CUDA Programming

Agenda

- **Computation**
- Memory
- Synchronization
- Functions
- Support
- Topics

Hello World.

```
#include <stdio.h>

int main() {

    printf("Hello World.\n");

    return 0;

}
```

Compile: nvcc hello.cu
Run: a.out

GPU Hello World.

Kernel

Kernel Launch →

```
#include <stdio.h>

#include <cuda.h>

__global__ void dkernel() {

    printf(“Hello World.\n”);

}

int main() {

    dkernel<<<1, 1>>>();

    return 0;

}
```

Compile: nvcc hello.cu
Run: ./a.out
– No output. --

GPU Hello World.

```
#include <stdio.h>
#include <cuda.h>
__global__ void dkernel() {
    printf("Hello World.\n");
}
int main() {
    dkernel<<<1, 1>>>();
    cudaDeviceSynchronize();
    return 0;
}
```

```
Compile: nvcc hello.cu
Run: ./a.out
Hello World.
```

Takeaway

CPU function
and GPU kernel
run asynchronously.

GPU Hello World.

```
#include <stdio.h>
#include <cuda.h>
__global__ void dkernel() {
    printf(“Hello World.\n”);
}
int main() {
    dkernel<<<1, 1>>>();
    dkernel<<<1, 1>>>();
    dkernel<<<1, 1>>>();
    cudaDeviceSynchronize();
    printf(“on CPU\n”);
    return 0;
}
```

Takeaway

Kernels (by default) are executed one after another.

CPU launches them and moves ahead.

CPU waits at CDS.

GPU Hello World.

```
__global__ void dkernel1() {
    printf(“Hello World 1.\n”);
}
__global__ void dkernel2() {
    printf(“Hello World 2.\n”);
}
int main() {
    dkernel1<<<1, 1>>>();
    dkernel2<<<1, 1>>>();
    cudaDeviceSynchronize();
    printf(“on CPU\n”);
    return 0;
}
```

GPU Hello World.

```
__global__ void dkernel1() {
    printf(“Hello World 1.\n”);
}
__global__ void dkernel2() {
    printf(“Hello World 2.\n”);
}
int main() {
    dkernel1<<<1, 1>>>();
    dkernel2<<<1, 1>>>();
    printf(“on CPU\n”);
    cudaDeviceSynchronize();
    return 0;
}
```

GPU Hello World.

```
__global__ void dkernel() {
    printf(“Hello World.\n”);
}
int main() {
    dkernel<<<1, 1>>>();
    printf(“CPU one\n”);
    dkernel<<<1, 1>>>();
    printf(“CPU two\n”);
    dkernel<<<1, 1>>>();
    printf(“CPU three\n”);
    cudaDeviceSynchronize();
    printf(“on CPU\n”);
    return 0;
}
```

Identify which printfs can execute in parallel.

Homework

- Find out where *nvcc* is.
- Find out the CUDA version.
- Find out where *deviceQuery* is.

GPU Hello World in Parallel.

```
#include <stdio.h>
#include <cuda.h>
__global__ void dkernel() {
    printf("Hello World.\n");
}
int main() {
    dkernel<<<1, 32>>>();
    cudaDeviceSynchronize();
    return 0;
}
```

```
Compile: nvcc hello.cu
Run: ./a.out
Hello World.
Hello World.
...
```

32 times

Parallel Programming Concepts

- Process: a.out, notepad, chrome
- Thread: light-weight process
- Operating system: Windows, Android, Linux
  - OS is a software, but it manages the hardware.
- Hardware
  - Cache, Main memory
  - Cores
- Core
  - Threads run on cores.
  - A thread may jump from one core to another.

Classwork

Can this be made parallel?

- Write a CUDA code corresponding to the following sequential C code.

```
#include <stdio.h>
#define N 100
int main() {
    int i;
    for (i = 0; i < N; ++i)
        printf("%d\n", i * i);
    return 0;
}
```

```
#include <cuda.h>
#define N 100
__global__ void fun() {
    for (int i = 0; i < N; ++i)
        printf("%d\n", i * i);
}
int main() {
        fun<<<1, 1>>>();
        cudaDeviceSynchronize();
        return 0;
}
```

Classwork

- Write a CUDA code corresponding to the following sequential C code.

```
#include <stdio.h>
#define N 100
int main() {
    int i;
    for (i = 0; i < N; ++i)
        printf("%d\n", i * i);
    return 0;
}
```

```
#include <cuda.h>
#define N 100
__global__ void fun() {
        printf("%d\n", threadIdx.x *
                       threadIdx.x);
}
int main() {
        fun<<<1, N>>>();
        cudaDeviceSynchronize();
        return 0;
}
```

Note that there is no loop here.

Classwork

- Write a CUDA code corresponding to the following sequential C code.

Problem with this code?

```
#include <stdio.h>
#define N 100
int main() {
    int i;
    for (i = 0; i < N; ++i)
        printf("%d\n", i * i);
    return 0;
}
```

```
#include <cuda.h>
#define N 100
__global__ void fun() {
        printf("%d\n", threadIdx.x *
                       threadIdx.x);
}
int main() {
        fun<<<1, N>>>();
        cudaDeviceSynchronize();
        return 0;
}
```