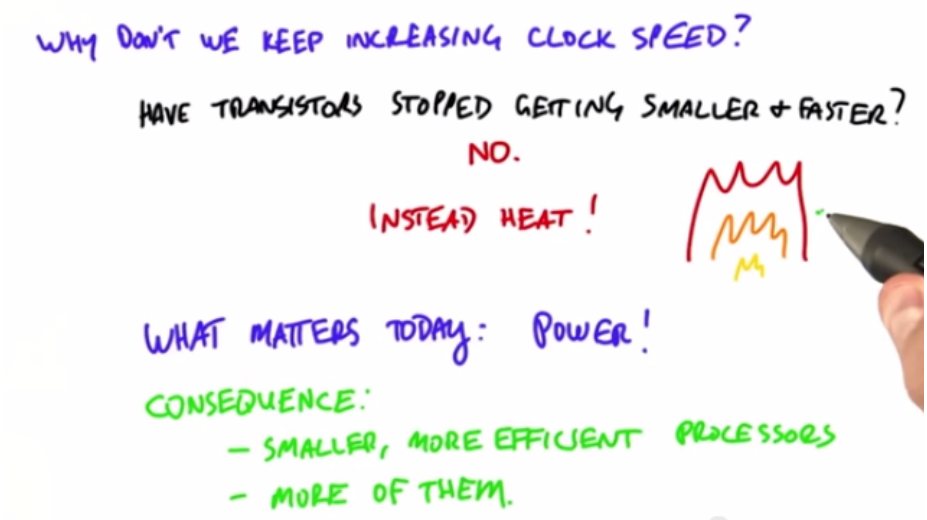




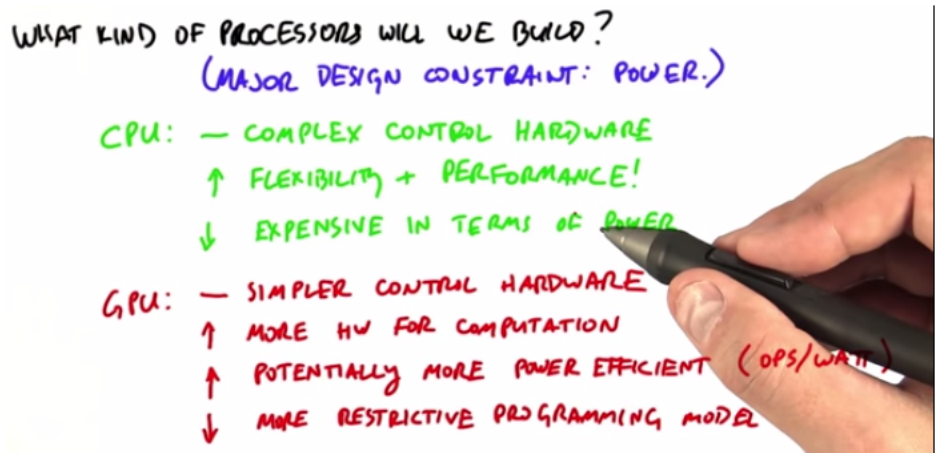
L1-1.8-How Are CPUs Getting Faster? Quiz 3



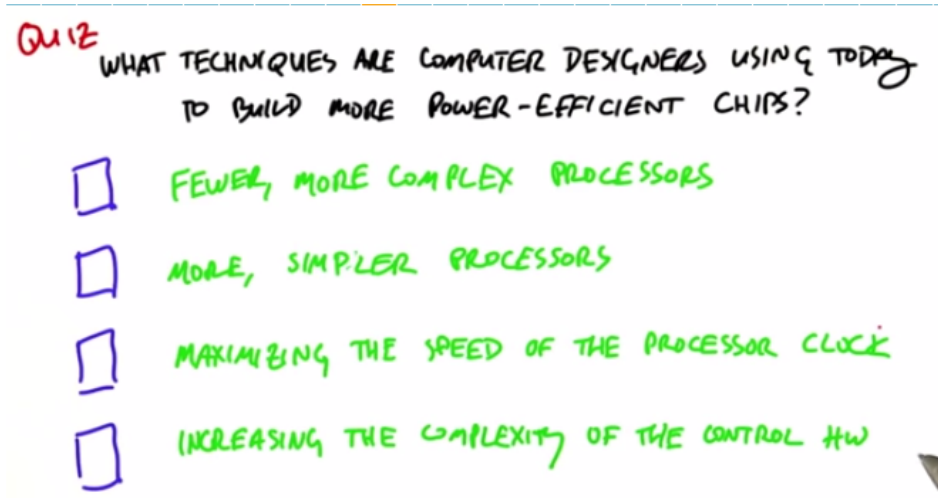
L1-1.9-Why We Cannot Keep Increasing CPU Speed?



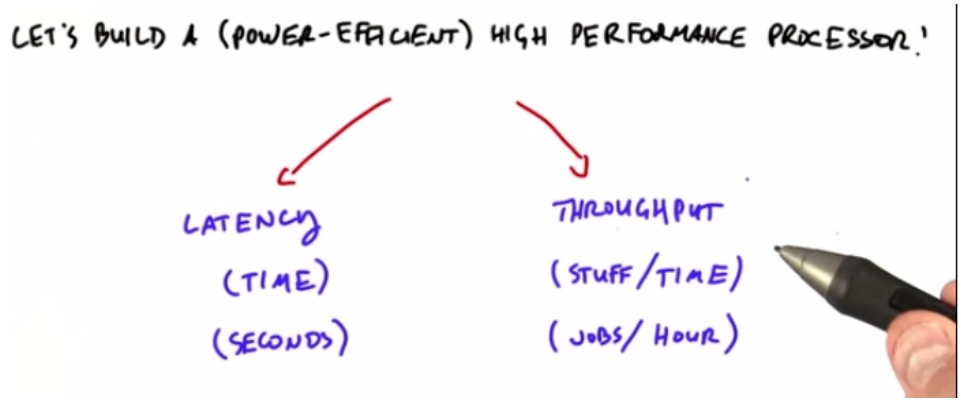
L1-1.10-What Kind of Processors Are We Building?



