GPU Teaching Kit

Accelerated Computing

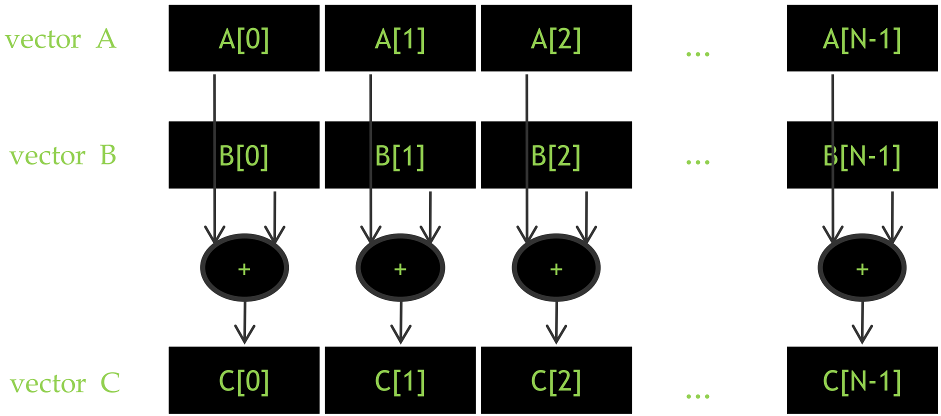
Lecture 2.2 - Introduction to CUDA C

Memory Allocation and Data Movement API Functions

Objective

- To learn the basic API functions in CUDA host code
  - Device Memory Allocation
  - Host-Device Data Transfer

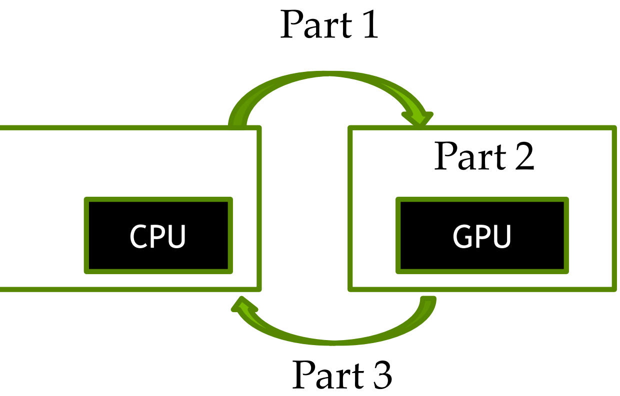
Data Parallelism - Vector Addition Example



Vector Addition – Traditional C Code

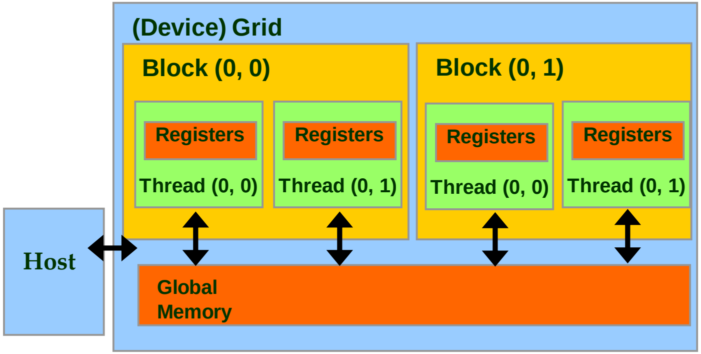
```
// Compute vector sum C = A + B
void vecAdd(float *h_A, float *h_B, float *h_C, int n)
{
    int i;
    for (i = 0; i<n; i++) h_C[i] = h_A[i] + h_B[i];
}

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




```
#include <cuda.h>
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;
  // Part 1
  // Allocate device memory for A, B, and C
  // copy A and B to device memory

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

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

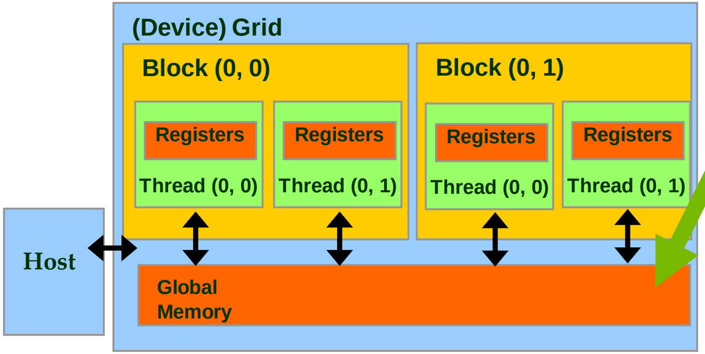
Partial Overview of CUDA Memories



- 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

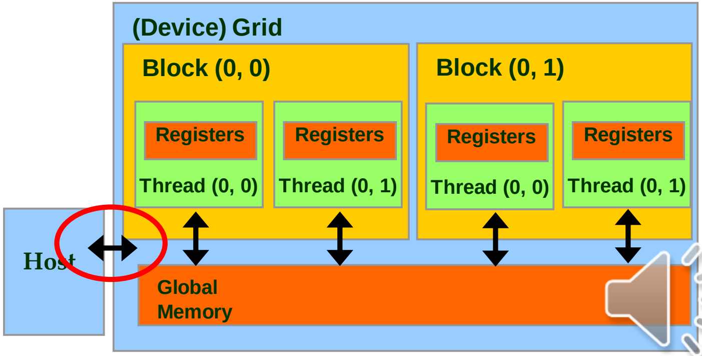
We will cover more memory types and more sophisticated memory models later.

