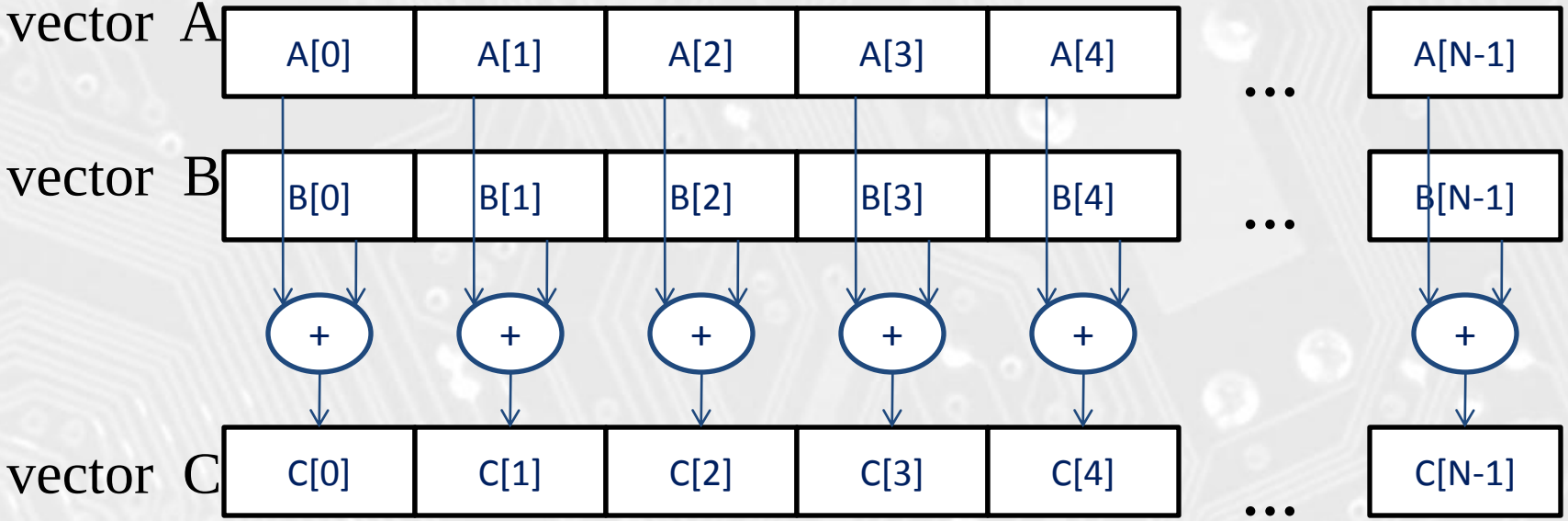
Introduction to CUDA C

Wen-mei Hwu
University of Illinois at Urbana-Champaign

Objective

- To learn about data parallelism and the basic features of CUDA C, a heterogeneous parallel programming interface that enables exploitation of data parallelism
  - Hierarchical thread organization
  - Main interfaces for launching parallel execution
  - Thread index(es) to data index mapping

Data Parallelism - Vector Addition Example



CUDA /OpenCL – Execution Model

- Heterogeneous host+device application C program
  - Serial parts in **host** C code
  - Parallel parts in **device** SPMD kernel C code

Serial Code (host)

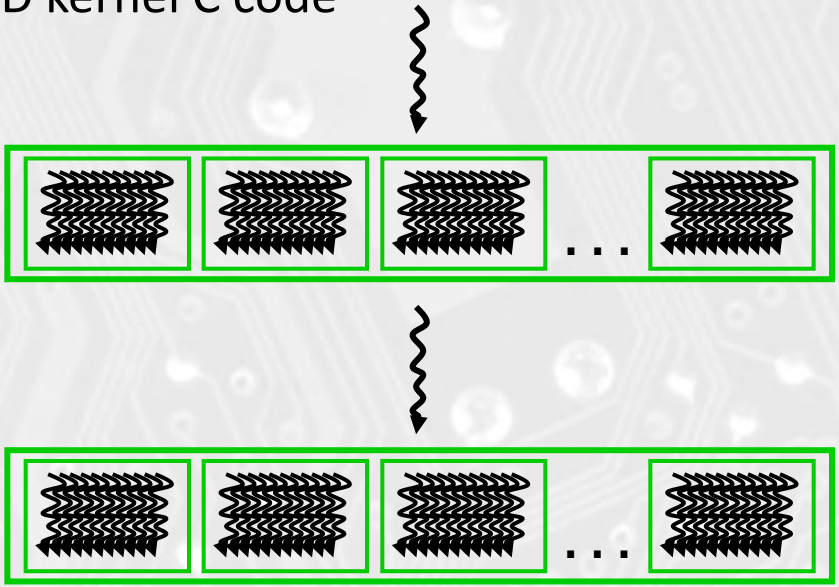
Parallel Kernel (device)