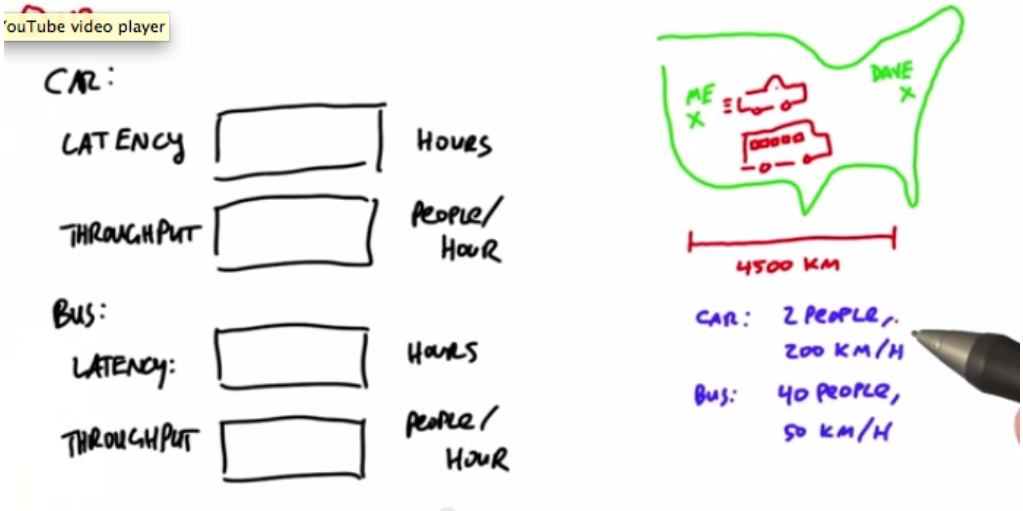
L1-1.11-Techniques To Building Power-efficient Chips Quiz4



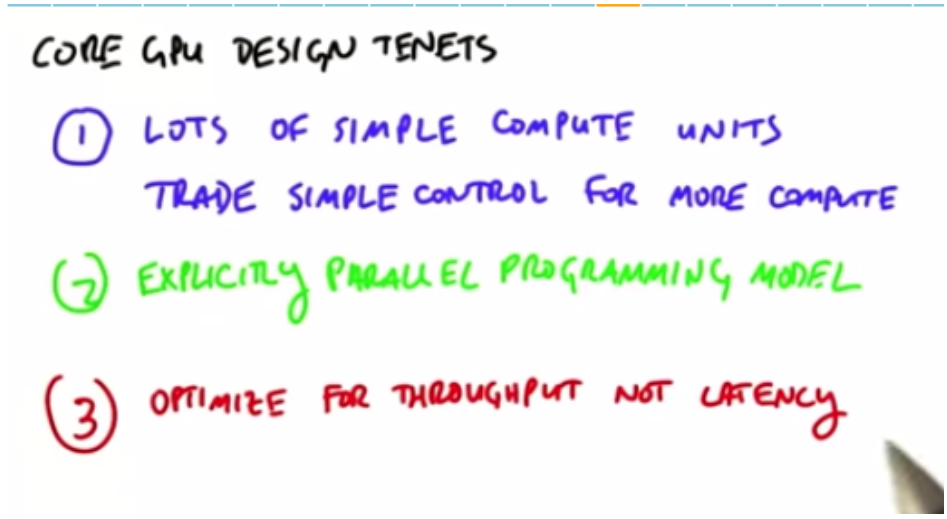
L1-1.12-Building A Power Efficient Processor



