Parallel Computing Workshop

Ms. Kranti Ingale

Prime Minister's Research Fellows (**PMRF**) Scholar

CSE, IIT Madras

Roadmap

- Introduction to GPU
- CUDA Program Flow and CPU-GPU Communication
- Thread organization (Grids, Blocks, Threads, 1D/2D)
- CUDA Memory Model
- CUDA Functions
- CUDA Thrust

Roadmap

- Introduction to GPU
- CUDA Program Flow and CPU-GPU Communication
- Thread organization (Grids, Blocks, Threads, 1D/2D)
- CUDA Memory Model
- CUDA Functions
- CUDA Thrust

A simple kernel to add two integers

- addition() will execute on the **device**
- addition() will be called from the **host**

```
__global__ void addition(int *num1, int *num2, int *result)
{
        *result = *num1 + *num2;
}
```

A simple kernel to add two integers

- addition() will execute on the **device**
- addition() will be called from the **host**

```
__global__ void addition(int *num1, int *num2, int *result)
{
        *result = *num1 + *num2;
}
```

A simple kernel to add two integers

- addition() will execute on the **device** – num1, num2 and result must point to device memory
- addition() will be called from the **host**

```
__global__ void addition(int *num1, int *num2, int *result)
{
        *result = *num1 + *num2;
}
```

A simple kernel to add two integers

- addition() will execute on the **device** – num1, num2 and result must point to device memory
- addition() will be called from the **host**

```
__global__ void addition(int *num1, int *num2, int *result)
{
        *result = *num1 + *num2;
}
```

We need to allocate memory on the GPU

CUDA API for handling device memory

- cudaMalloc()
- cudaFree()
- cudaMemcpy()

CUDA API for handling device memory

- cudaMalloc()
- cudaFree()
- cudaMemcpy()

- Similar to the C equivalents malloc(), free(), memcpy()

cudaMalloc()

Syntax :

cudaError_t **cudaMalloc** (void** devPtr, size_t size) {}

cudaMalloc()

Syntax :

cudaError_t **cudaMalloc** (void** devPtr, size_t size) {}

- Parameters :

  devPtr : pointer to allocated device memory

  size : requested allocated size in bytes

cudaMalloc()

Syntax :

cudaError_t **cudaMalloc** (void** devPtr, size_t size) {}

- Parameters :

  devPtr : pointer to allocated device memory

  size : requested allocated size in bytes

- Returns : cudaSuccess , cudaErrorMemoryAllocation

cudaFree()

Syntax :

cudaError_t **cudaFree** (void* devPtr) {}

cudaFree()

Syntax :

cudaError_t **cudaFree** (void* devPtr) {}

- Parameters :

devPtr : deallocates device memory pointed by devPtr

cudaFree()

Syntax :

cudaError_t **cudaFree** (void* devPtr) {}

- Parameters :

  devPtr : deallocates device memory pointed by devPtr

- Returns : cudaSuccess , cudaErrorInvalidValue

cudaMemcpy()

Syntax :

```
cudaError_t  cudaMemcpy (void* dst, const void * src ,
                         size_t size, cudaMemcpyType  kind ) {}
```

cudaMemcpy()

Syntax :

cudaError_t **cudaMemcpy** (void* dst, const void * src ,
size_t size, cudaMemcpyType kind) {}

- Parameters :

dst, src : destination & Source memory address

Kind : type of transfer

cudaMemcpy()

Syntax :

cudaError_t **cudaMemcpy** (void* dst, const void * src ,
size_t size, cudaMemcpyType kind) {}

- Parameters :

  dst, src : Destination & Source memory address

  Kind : Type of transfer

- Returns : cudaSuccess , cudaErrorInvalidValue , cudaErrorInvalidMemcpyDirection,

Let’s back to our problem of addition

```
main() {
        int a=10,b=20,c;
        int *d_a,*d_b,*d_c;

        cudaMalloc(&d_a,sizeof(int));
        cudaMalloc(&d_b,sizeof(int));
        cudaMalloc(&d_c,sizeof(int));
```

Let’s back to our problem of addition

```
main() {
        int a=10,b=20,c;          ──────►  Host copies of a, b, c
        int *d_a,*d_b,*d_c;       ──────►  Device copies of a, b, c

        cudaMalloc(&d_a,sizeof(int));
        cudaMalloc(&d_b,sizeof(int));
        cudaMalloc(&d_c,sizeof(int));
```