KernelA<<< nBlk, nTid >>>(args);

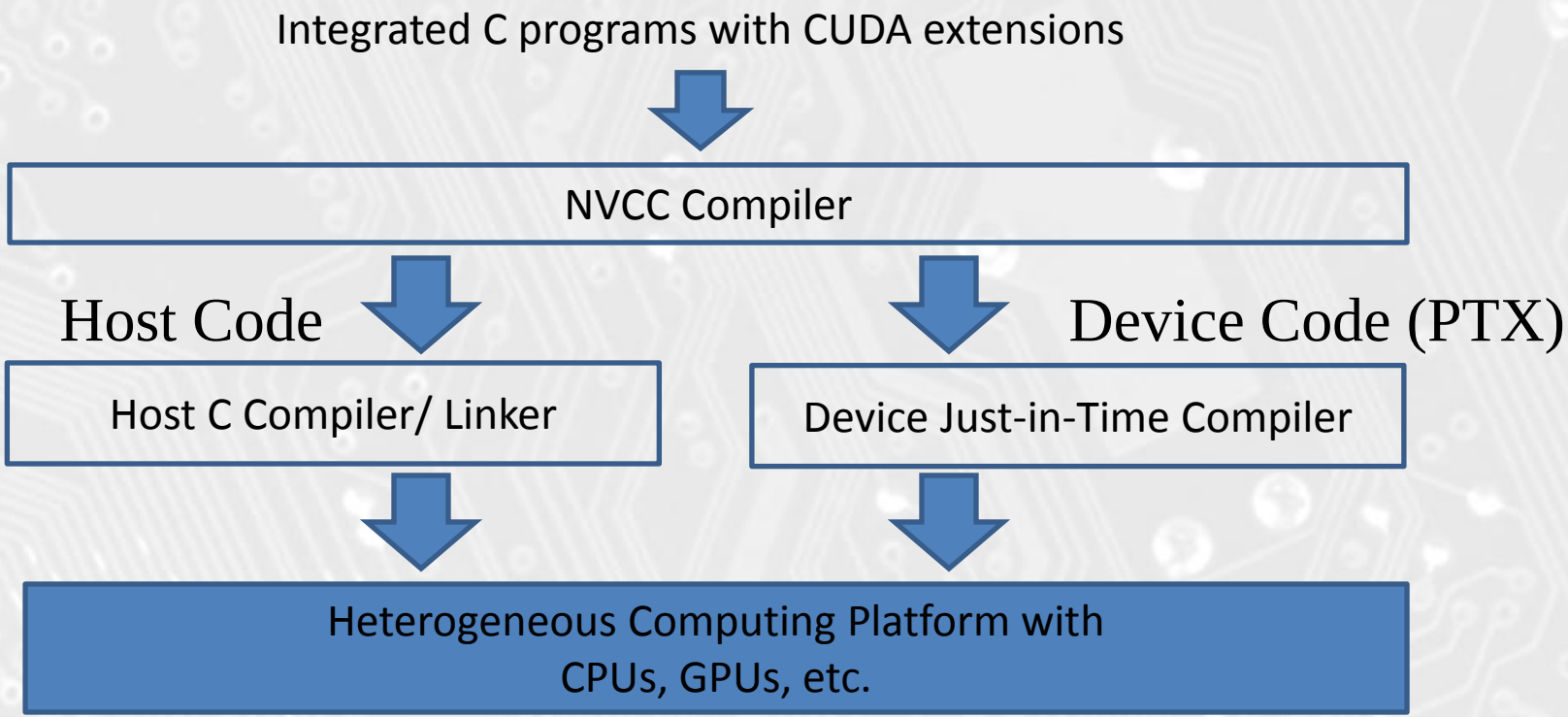
Serial Code (host)

Parallel Kernel (device)