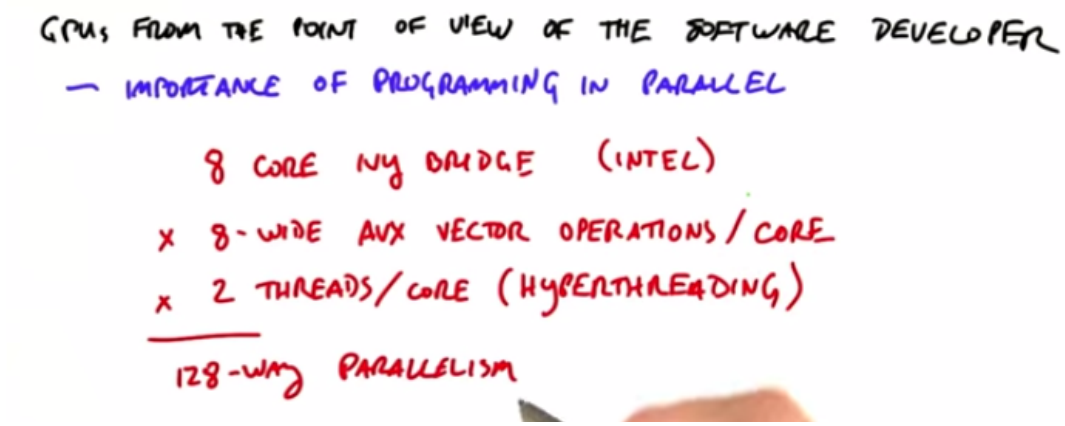
L1-1.13-Latency vs Bandwidth Quiz 4



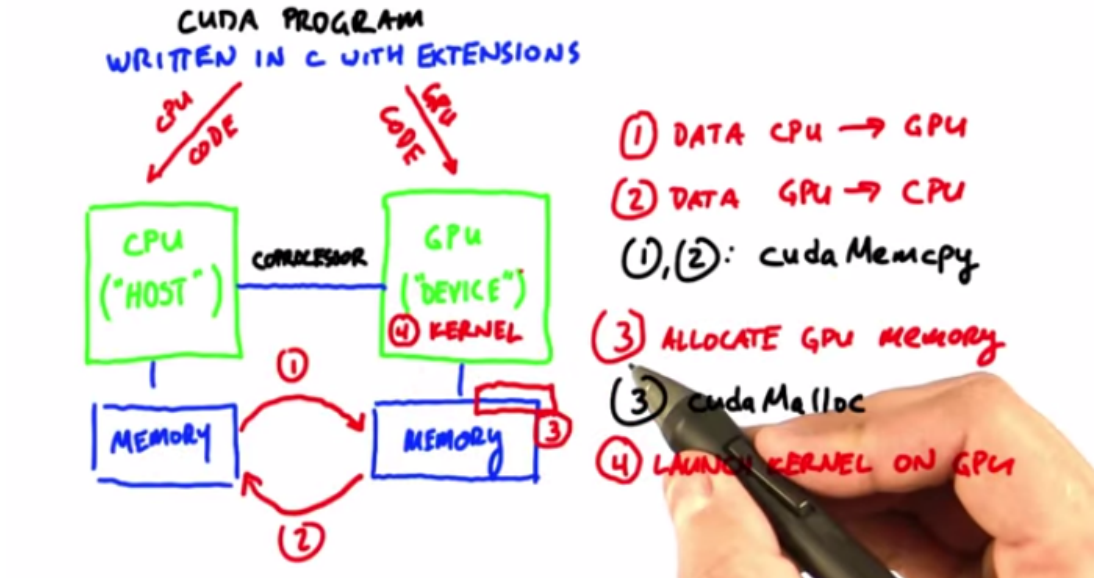
L1-1.14- Core GPU Design Tenets



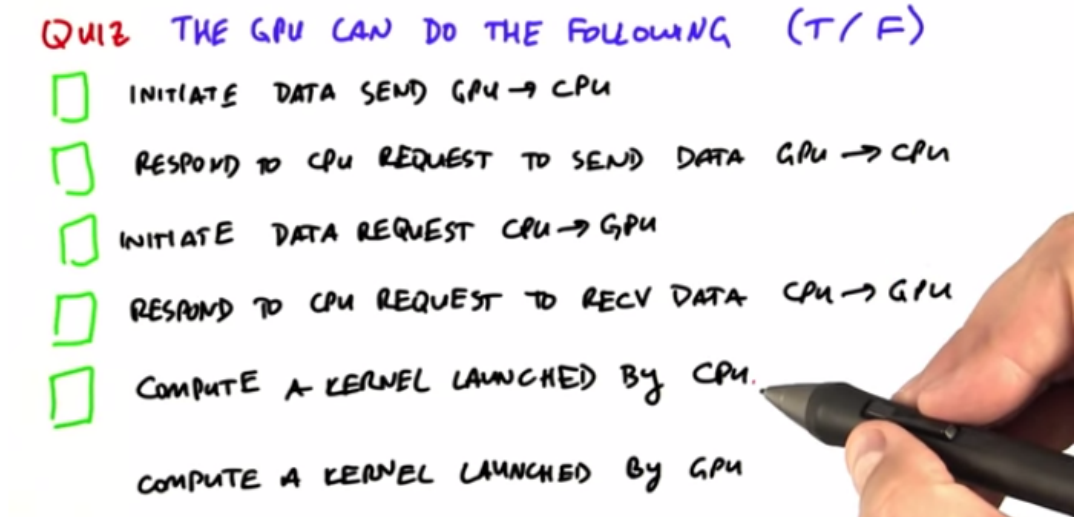
L1-1.15-GPU from the Point of View of the Developer