Let’s back to our problem of addition

```
main() {
        int a=10,b=20,c;
        int *d_a,*d_b,*d_c;

        cudaMalloc(&d_a,sizeof(int));
        cudaMalloc(&d_b,sizeof(int));
        cudaMalloc(&d_c,sizeof(int));
```

int a=10,b=20,c; → Host copies of a, b, c

int *d_a,*d_b,*d_c; → Device copies of a, b, c

cudaMalloc(&d_a,sizeof(int));
cudaMalloc(&d_b,sizeof(int));
cudaMalloc(&d_c,sizeof(int));
} Allocate space for device copies of a, b, c

Let’s back to our problem of addition

```
    cudaMemcpy(d_a, &a, sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, &b, sizeof(int), cudaMemcpyHostToDevice);

    addition<<<1,1>>>(d_a,d_b,d_c);

    cudaMemcpy(&c, d_c, sizeof(int), cudaMemcpyDeviceToHost);
    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);

}// end main
```

Let’s back to our problem of addition

```
cudaMemcpy(d_a, &a, sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &b, sizeof(int), cudaMemcpyHostToDevice);

addition<<<1,1>>>(d_a,d_b,d_c);

cudaMemcpy(&c, d_c, sizeof(int), cudaMemcpyDeviceToHost);
cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);

}// end main
```

Copy inputs to device

Let’s back to our problem of addition

```
cudaMemcpy(d_a, &a, sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &b, sizeof(int), cudaMemcpyHostToDevice);

addition<<<1,1>>>(d_a,d_b,d_c);

cudaMemcpy(&c, d_c, sizeof(int), cudaMemcpyDeviceToHost);
cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);

}// end main
```

Copy inputs to device

kernel Launch

Let’s back to our problem of addition

```
cudaMemcpy(d_a, &a, sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &b, sizeof(int), cudaMemcpyHostToDevice);

addition<<<1,1>>>(d_a,d_b,d_c);

cudaMemcpy(&c, d_c, sizeof(int), cudaMemcpyDeviceToHost);
cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);

}// end main
```

Copy inputs to device

kernel Launch

Copy result to Host

Let’s back to our problem of addition

```
cudaMemcpy(d_a, &a, sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &b, sizeof(int), cudaMemcpyHostToDevice);

addition<<<1,1>>>(d_a,d_b,d_c);

cudaMemcpy(&c, d_c, sizeof(int), cudaMemcpyDeviceToHost);
cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);

}// end main
```

Copy inputs to device

kernel Launch

Copy result to Host

Clean-up

Write a CUDA code corresponding to the following sequential C code

```
#include <stdio.h>
#define N 100
int main() {
        int a[N], i;
        for (i = 0; i < N; ++i)
        a[i] = i * i *i ;
        return 0;
}
```

Write a CUDA code corresponding to the following sequential C code

```
#include <stdio.h>
#define N 100
int main() {
        int a[N], i;
        for (i = 0; i < N; ++i)
        a[i] = i * i *i ;
        return 0;
}
```

```
#define N 100
```

Write a CUDA code corresponding to the following sequential C code

```
#include <stdio.h>
#define N 100
int main() {
        int a[N], i;
        for (i = 0; i < N; ++i)
        a[i] = i * i *i ;
        return 0;
}
```

```
#define N 100
int main() {
int a[N], *d_a, i;
```

Write a CUDA code corresponding to the following sequential C code

```
#include <stdio.h>
#define N 100
int main() {
        int a[N], i;
        for (i = 0; i < N; ++i)
        a[i] = i * i *i ;
        return 0;
}
```

```
#define N 100
int main() {
int a[N], *d_a, i;
cudaMalloc(&d_a, N * sizeof(int));
return 0;
}
```

Allocate space for array on device

Write a CUDA code corresponding to the following sequential C code

```
#include <stdio.h>
#define N 100
int main() {
        int a[N], i;
        for (i = 0; i < N; ++i)
        a[i] = i * i *i ;
        return 0;
}
```

```
#define N 100
int main() {
int a[N], *d_a, i;
cudaMalloc(&d_a, N * sizeof(int));
fun<<<1, N>>>(d_a);
return 0;
}
```


Kernel launch

Write a CUDA code corresponding to the following sequential C code

```
#include <stdio.h>
#define N 100
int main() {
        int a[N], i;
        for (i = 0; i < N; ++i)
        a[i] = i * i *i ;
        return 0;
}
```

```
#define N 100
int main() {
int a[N], *d_a, i;
cudaMalloc(&d_a, N * sizeof(int));
fun<<<1, N>>>(d_a);
cudaMemcpy(a, d_a, N * sizeof(int), cudaMemcpyDeviceToHost );
return 0;
}
```