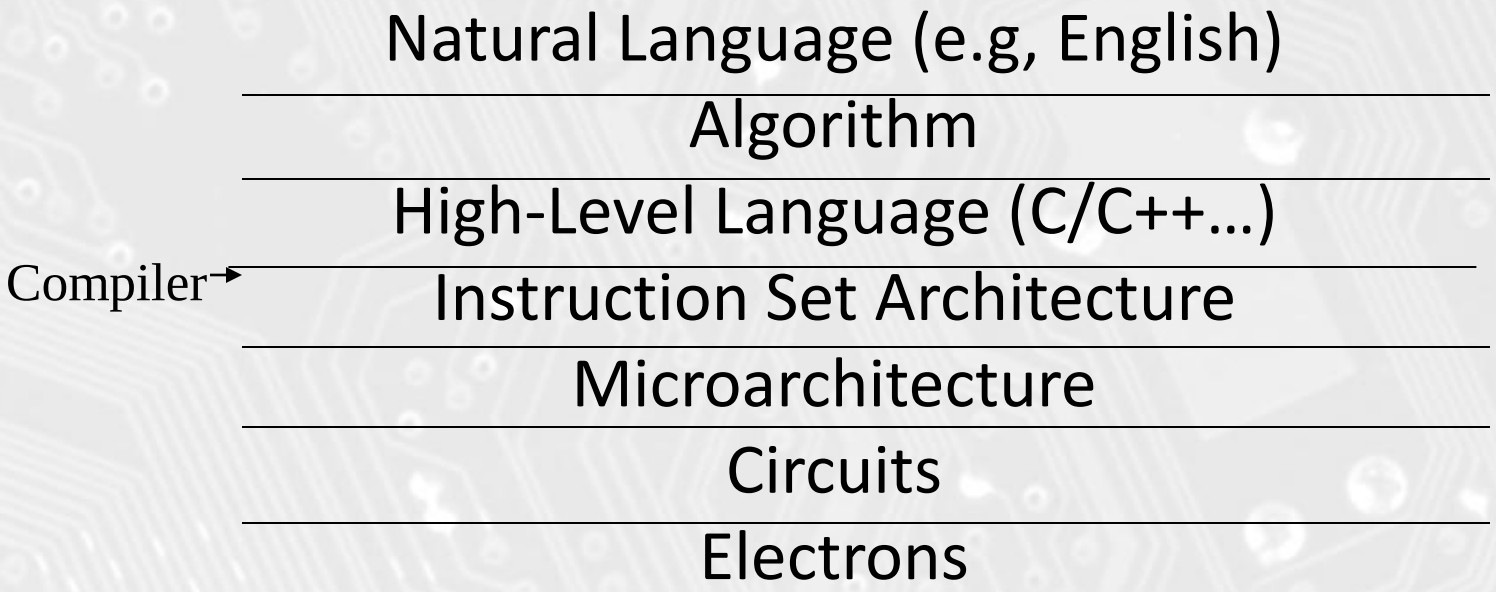
KernelB<<< nBlk, nTid >>>(args);