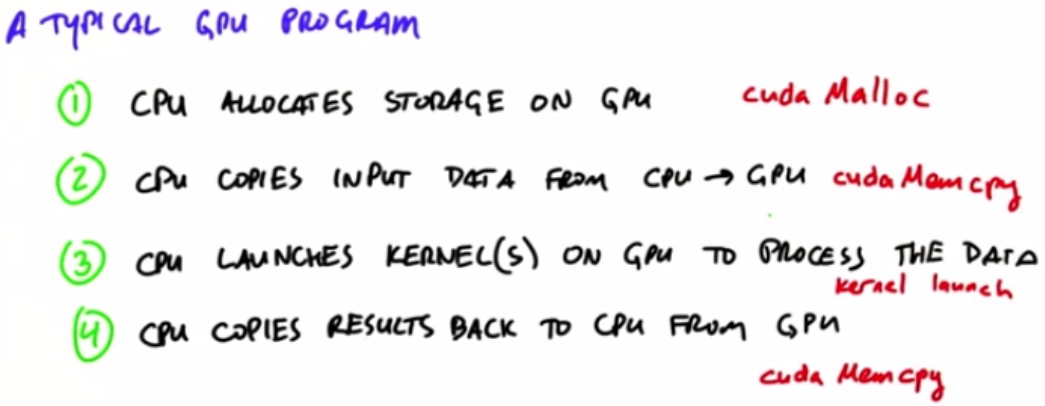
L1-1.16-CUDA Program Diagram



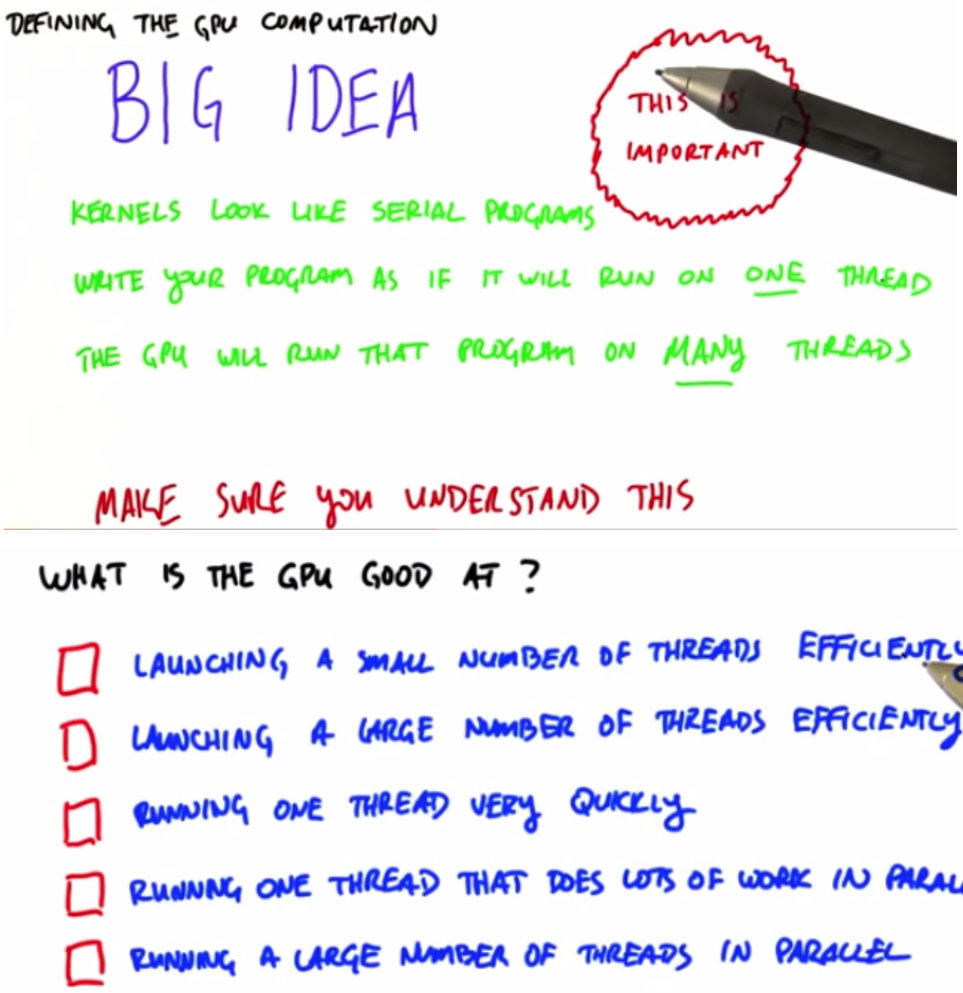
L1-1.17-What Can GPU Do in CUDA Quiz 6



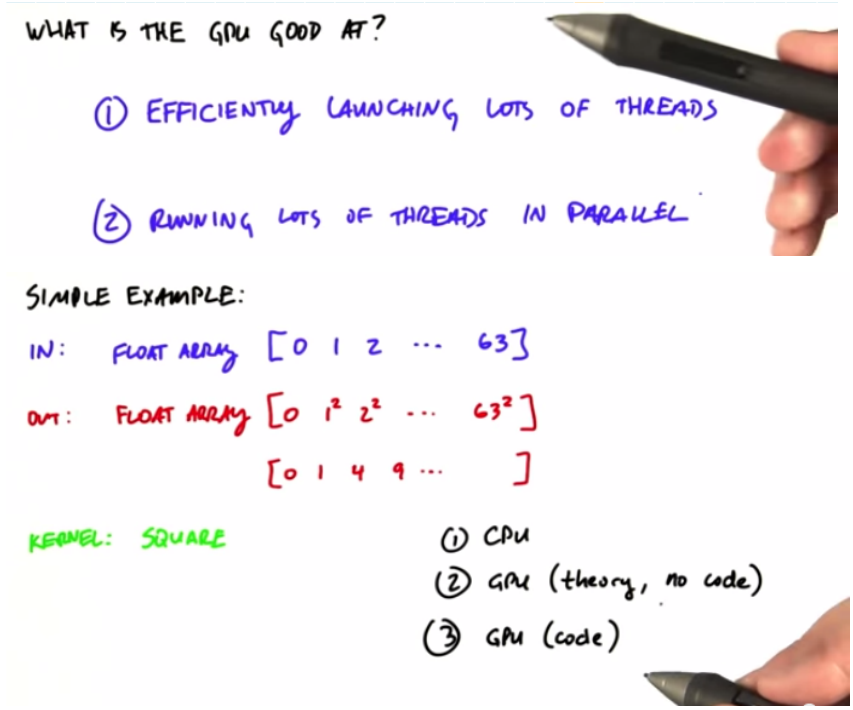
L1-1.18-A CUDA Program



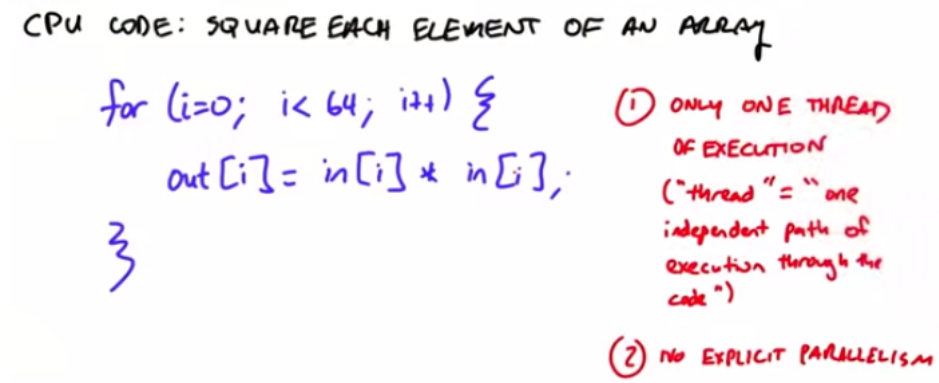
