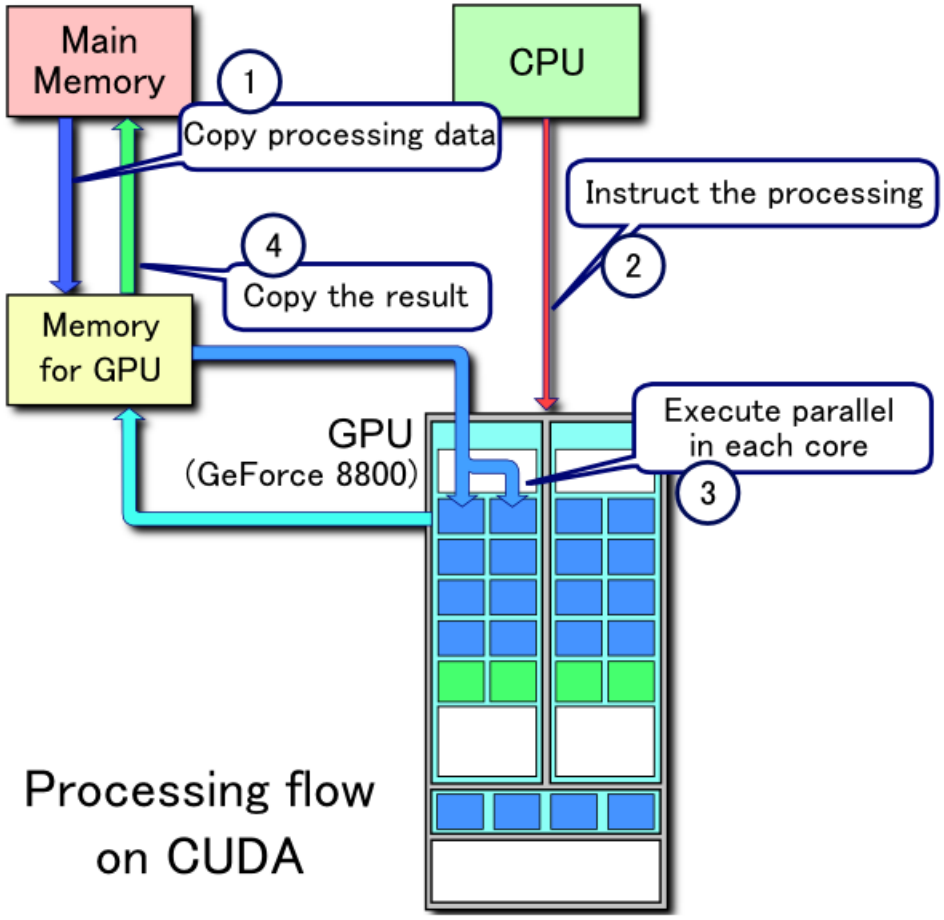
2110412 Parallel Comp Arch
CUDA Programming II

Natawut Nupairoj, Ph.D.
Department of Computer Engineering, Chulalongkorn University

