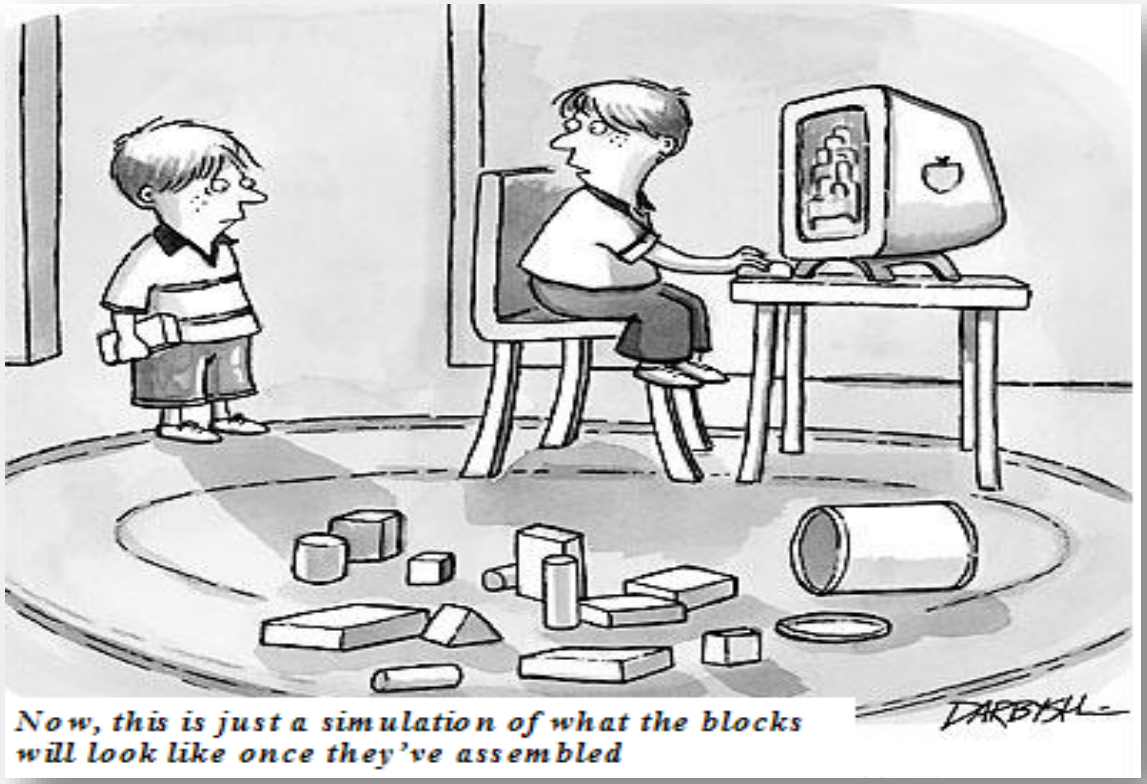
ECE569
Module 6



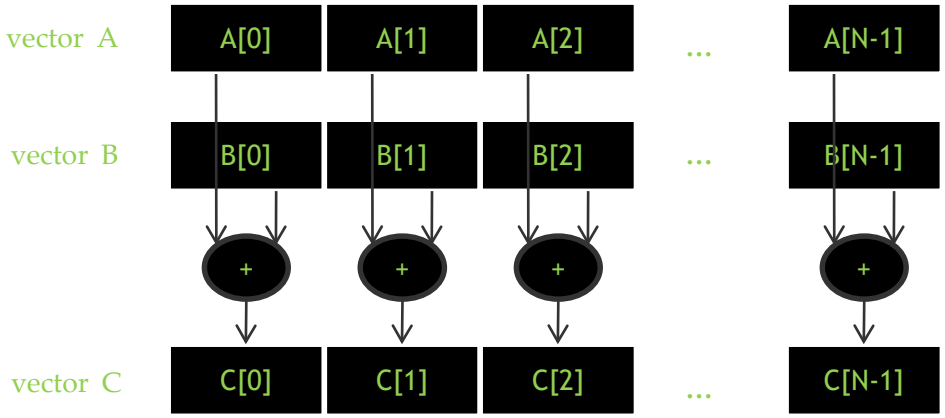
- Data movement between Host-Device

Memory Allocation and Data Movement API

- **Basic API functions in CUDA host code**
  - Device Memory Allocation
  - Host-Device Data Transfer