L1-1.19-Defining the GPU Computation Quiz 7



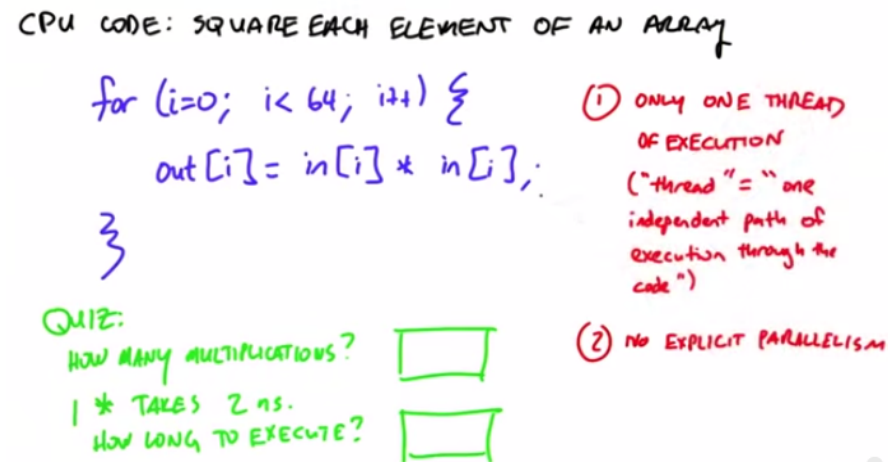
L1-1.20-What the GPU is Good at



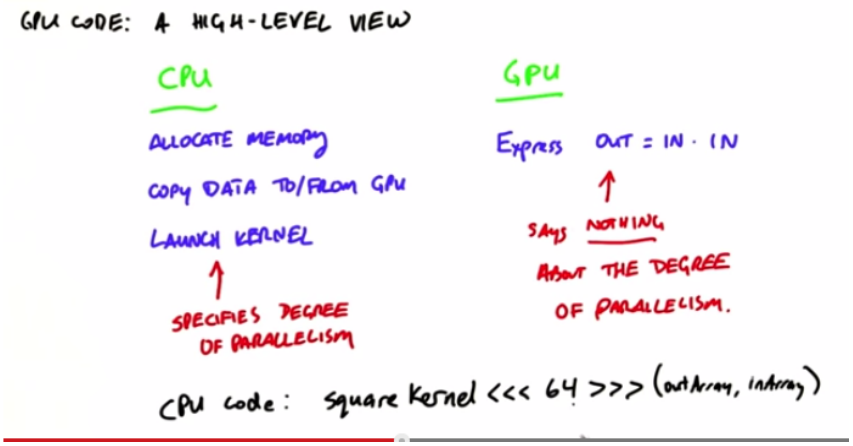
L1-1.21-Squaring A Number on the CPU



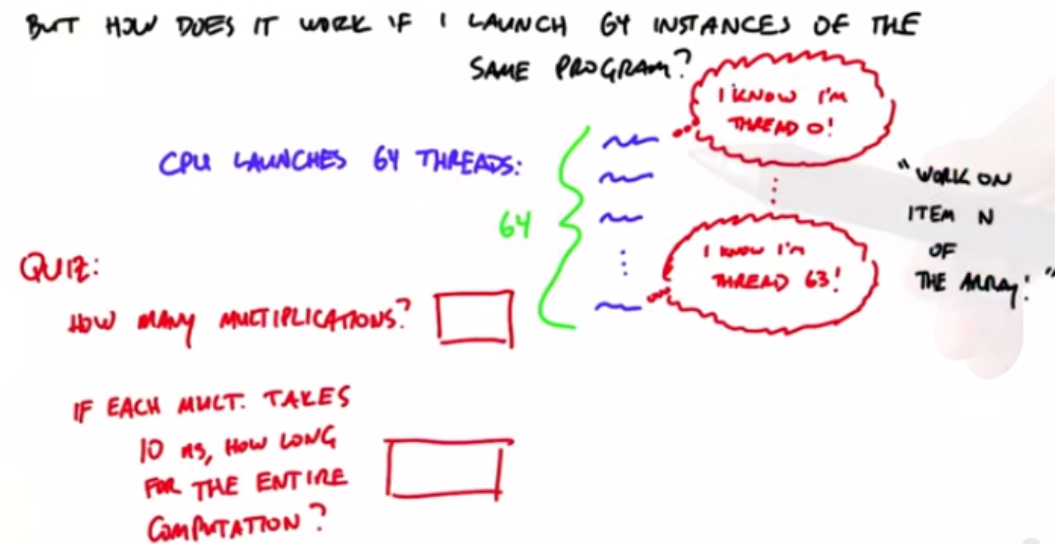
L1-1.22-Calculation Time on the CPU Quiz 8