Outline

- CUDA Shared Memory and Synchronization

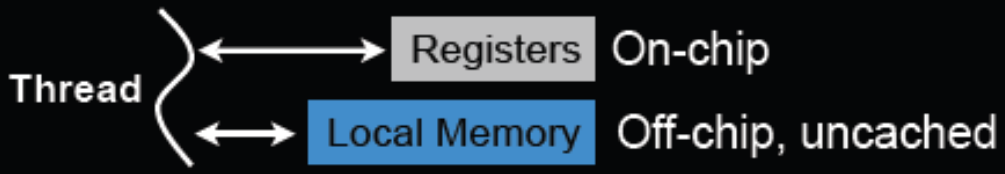
Processing Flow on CUDA



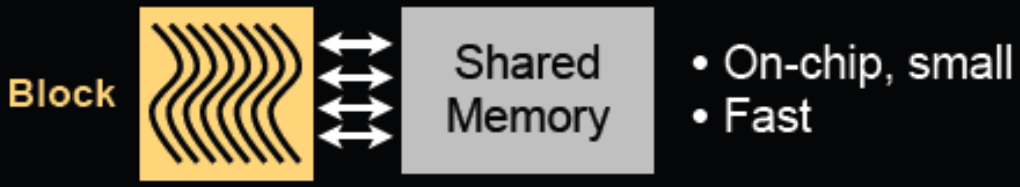
Source: Wikipedia

Kernel Memory Access

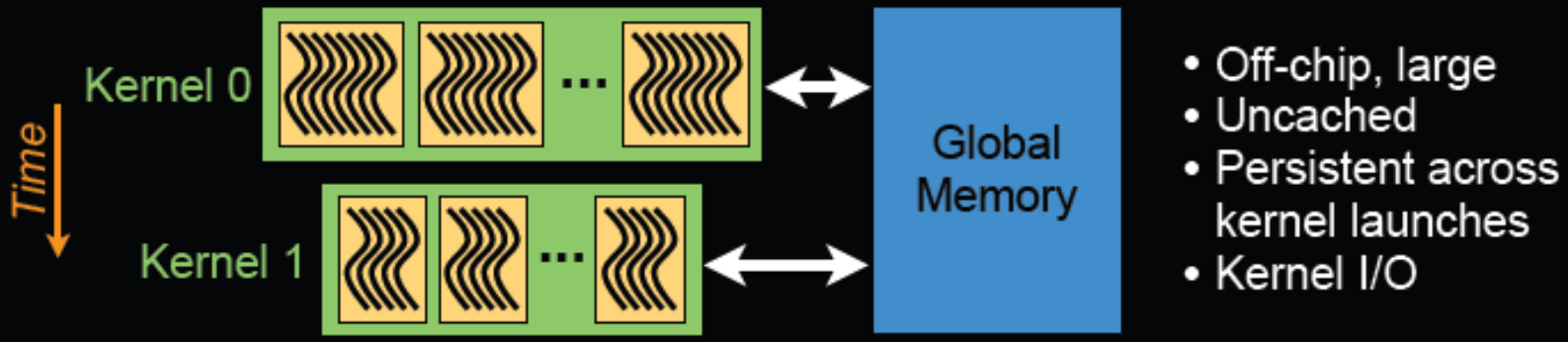
- **Per-thread**



- **Per-block**



- **Per-device**



Source: High Performance Computing with CUDA, DoD HPCMP: 2009

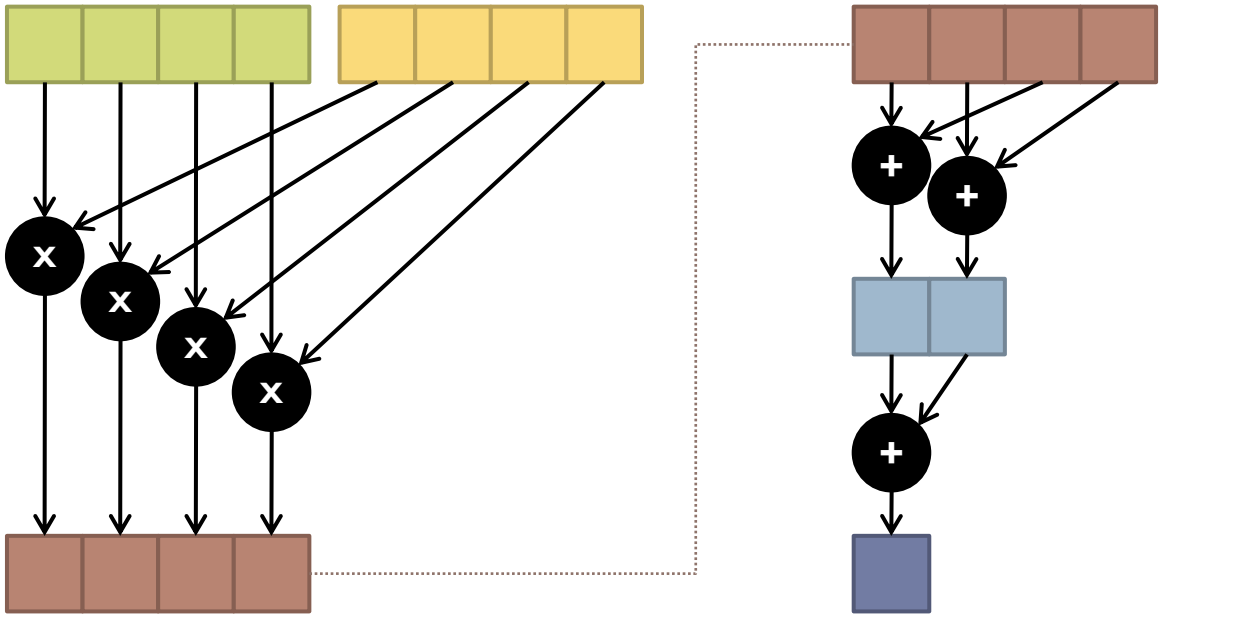
CUDA Shared Memory

- Each block has shared memory
  - Every thread in the same block shares the memory
  - Threads cannot see or modify the copy of the shared memory in other blocks
- Shared memory has very low access latency
  - Resided in GPU, not in off-chip DRAM
  - Excellent for communication and collaboration among threads in the same block

Example: Dot Product

- How can we improve the performance
  - Using shared memory as a cache
- Considering simplified example

$$(x_1, x_2, x_3, x_4) \bullet (y_1, y_2, y_3, y_4) = x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4$$



Example: Dot Kernel

```
__global__ void dot(float A[N], float B[N],  float C[N])
{
   __shared__ float cache[threadsPerBlock];
   int tid = threadIdx.x + blockIdx.x*blockDim.x;
   int cacheIndex = threadIdx.x;
   float temp = 0;
   while(tid < N) {
        temp += A[tid] * b[tid];
        tid += blockDim.x * gridDim.x;
   }
   cache[cacheIndex] = temp;
   __syncthreads();
```

Continued next page

From previous page

Example: Dot Kernel (cont)

```
    int i = blockDim.x/2;
    while(i != 0) {
        if(cacheIndex < i)
            cache[cacheIndex] += cache[cacheIndex + 1];
        __syncthreads();
        i /= 2;
    }