Write a CUDA code corresponding to the following sequential C code

```
#include <stdio.h>
#define N 100
int main() {
        int a[N], i;
        for (i = 0; i < N; ++i)
        a[i] = i * i *i ;
        return 0;
}
```

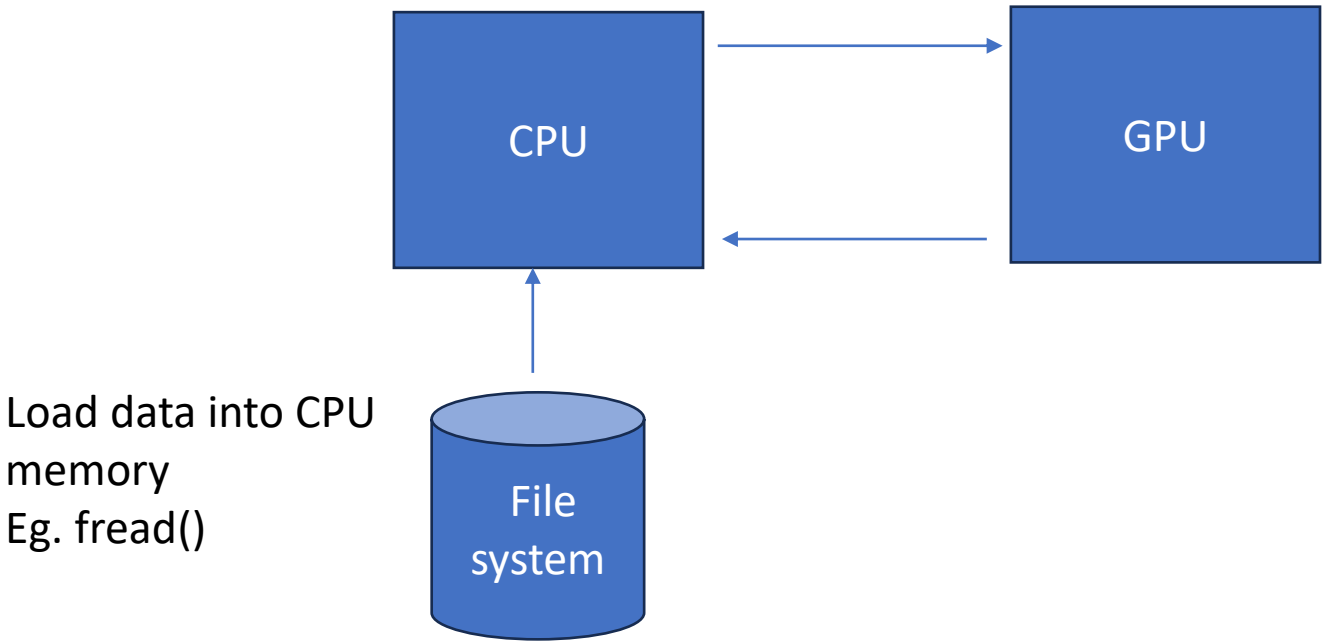
```
#define N 100
int main() {
int a[N], *d_a, i;
cudaMalloc(&d_a, N * sizeof(int));
fun<<<1, N>>>(d_a);
cudaMemcpy(a, da, N * sizeof(int), cudaMemcpyDeviceToHost );
return 0;
}
```

Write a CUDA code corresponding to the following sequential C code

```
#include <stdio.h>
#define N 100
int main() {
        int a[N], i;
        for (i = 0; i < N; ++i)
        a[i] = i * i *i ;
        return 0;
}
```