L1-1.23-GPU Code A High Level View



L1-1.24-Calculation TIme on The GPU

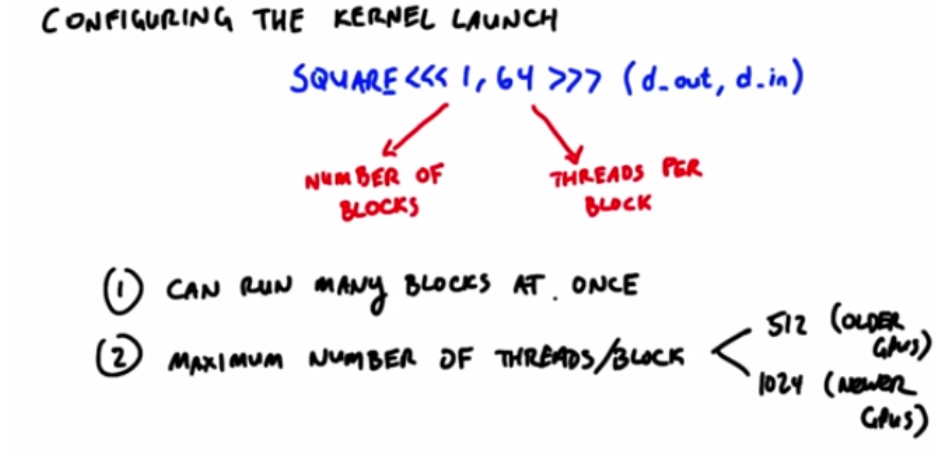


L1-1.25-Squaring Numbers Using CUDA Part 1
L1-1.26-Squaring Numbers Using CUDA Part 2
L1-1.27-Copy to Host or Copy to Device Quiz10
L1-1.28-Squaring Numbers Using CUDA Part 3
L1-1.29-Squaring Numbers Using CUDA Part 4

```cpp
#include <stdio.h>

__global__ void square(float * d_out, float * d_in) {
  int idx = threadIdx.x;
  float f = d_in[idx];
  d_out[idx] = f * f;
}

int main(int argc, char ** argv) {
  const int ARRAY_SIZE = 64;
  const int ARRAY_BYTES = ARRAY_SIZE * sizeof(float);

  // generate the input array on the host
  float h_in[ARRAY_SIZE];
  for (int i = 0; i < ARRAY_SIZE; i++) {
    h_in[i] = float(i);
  }
  float h_out[ARRAY_SIZE];

  // declare GPU memory pointers
  float * d_in;
  float * d_out;

  // allocate GPU memory
  cudaMalloc((void **) &d_in, ARRAY_BYTES);
  cudaMalloc((void **) &d_out, ARRAY_BYTES);

  // transfer the array to the GPU
  cudaMemcpy(d_in, h_in, ARRAY_BYTES, cudaMemcpy???);

  // launch the kernel
  square<<<1, ARRAY_SIZE>>>(d_out, d_in);

  // copy back the result array to the CPU
  cudaMemcpy(h_out, d_out, ARRAY_BYTES, cudaMemcpy???);

  // print out the resulting array
  for (int i = 0; i < ARRAY_SIZE; i++) {
    printf("%f", h_out[i]);
    printf(((i % 4) != 3) ? "\t" : "\n");
  }

  // free GPU memory allocation
  cudaFree(d_in);
  cudaFree(d_out);

  return 0;
}
```

L1-1.30-Cubing Numbers Using CUDA Quiz11

L1-1.31-Configuring the Kernel Launch Parameters Part 1



CONFIGURING THE KERNEL LAUNCH

```
SQUARE <<< 1, 64 >>> (d_out, d_in)
```

NUMBER OF BLOCKS

THREADS PER BLOCK

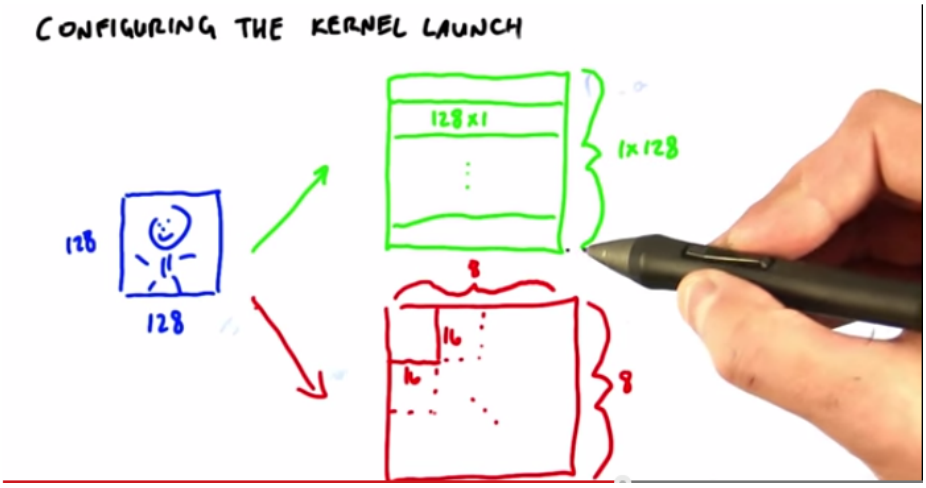
128 THREADS? `SQUARE <<< 1, 128 >>> ( ... )`