Compiling A CUDA Program

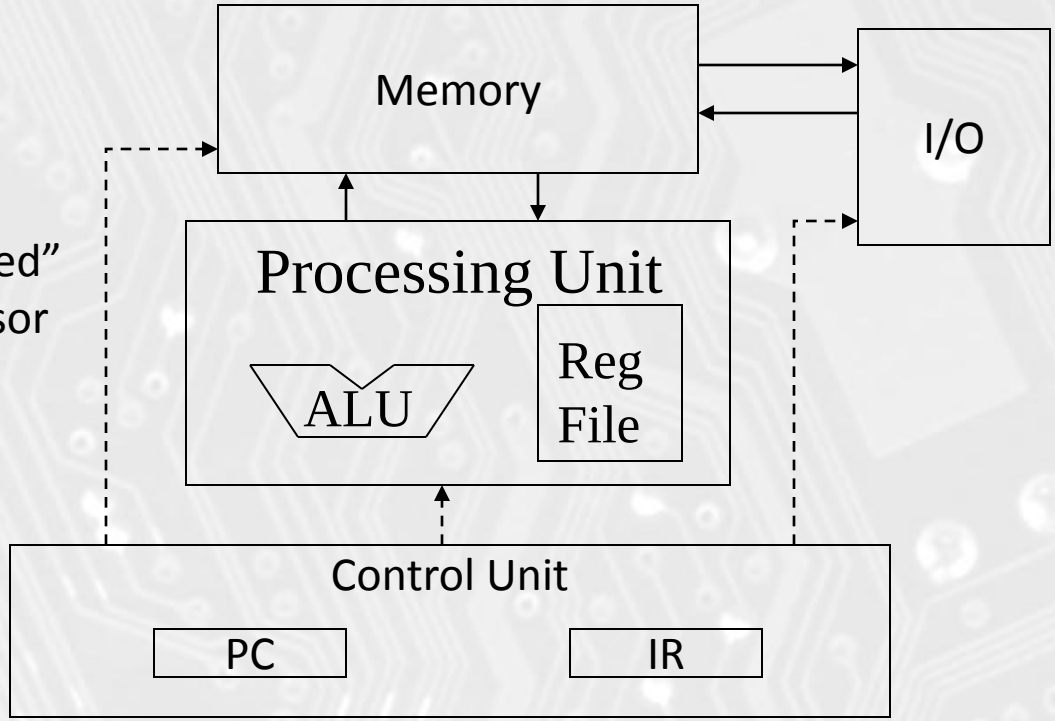


From Natural Language to Electrons



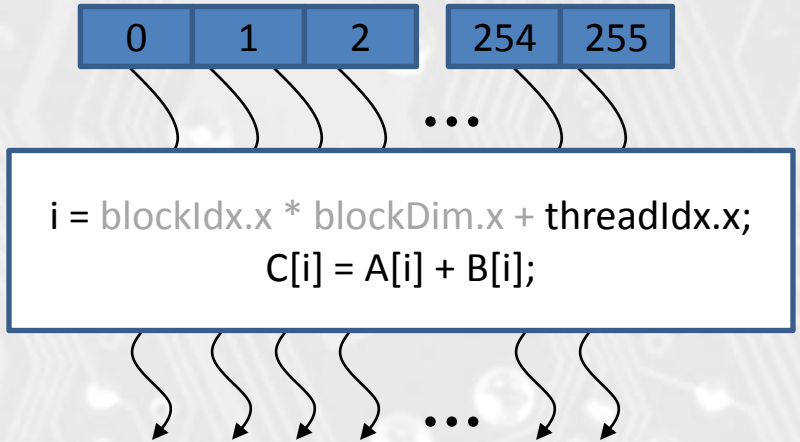
A Von-Neumann Processor

A thread is a “virtualized” Von-Neumann Processor



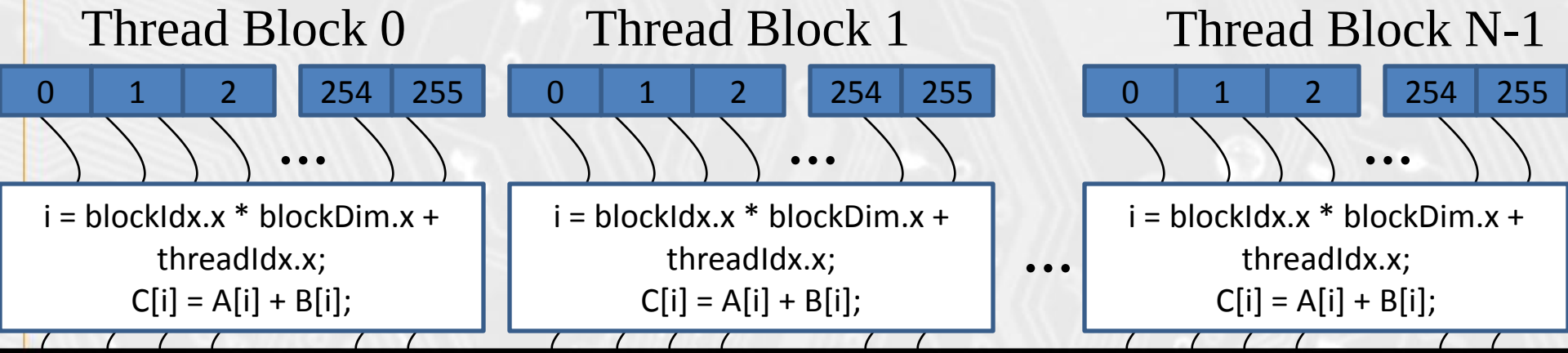
Arrays of Parallel Threads

- A CUDA kernel is executed by a grid (array) of threads
  - All threads in a grid run the same kernel code (SPMD)
  - Each thread has an index that it uses to compute memory addresses and make control decisions



