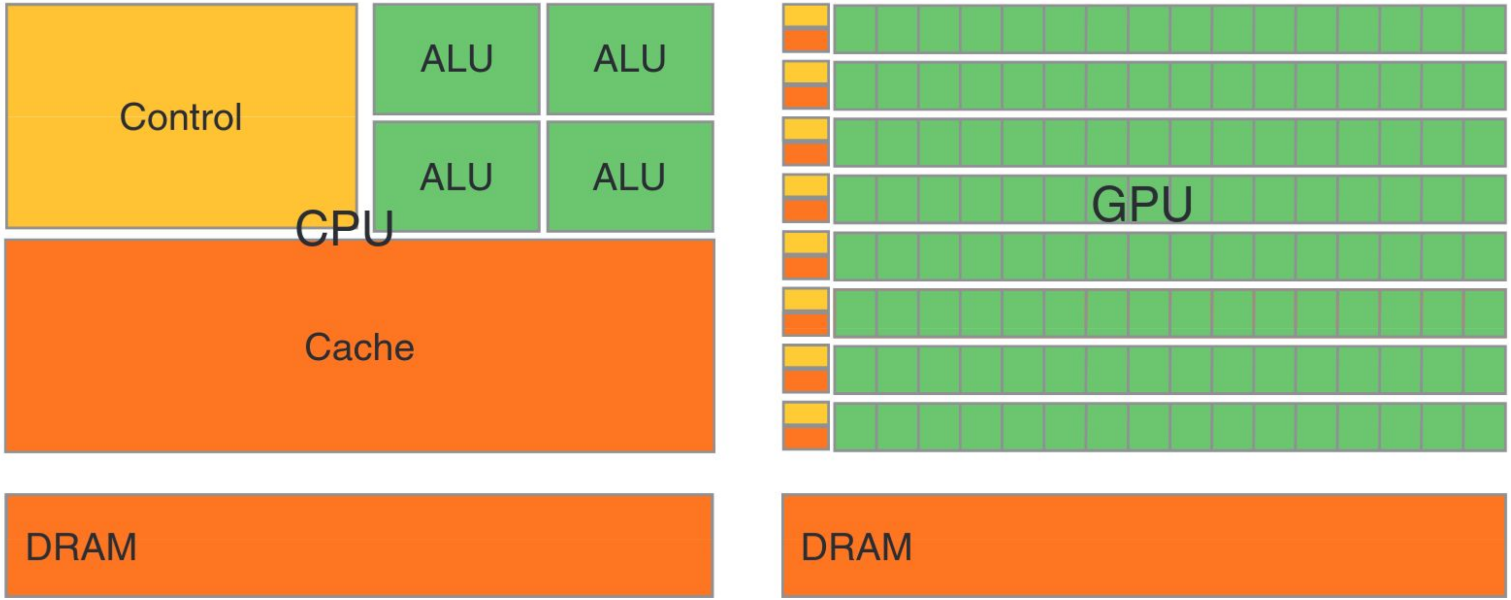
Vector Addition

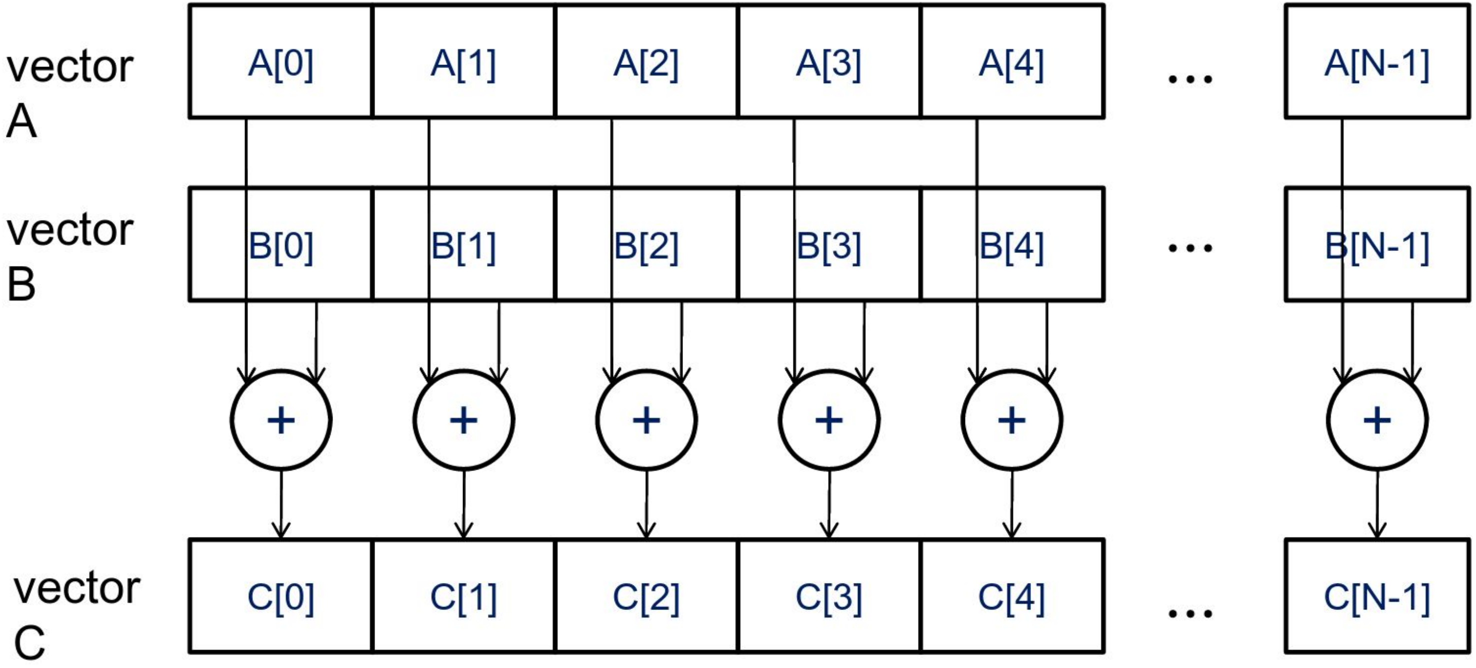
Simple Example

John H. Osorio Ríos

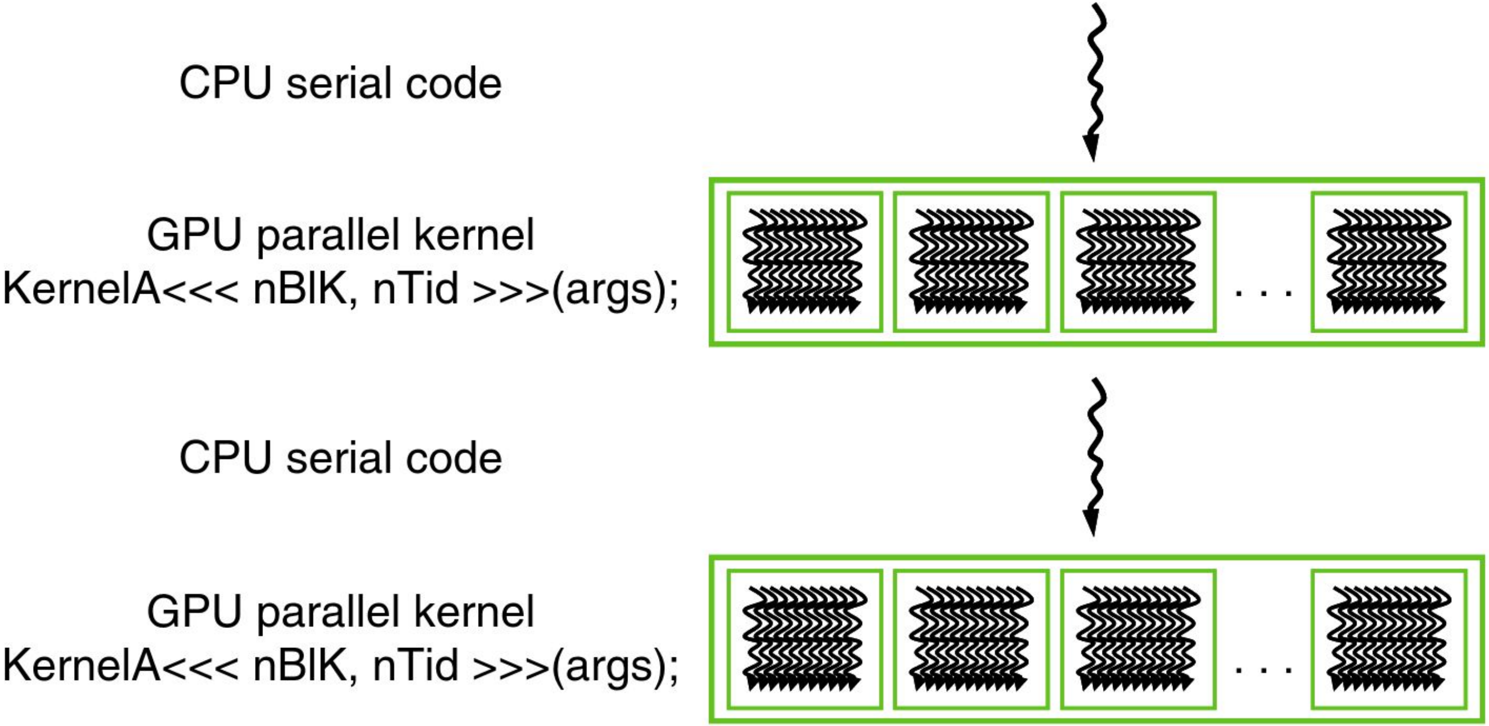