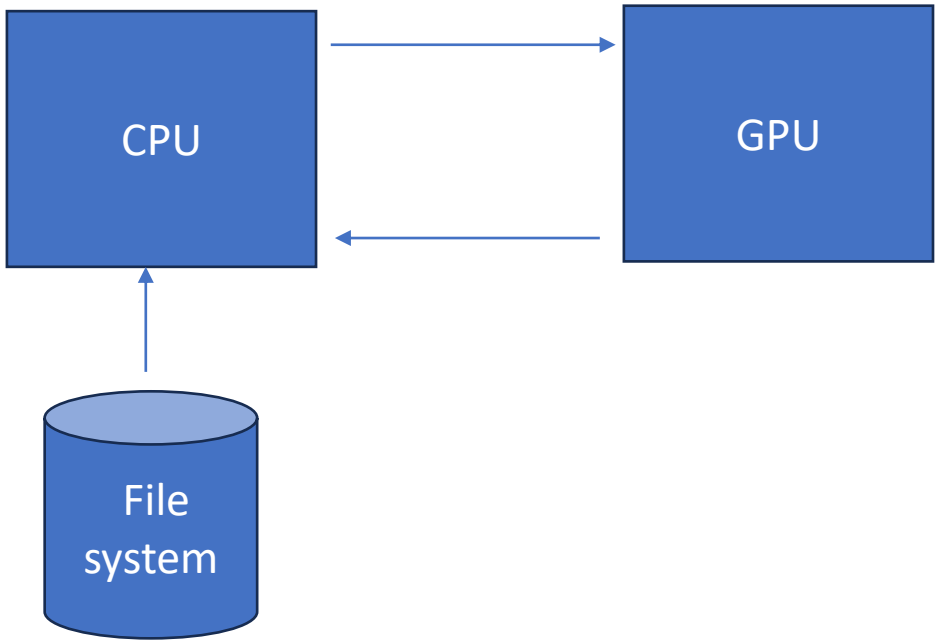
```
__global__ void fun(int *a) {
a[threadIdx.x] = threadIdx.x * threadIdx.x * threadIdx.x;
}
```