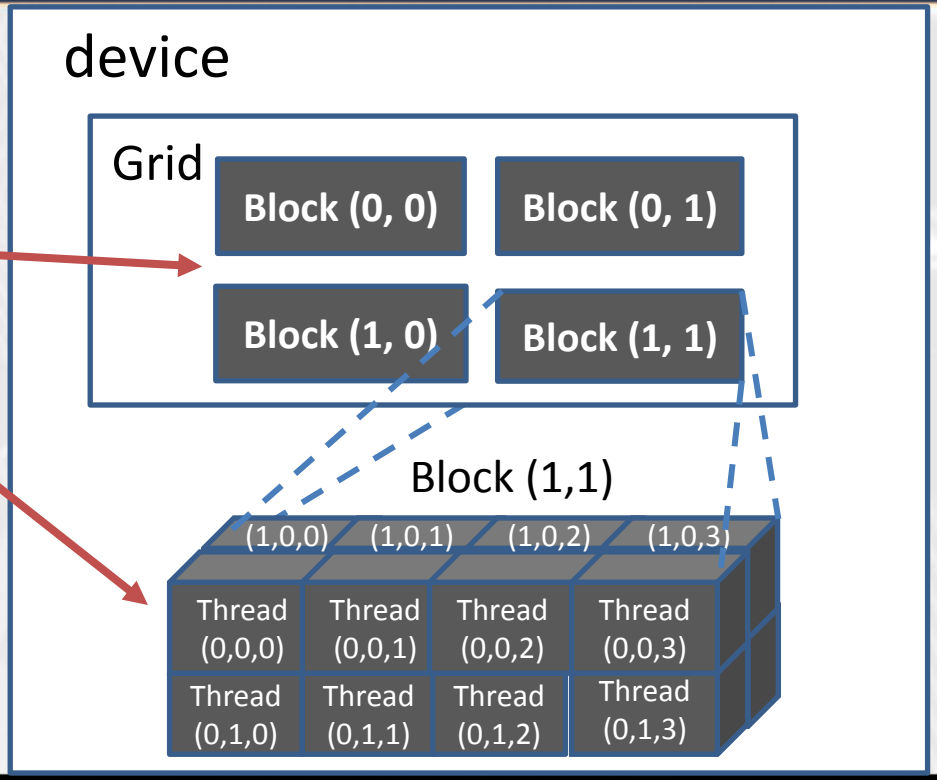
Thread Blocks: Scalable Cooperation

- Divide thread array into multiple blocks
  - Threads within a block cooperate via **shared memory, atomic operations** and **barrier synchronization**
  - Threads in different blocks do not interact

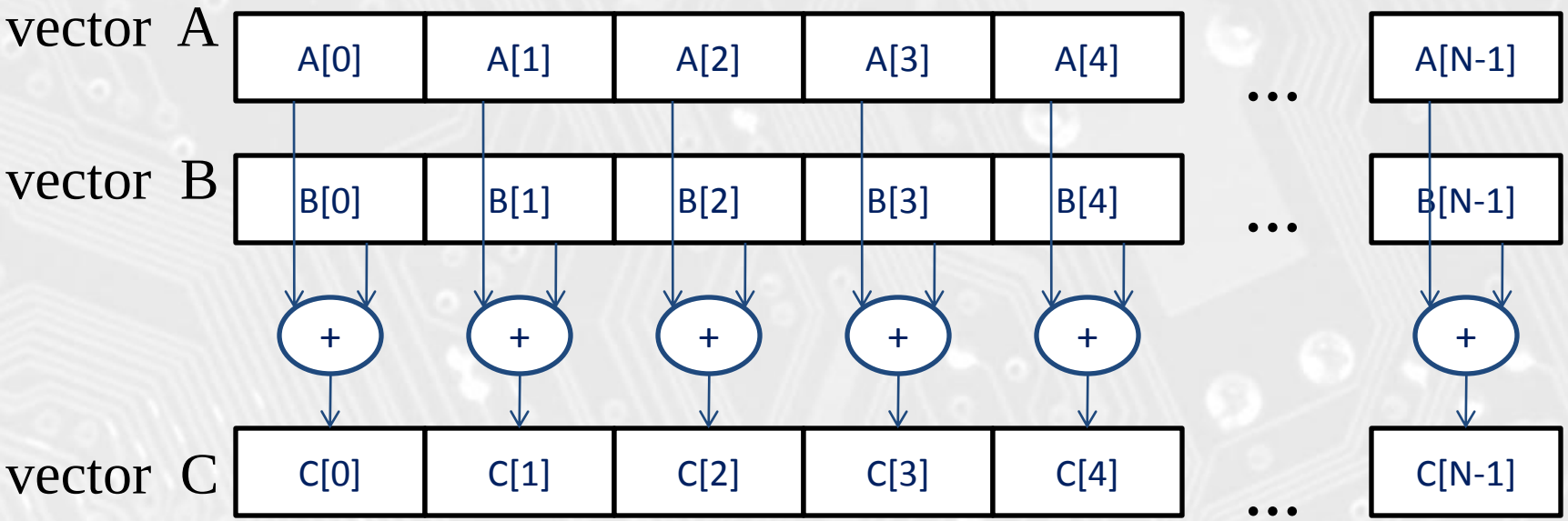


blockIdx and threadIdx

- Each thread uses indices to decide what data to work on
  - blockIdx: 1D, 2D, or 3D (CUDA 4.0)
  - threadIdx: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes
  - …