Overview (1/3)



Overview (2/3)


vector A
A[0]
A[1]
A[2]
A[3]
A[4]
…
A[N-1]
vector B
B[0]
B[1]
B[2]
B[3]
B[4]
…
B[N-1]
+
+
+
+
+
+
vector C
C[0]
C[1]
C[2]
C[3]
C[4]
…
C[N-1]

Overview (3/3)



CPU serial code

GPU parallel kernel
KernelA<<< nBlK, nTid >>>(args);

CPU serial code

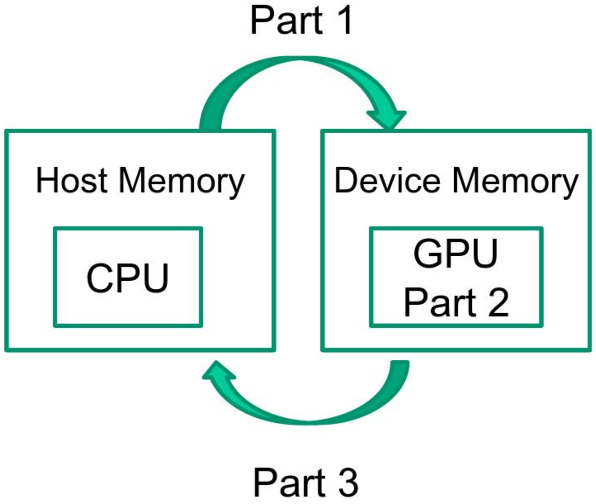
GPU parallel kernel
KernelA<<< nBlK, nTid >>>(args);

