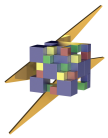
Lecture 23

Double Buffering and Local Memory

이재진

서울대학교 컴퓨터공학부

http://aces.snu.ac.kr



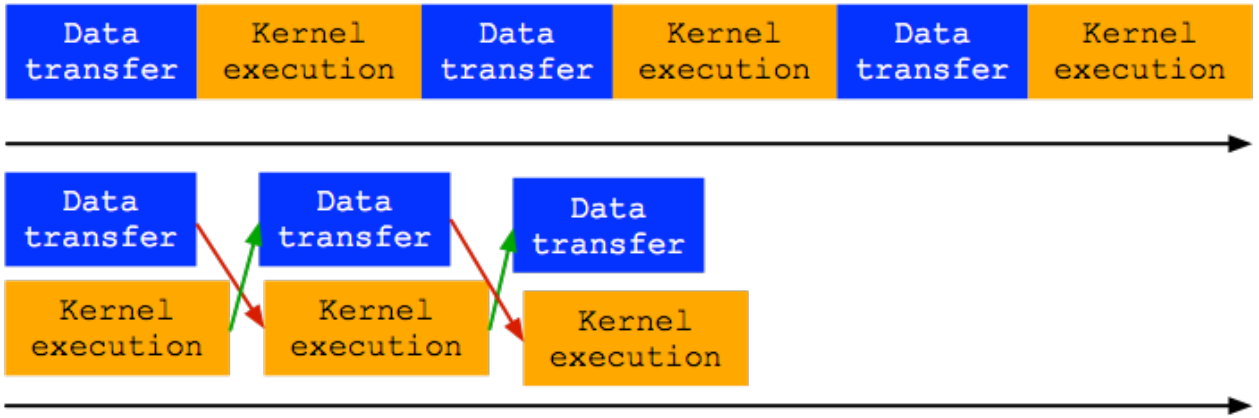
THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

Host-device Data Transfer

- Data transfer between the host and the device has much lower bandwidth than global memory accesses
  - PCI-E : a few GB/s
  - Global memory: a few hendred GB/s
- Minimize data transfer between the host and the device
  - Better to recompute on the accelerator
  - Use the global memory on the accelerator for intermediate data
- One large transfer is much faster than many small ones

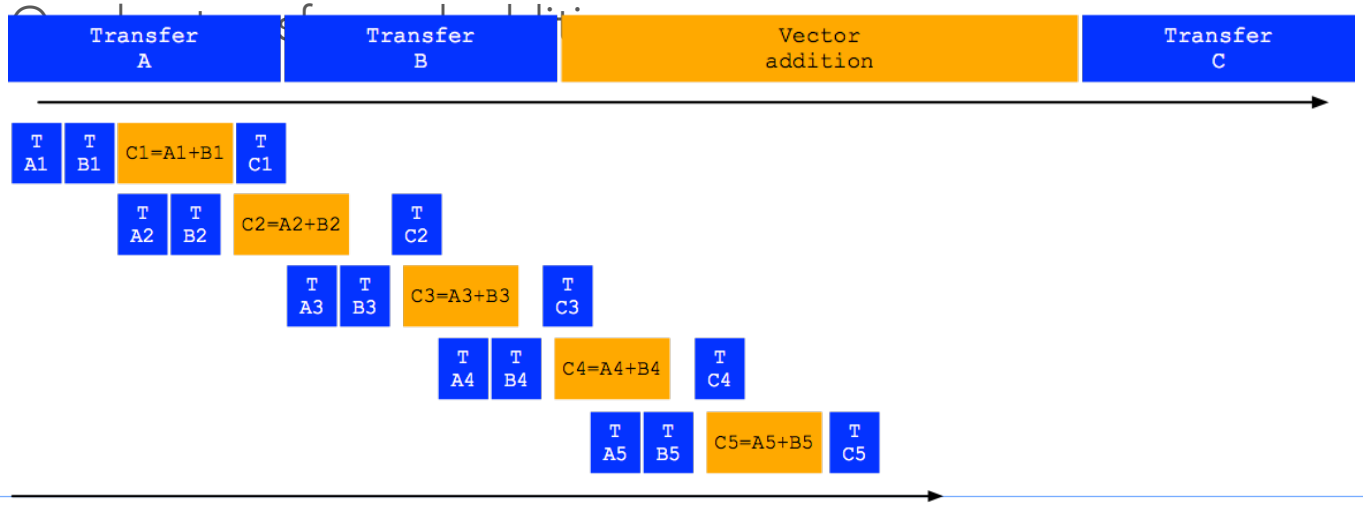
Asynchronous Copy

- Overlap of data transfer and code execution
  - Simultaneously execute a kernel while performing a copy between device and host memory
- Relevant only for accelerators using PCI-E bus
- Use two queues
  - One for data transfer and one for execution
  - Use events to enforce dependences



Pipelining (Double Buffering)

- Vector addition example
  - A + B = C
  - Divide large vectors into segments
    - A1, A2, ..., An
    - B1, B2, ..., Bn
    - C1, C2, ..., Cn
  -



Using Local Memory

- Local memory is local to a work-group
  - Shared by all work-items of work-group
  - Used to cache global memory
  - Low latency access

- Two ways to allocate it
  - Statically, inside the kernel
  - Dynamically, from the host as a parameter

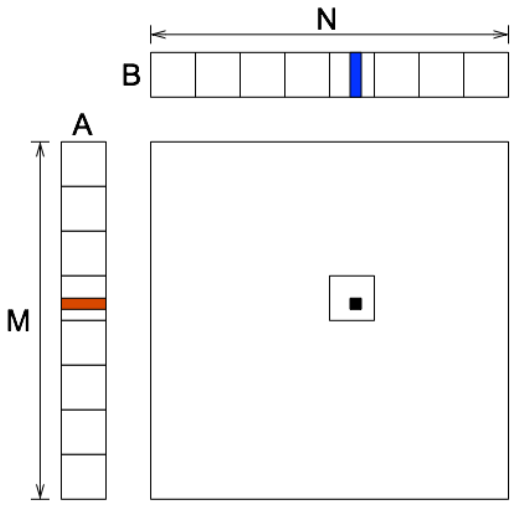
Allocating Local Memory

- Inside a kernel
  - Declare local memory as a static array
  - Use the keyword **`__local`**

- Kernel with parameter in local memory
  - Allocation done during set of kernel arguments

Matrix Multiplication Using Local Memory

- Every work-item takes care of one element in C




```
__kernel void MatMul(__global float* a,
                     __global float* b,
                     __global float* c,
                     int N)
{
    int row = get_global_id(1);
    int col= get_global_id(0);
    float sum = 0.0f;
    for (int i= 0; i < T_SIZE; i++) {
        sum += a[row*T_SIZE+i]
                * b[i*N+col];
    }
    c[row*N+col] = sum;
}
```

Matrix Multiplication Using Local Memory (cont'd)

```
__kernel void TMatMul(__global float* a,
                      __global float* b,
                      __global float* c,intN,
                      __local float tile[T_SIZE][T_SIZE])
{
    int row = get_global_id(1);
    int col= get_global_id(0);
    float sum = 0.0f;
    int x = get_local_id(0);
    int y = get_local_id(1);

    tile[y][x] = a[row*T_SIZE+x];
    for (int i= 0; i< T_SIZE; i++) {
        sum += tile[y][i]* b[i*N+col];
    }
    c[row*N+col] = sum;
}
```