Data Parallelism – Vector Addition

- **First identify the level of parallelism**
  - Independent computations for each pair of inputs

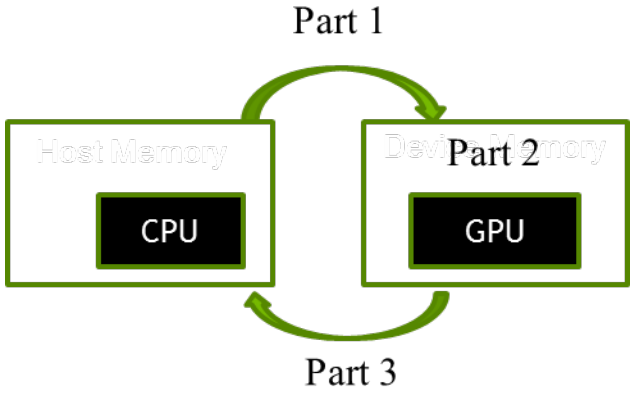


Vector Addition – Traditional C Code

```
// 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);
}
```

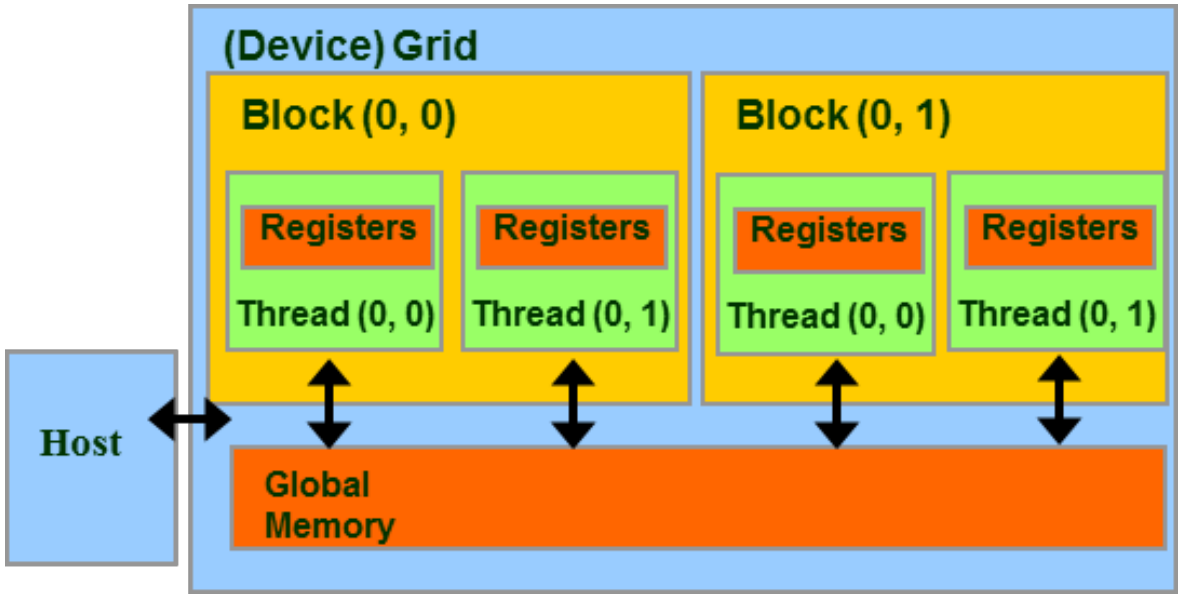
Heterogeneous Computing vecAdd CUDA Host Code

```
#include <cuda.h>  //CUDA API functions
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
}
```