Typical CUDA Program flow



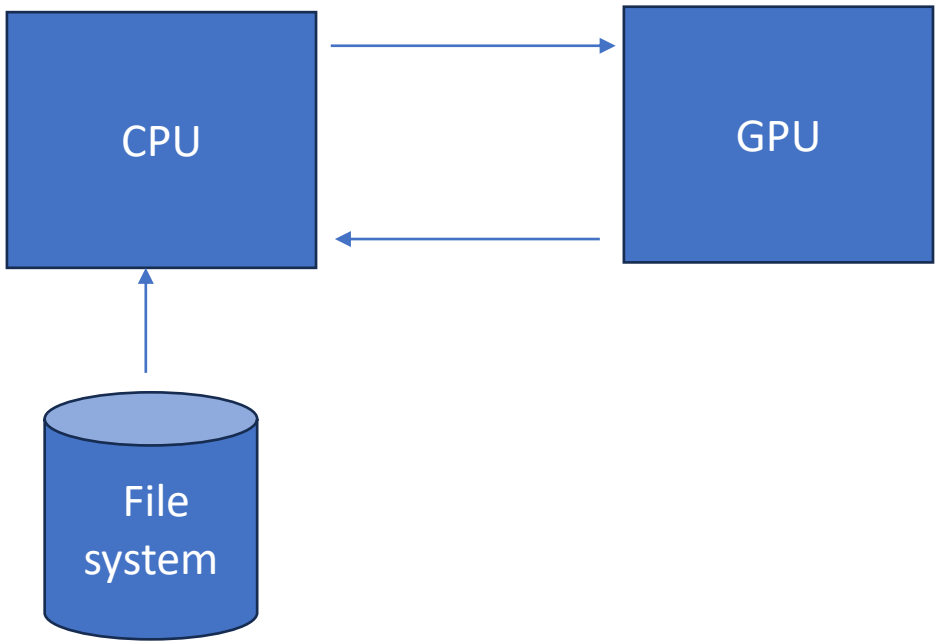
Typical CUDA Program flow

Copy data from CPU to GPU memory

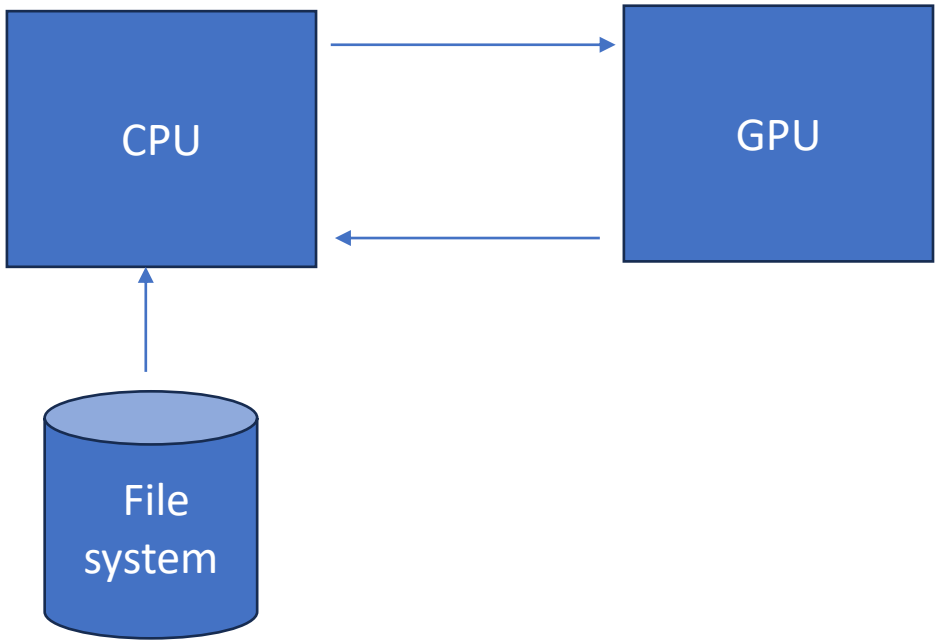


Typical CUDA Program flow

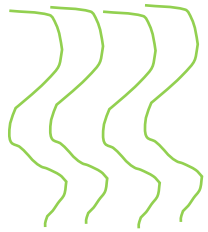
Copy data from CPU to GPU memory
cudaMemcpy(.., cudaMemcpyHostToDevice)