Traditional C Vector Addition

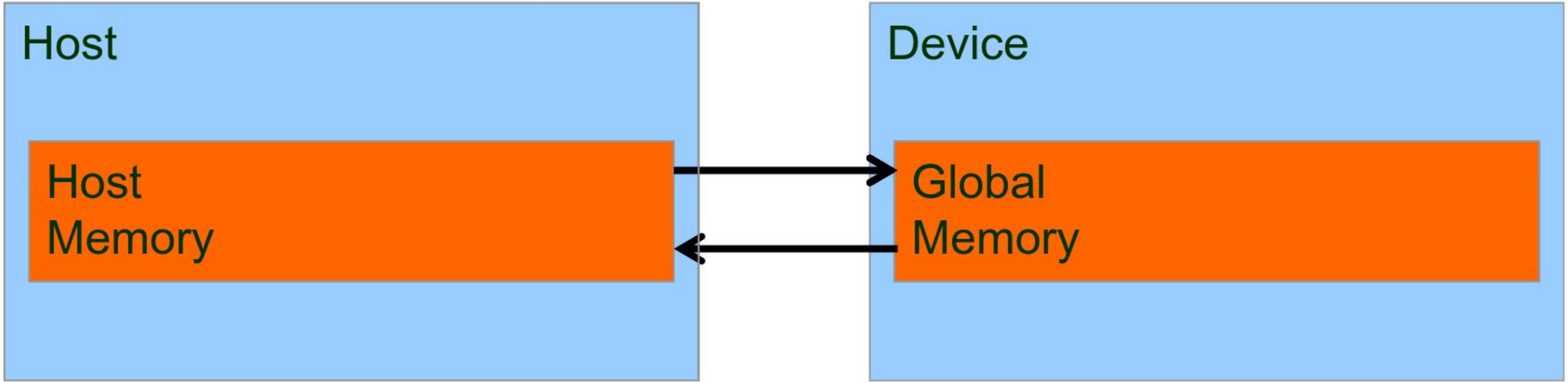
```
// Compute vector sum h_C = h_A+h_B
void vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
  for (i = 0; i < n; i++) h_C[i] = h_A[i] + h_B[i];
}

int main()
{
    // Memory allocation for h_A, h_B, and h_C
    // I/O to read h_A and h_B, N elements each
    …
    vecAdd(h_A, h_B, h_C, N);
}
```

vecAdd Process

```
#include <cuda.h>
…
void vecAdd(float* A, float*B, float* C, int n)
{
   int size = n* sizeof(float);
    float *A_d, *B_d, *C_d;
   …
1. // Allocate device memory for A, B, and C
    // copy A and B to device memory

2. // Kernel launch code – to have the device
    // to perform the actual vector addition

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



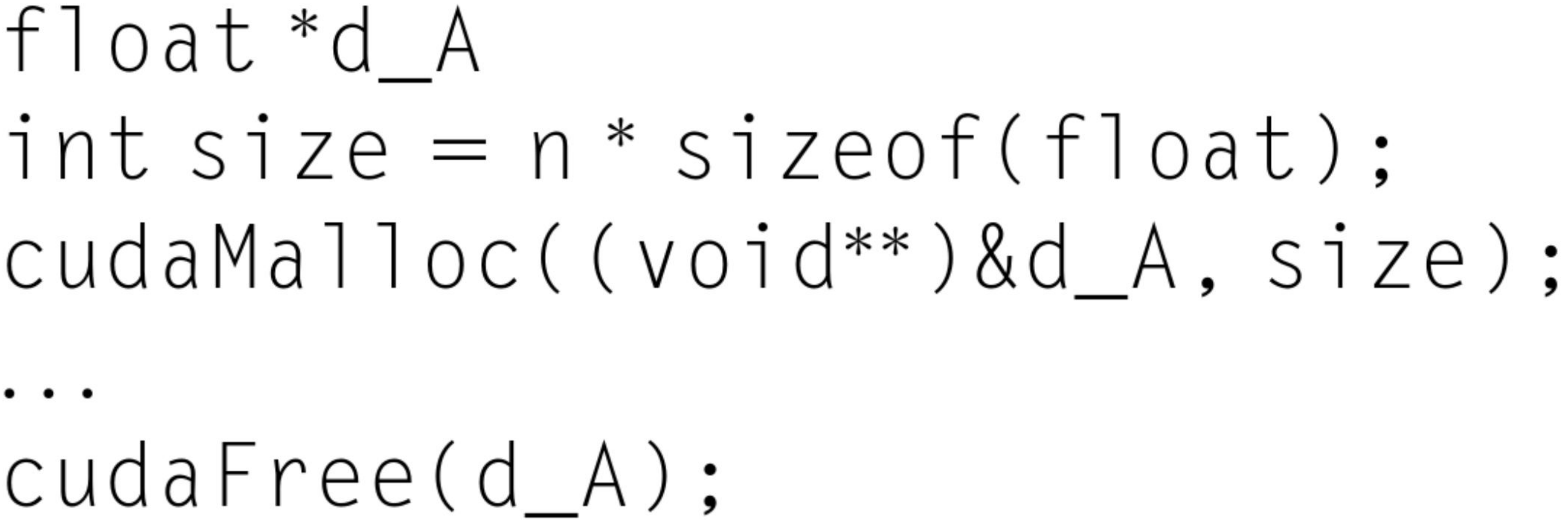
Host and Device Memory



cudaMalloc and cudaFree (1/2)

- cudaMalloc()
  - Allocates 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 memoryv
    - **Pointer** to freed object

cudaMalloc and cudaFree (2/2)