CUDA Device Memory Management API functions



- cudaMalloc()
  - Allocates an object in the device global memory
  - Two parameters
    - **Address of a pointe**r to the allocated object
    - **Size of** allocated object in terms of bytes
- cudaFree()
  - Frees object from device global memory
  - One parameter
    - **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

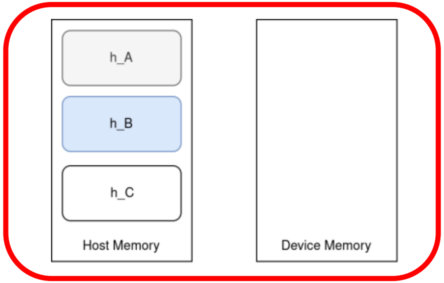
  – Transfer to device is synchronous with respect to the host

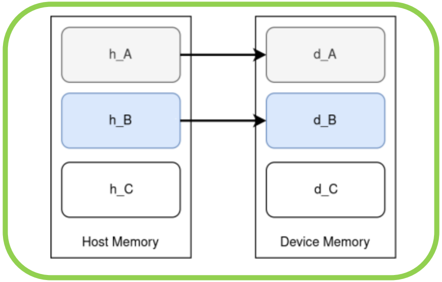
Vector Addition, Explicit Memory Management

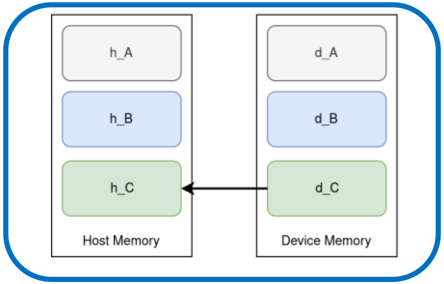
… Allocate h_A, h_B, h_C …

```
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);
    cudaMalloc((void **) &d_B, size);
    cudaMalloc((void **) &d_C, size);

    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Kernel invocation code – to be shown later

    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    cudaFree(d_A); cudaFree(d_B); cudaFree (d_C);
}
```

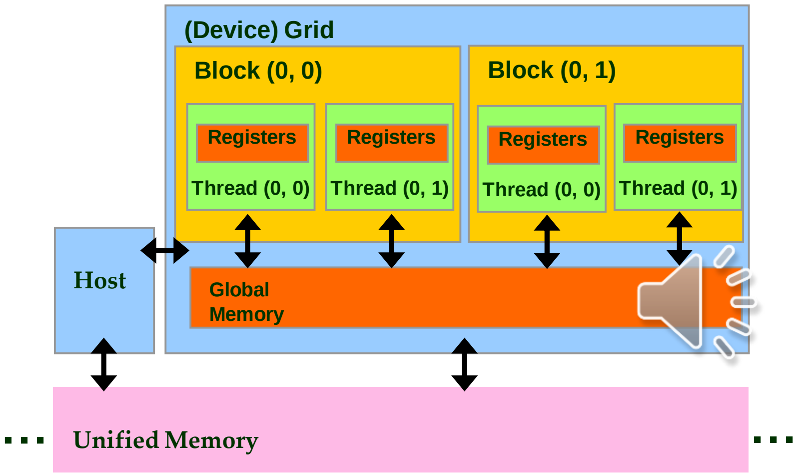
… Free h_A, h_B, h_C …







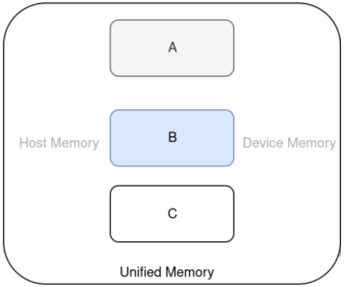
Unified Memory



- cudaMallocManaged(void** ptr, size_t size)
  - Single memory space for all CPUs/GPUs
    - Maintain single copy of data
  - CUDA-managed data
    - On-demand page migration
  - Compatible with cudaMalloc(), cudaFree()
  - Can be optimized
    - cudaMemAdvise(), cudaMemPrefetchAsync(),
    - cudaMemcpyAsync()

Vector Addition, Unified Memory

```
float *A, *B, *C
cudaMallocManaged(&A, n * sizeof(float));
cudaMallocManaged(&B, n * sizeof(float));
cudaMallocManaged(&C, n * sizeof(float));

// Initialize A, B

void vecAdd(float *A, float *B, float *C, int n)
{
 // Kernel invocation code – to be shown later
}

cudaFree(A);
cudaFree(B);
cudaFree(C);
```



In Practice, Check for API Errors in Host Code

```
cudaError_t  err = cudaMalloc((void **) &d_A, size);

if (err != cudaSuccess)  {
   printf(“%s in %s at line %d\n”,   cudaGetErrorString(err), __FILE__,
   __LINE__);
   exit(EXIT_FAILURE);
}
```

GPU Teaching Kit

Accelerated Computing