    if(cacheIndex == 0)
        C[blockIdx.x] = cache[0];
}
```

Example: Dot Main

```
#define imin(a,b) ( (a<b)? a : b)
const int N = 33
const int threadsPerBlock = 256;
const int blocksPerGrid =
   imin( 32, (N+threadsPerBlock-1)/threadsPerBlock );

void main()
{
   float *a, *b, *partial_c;
   float *dev_a, *dev_b, *dev_partial_c;

   // allocate memory on the CPU side
   a = (float *)malloc(N*sizeof(float));
   b = (float *)malloc(N*sizeof(float));
   partial_c = (float *)malloc(blocksPerGrid*sizeof(float));
```

Continued next page

From previous page

Example: Dot Main (cont)

```
// allocate memory on the GPU
cudaMalloc((void **) &dev_a, N*sizeof(float));
cudaMalloc((void **) &dev_b, N*sizeof(float));
cudaMalloc((void **) &dev_partial_c,
     blocksPerGrid*sizeof(float));

for(int i=0 ; i < N ; i++) {
     a[i] = i;
     b[i] = i*2;
}

// copy the arrays 'a' and 'b' on the host to the GPU
cudaMemcpy(dev_a, a, N*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(dev_b, b, N*sizeof(float), cudaMemcpyHostToDevice);
```

Continued next page

From previous page

Example: Dot Main (cont)

```
    // invoke kernel on the GPU
    dot<<blocksPerGrid,threadsPerBlock>>( dev_a, dev_b,
          dev_partial_c);

    // copy result back from the GPU
    cudaMemcpy(partial_c, dev_partial_c,
          blocksPerGrid*sizeof(float), cudaMemcpyDeviceToHost);

    // finish up the computation on the CPU
    float c = 0;
    for(int i=0 ; i < blocksPerGrid ; i++)
          c += partial_c[i];

    // free all memory
    cudaFree(dev_a); cudaFree(dev_b); cudaFree(dev_partial_c);
    free(a); free(b); free(partial_c);
}
```