1280 THREADS? `SQUARE <<< 10, 128 >>> ( ... )`

`SQUARE <<< 5, 256 >>> ( ... )`

~~`SQUARE <<< 1, 1280 >>> ( ... )`~~

CONFIGURING THE KERNEL LAUNCH



CONFIGURING THE KERNEL LAUNCH

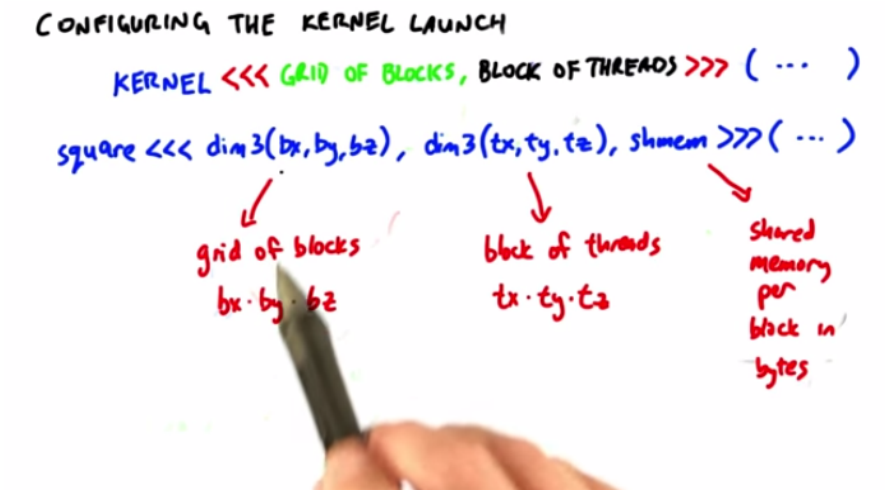
```
KERNEL <<< GRID OF BLOCKS, BLOCK OF THREADS >>> ( ... )
```

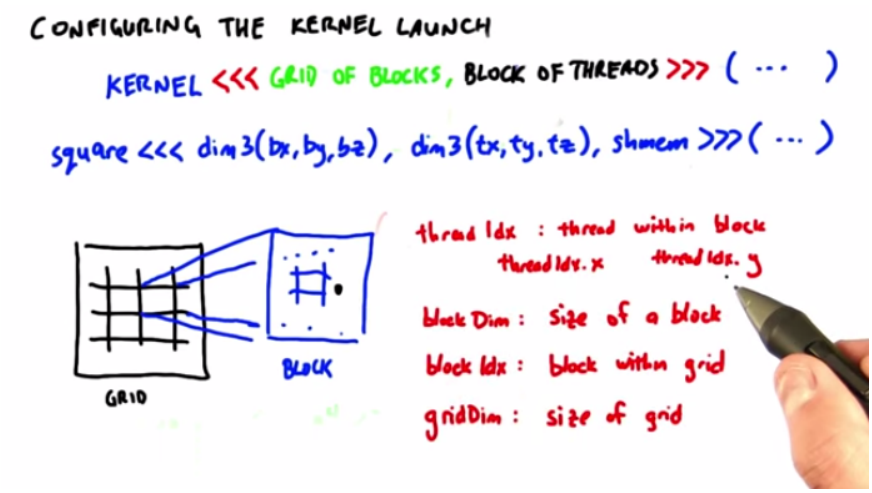
1,2, or 3D

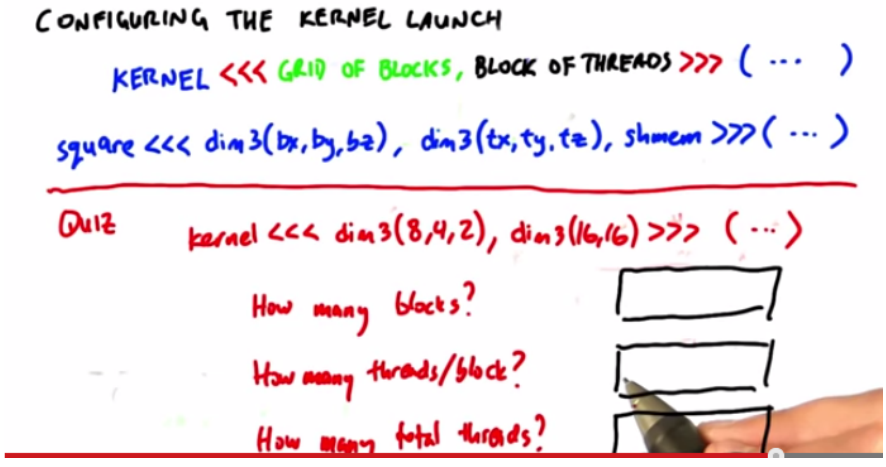
1,2, or 3D

```
dim3(x,y,z)

dim3(w,1,1) == dim3(w) == w

square <<< 1, 64 >>> == square <<< dim3(1,1,1),
                                   dim3(64,1,1) >>>
```