Vector Addition – Conceptual View

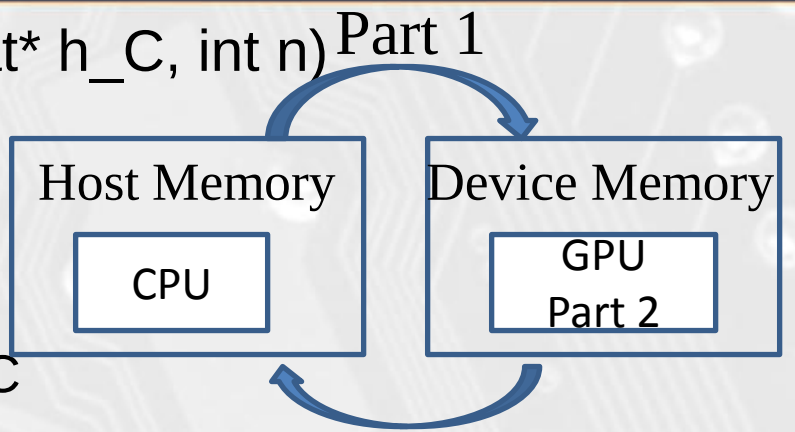


Vector Addition – Traditional C Code

```
    // Compute vector sum C = A+B
void vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
  int i;
  for (i = 0; i < n; i++) h_C[i] = h_A[i] + h_B[i];
}
int main()
{
    // Memory allocation for h_A, h_B, and h_C
    // I/O to read h_A and h_B, N elements
    …
    vecAdd(h_A, h_B, h_C, N);
}
```

Heterogeneous Computing vecAdd CUDA Host Code

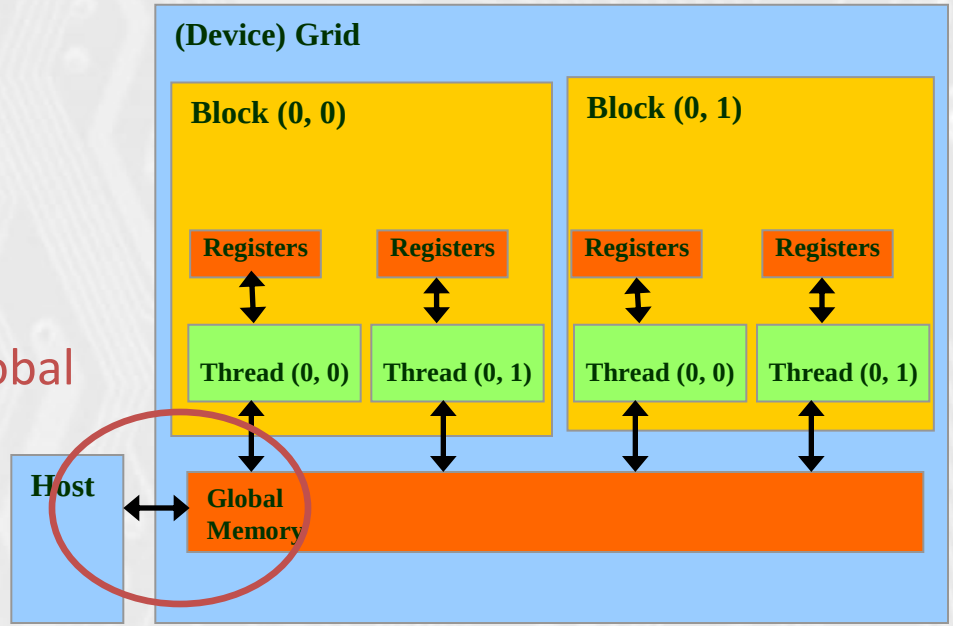
```
void vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
   int size = n* sizeof(float);
   float* d_A, d_B, d_C;
   …
1. // Allocate device memory for A, B, and C
    // copy A and B to device memory

2. // Kernel launch code – the device performs the actual vector addition

3. // copy C from the device memory  // Free device vectors
}
```