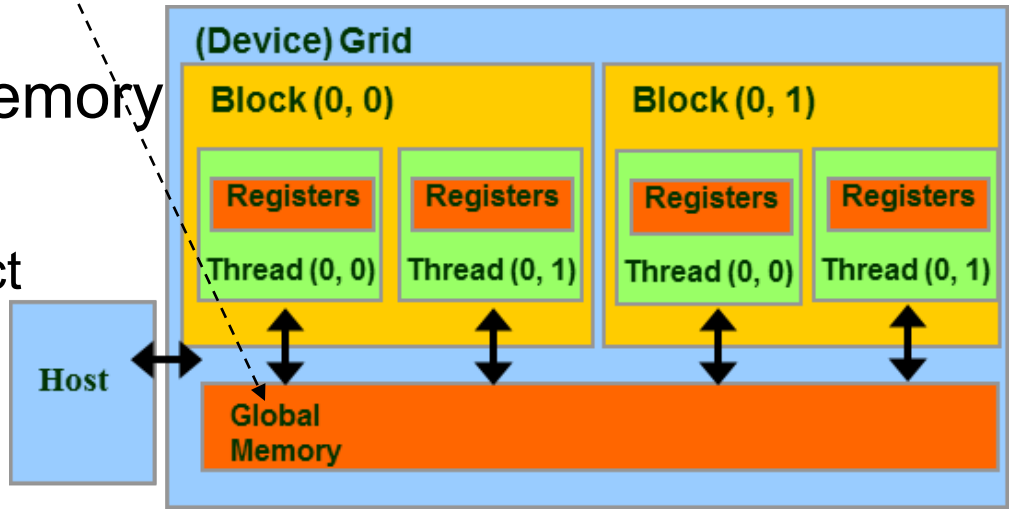
CUDA Memory Model - Simplified

- **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



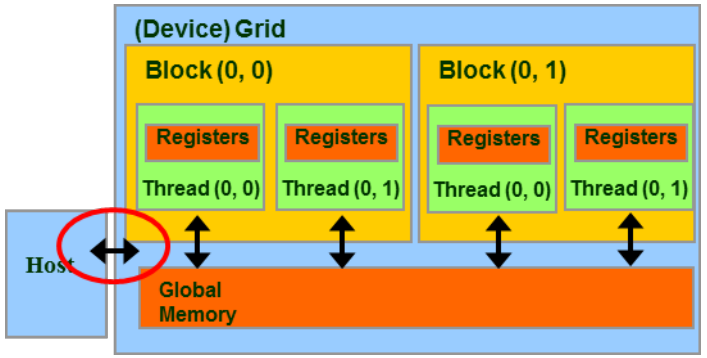
CUDA Device Memory Management API functions

- **cudaMalloc()**
  - Allocates an object in the device **global memory**
  - **Part of the host code**
  - Two parameters
    - Address of a pointer to the allocated object
      - In C: returns the pointer to the allocated object
      - All CUDA API functions return error code
    - Size of allocated object in terms of bytes
- **cudaFree()**
  - Frees from 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**

The address of the pointer

```
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);
        ….
        cudaFree(d_A);
}
```

The address of the pointer variable should be cast to (void **) because the function **expects a generic pointer** that is not restricted to any particular type of objects

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);
}
```

cudaMemcpy

- **Type of memory involved**
  - from host memory to host memory,
  - from host memory to device memory,
  - from device memory to host memory,
  - from device memory to device memory.
    - For example, the memory copy function can be used to copy data from one location of the device memory to another location of the device memory

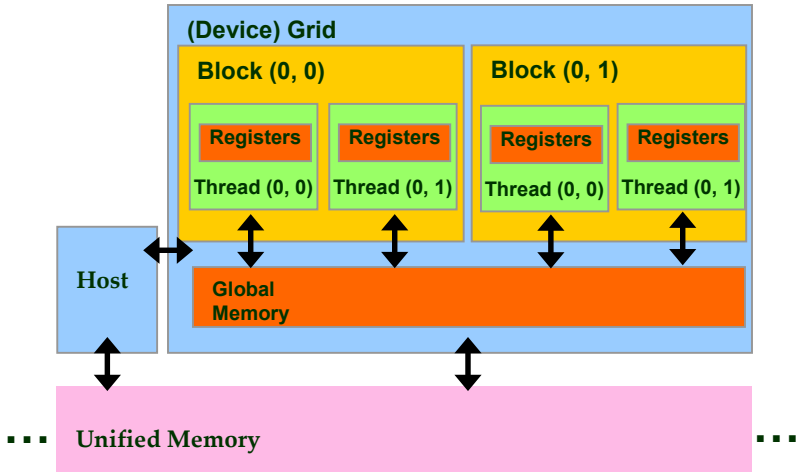
In Practice, Check for API Errors in Host Code

```
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);
}
```

- **Typical error: When global memory doesn't have enough space**

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()

Data Management with Unified Memoery

```
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);
```

Quick Check

The GPU can do the following:

- ❑ Initiate data send GPU->CPU
- ❑ Respond to CPU request to send GPU->CPU
- ❑ Initiate data request CPU->GPU
- ❑ Respond to CPU request to receive CPU->GPU
- ❑ Compute a kernel launched by CPU

Next

- **Thread organization**