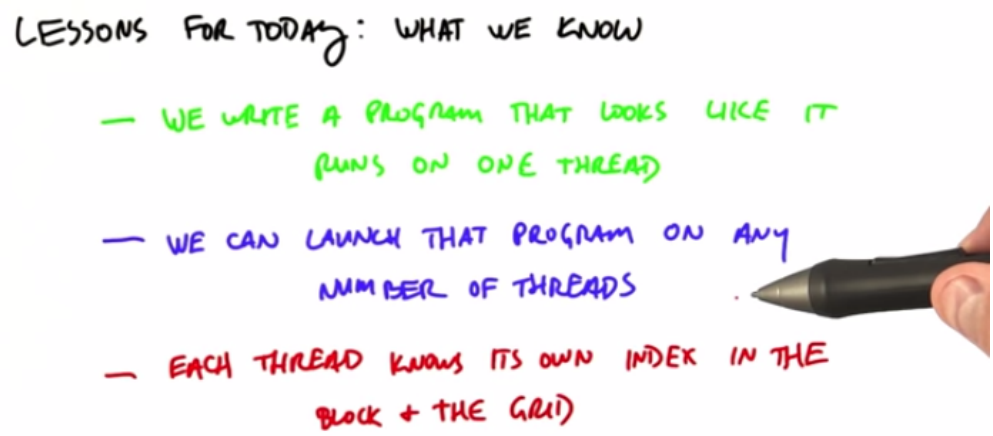
L1-1.32-Configuring the Kernel Launch Parameters Part 2



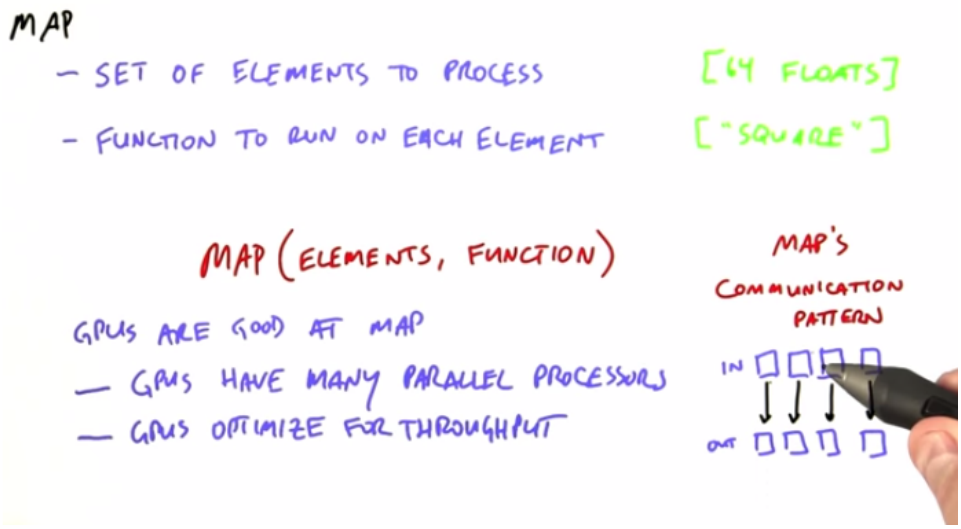


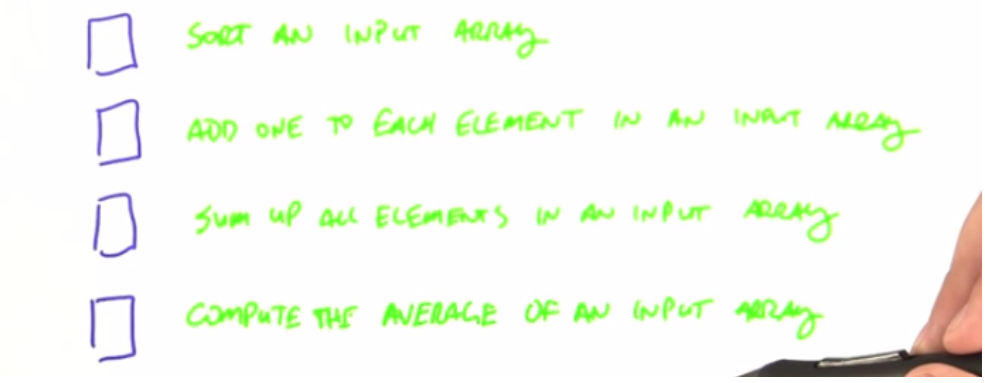


L1-1.33-What We Know So Far

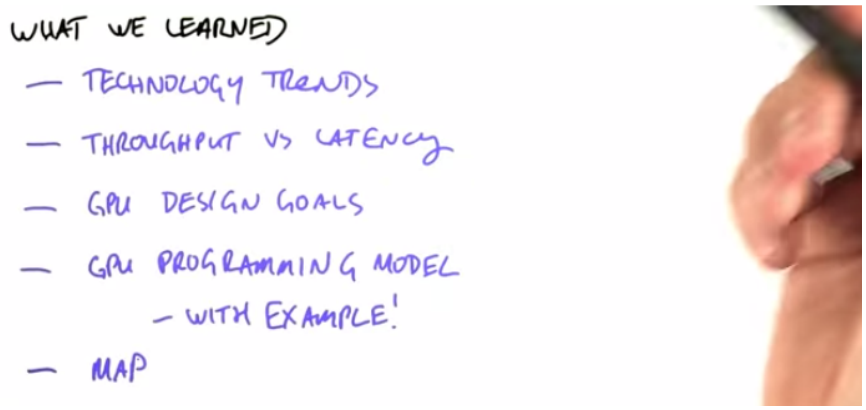


L1-1.34-Map





L1-1.35-Summary of Lesson 1



L1-1.37-Problem Set #1