Partial Overview of CUDA Memories

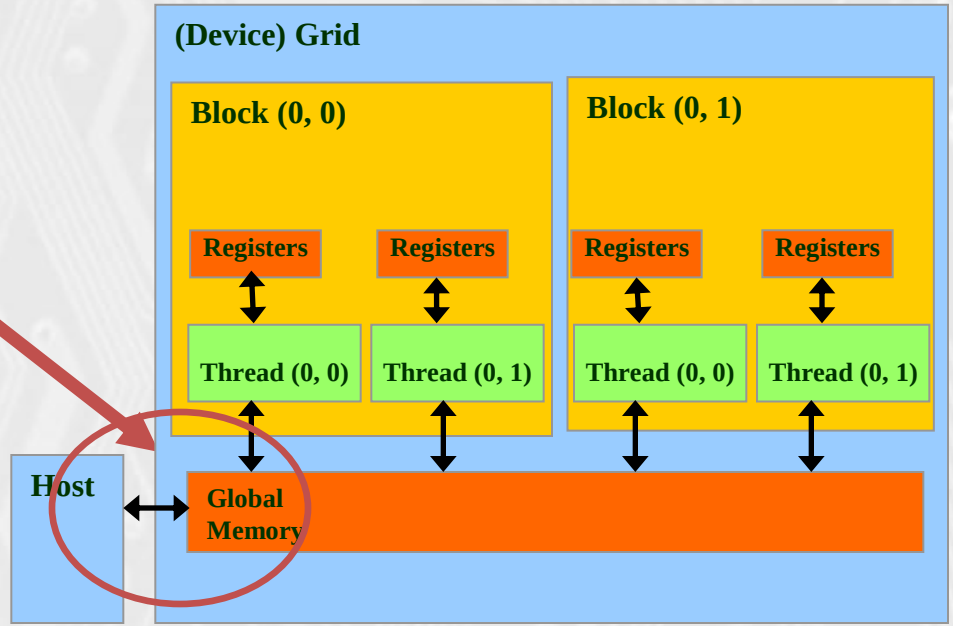
- Device code can:
  - R/W per-thread registers
  - R/W all-shared global memory
- Host code can
  - Transfer data to/from per grid global memory

We will cover more memory types later.



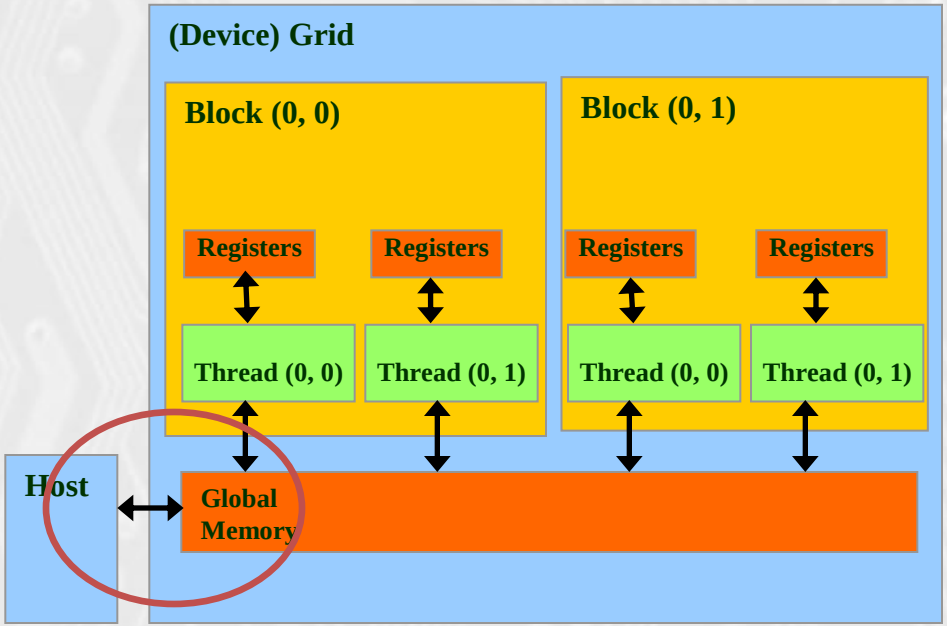
CUDA Device Memory Management API functions