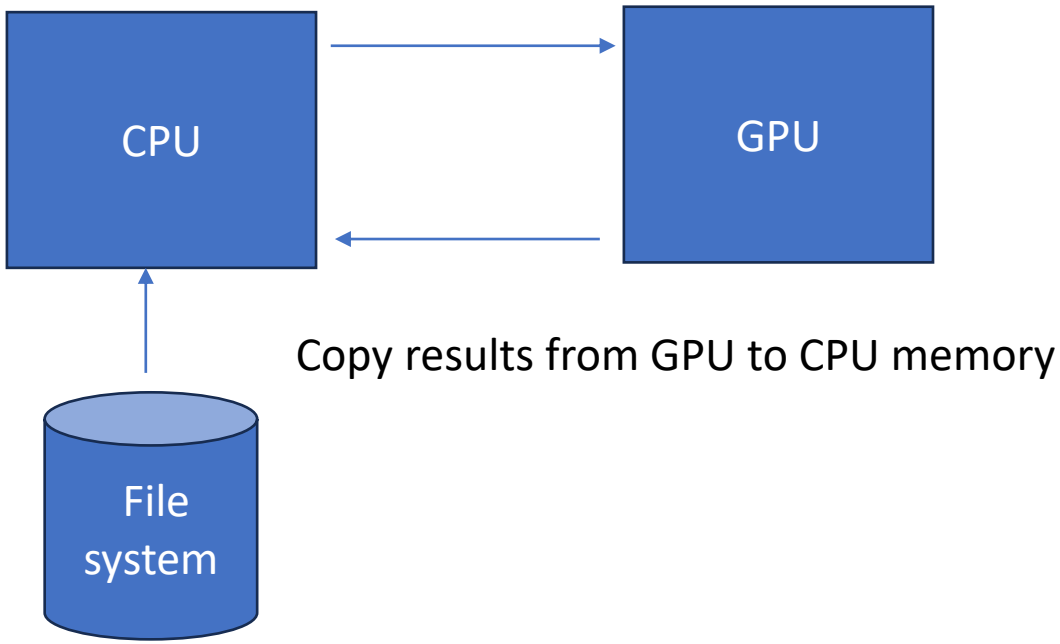
Typical CUDA Program flow



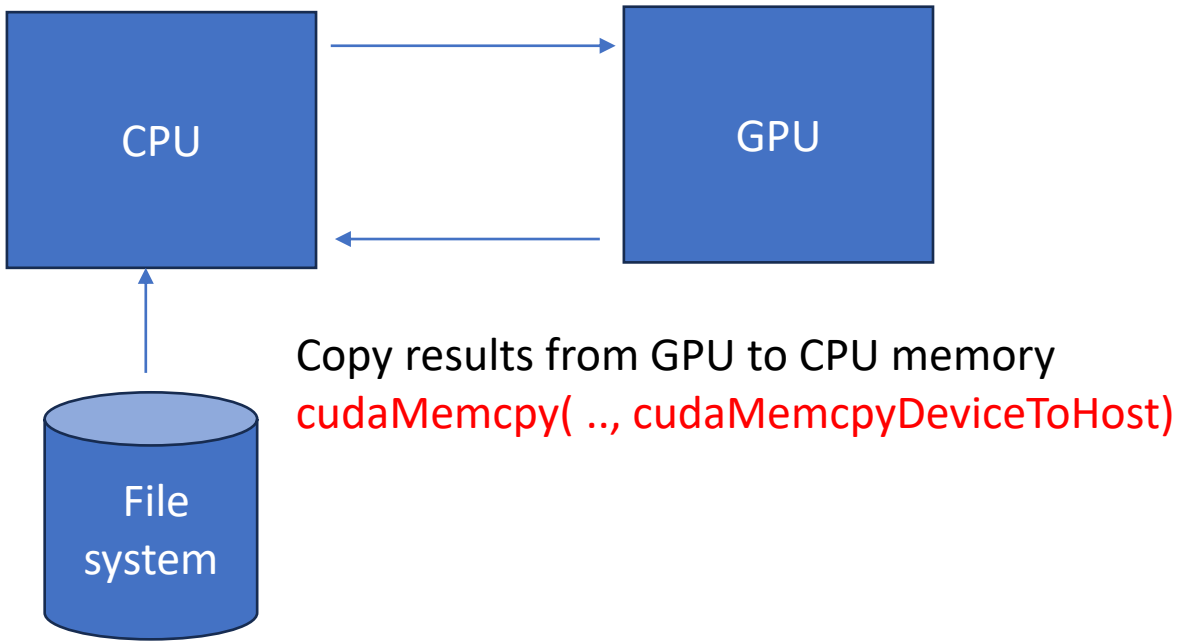
Execute GPU kernel



Typical CUDA Program flow

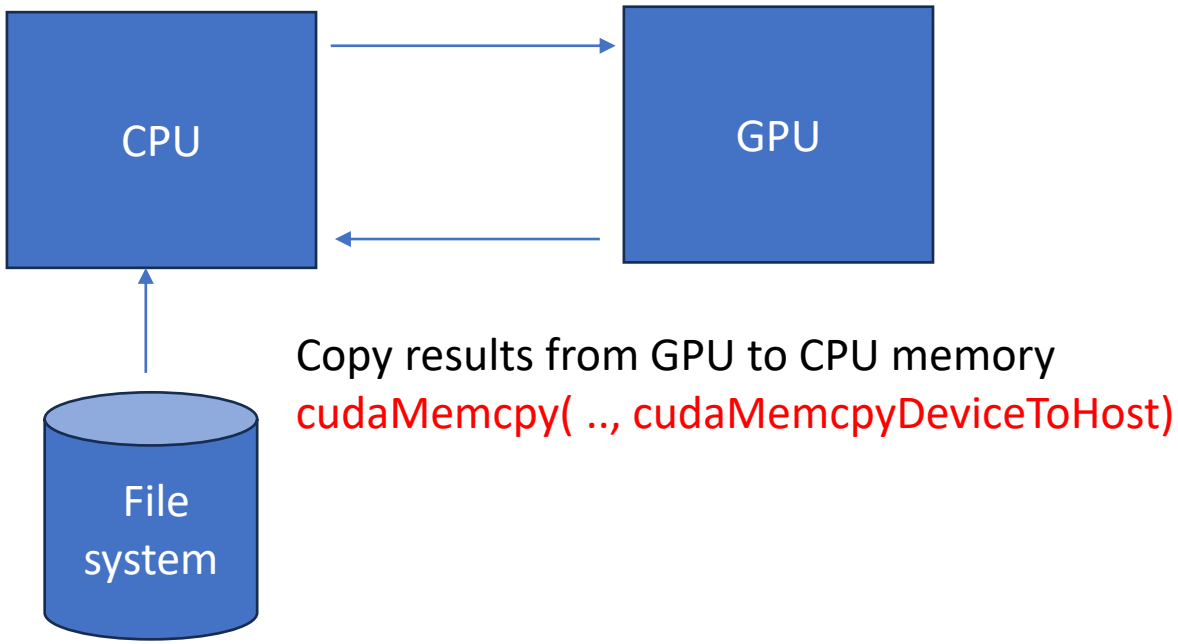


Typical CUDA Program flow



CPU-GPU COMMUNICATION

Copy data from CPU to GPU memory
cudaMemcpy(.., cudaMemcpyHostToDevice)



Copy results from GPU to CPU memory
cudaMemcpy(.., cudaMemcpyDeviceToHost)

Conclusion of Today's class

- We allocate memory on device with cudaMalloc().
- cudaFree() use to free the device memory.
- cudaMemcpy() is use to provide cpu-gpu communication.
- cudaMemcpy() acts as blocking call.
- Device variable is pointer on CPU, holds address of GPU memory.

Thank You