- cudaMalloc()
  - Allocates object in the device global memory
  - Two parameters
    - **Address of a pointer** to the allocated object
    - **Size of** of allocated object in terms of bytes
- cudaFree()
  - Frees object from device global memory
    - **Pointer** to freed object



Host-Device Data Transfer API functions

- cudaMemcpy()
  - memory data transfer
  - Requires four parameters
    - Pointer to destination
    - Pointer to source
    - Number of bytes copied
    - Type/Direction of transfer
  - Transfer to device is asynchronous



Vector Addition Host Code

```
void vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
    int size = n * sizeof(float);  float* d_A, d_B, d_C;

    cudaMalloc((void **) &d_A, size);
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMalloc((void **) &d_B, size);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
    cudaMalloc((void **) &d_C, size);
// Kernel invocation code – to be shown later
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    cudaFree(d_A); cudaFree(d_B); cudaFree (d_C);
}
```

In Practice, Check for API Errors

```
cudaError_t  err = cudaMalloc((void **) &d_A, size);
if (error != cudaSuccess)  {
   printf("%s in %s at line %d\n",
         cudaGetErrorString(err), __FILE__, __LINE__);
   exit(EXIT_FAILURE);
}
```

Example: Vector Addition Kernel

Device Code

```
// Compute vector sum C = A+B
// Each thread performs one pair-wise addition

__global__
void vecAddKernel(float* A, float* B, float* C, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if(i<n) C[i] = A[i] + B[i];
}
```

Example: Vector Addition Kernel

```
__global__
void vecAddkernel(float* A, float* B, float* C, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if(i<n) C[i] = A[i] + B[i];
}
```

```
int vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
 // d_A, d_B, d_C allocations and copies omitted
 // Run ceil(n/256.0) blocks of 256 threads each
  vecAddKernnel<<<ceil(n/256.0),256>>>(d_A, d_B, d_C, n);
}
```

Host Code

More on Kernel Launch

```
int vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
 // d_A, d_B, d_C allocations and copies omitted
 // Run ceil(n/256.0) blocks of 256 threads each
  dim3 DimGrid(n/256, 1, 1);
  if (n%256) DimGrid.x++;
  dim3 DimBlock(256, 1, 1);

  vecAddKernnel<<<DimGrid,DimBlock>>>(d_A, d_B, d_C, n);
}
```

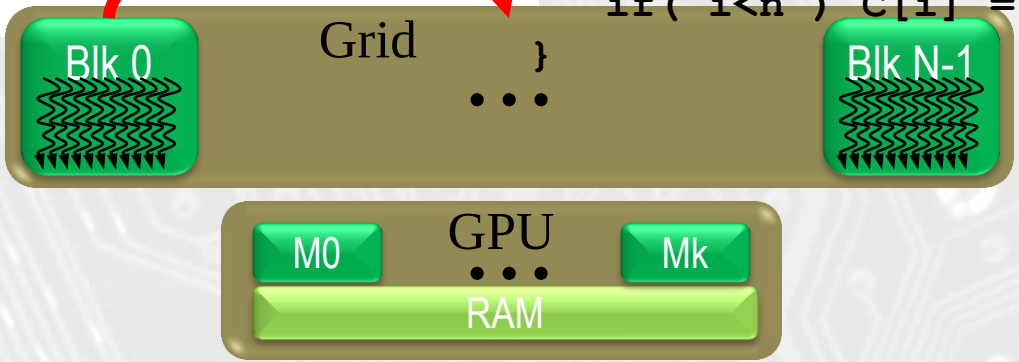
Host Code

This makes sure that there are enough threads to cover all elements.

Kernel execution in a nutshell

```
__host__
Void vecAdd(…)
{
  dim3 DimGrid = (ceil(n/256.0),1,1);
  dim3 DimBlock = (256,1,1);
vecAddKernel<<<DimGrid,DimBlock>>>(d_
A,d_B,d_C,n);
}
```

```
__global__
void vecAddKernel(float *A,
     float *B, float *C, int n)
{
  int i = blockIdx.x * blockDim.x
     + threadIdx.x;

  if( i<n ) C[i] = A[i]+B[i];
}
```



More on CUDA Function Declarations

	Executed on the:	Only callable from the:
`__device__ float DeviceFunc()`	device	device
`__global__ void KernelFunc()`	device	host
`__host__ float HostFunc()`	host	host

- `__global__` defines a kernel function
  - Each "__" consists of two underscore characters
  - A kernel function must return `void`
- `__device__` and `__host__` can be used together

THANK YOU! ANY MORE QUESTIONS?