```
float *d_A
int size = n * sizeof(float);
cudaMalloc((void**)&d_A, size);
...
cudaFree(d_A);
```

cudaMemcpy

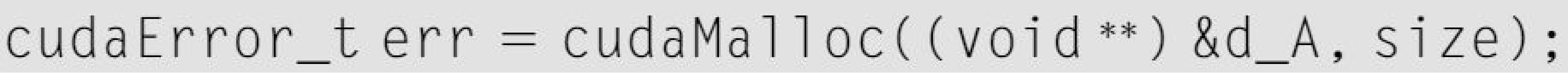
cudaMemcpy()

- memory data transfer
- Requires four parameters
  - Pointer to destination
  - Pointer to source
  - Number of bytes copied
  - Type/Direction of transfer

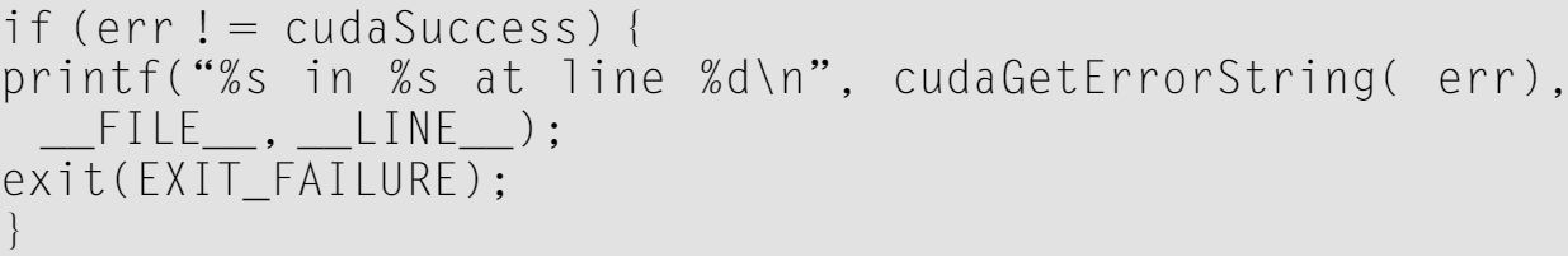
More Complete Version

```
void vecAdd(float* A, float* B, float* C, int n)
{
    int size = n * sizeof(float);
    float *d_A, *d_B, *d_C;

    cudaMalloc((void **) &d_A, size);
    cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice);
    cudaMalloc((void **) &B_d, size);
    cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice);

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

    // Kernel invocation code – to be shown later
    ...

    cudaMemcpy(C, d_C, size, cudaMemcpyDeviceToHost);

    // Free device memory for A, B, C
    cudaFree(d_Ad); cudaFree(d_B); cudaFree (d_C);
}
```

Error Handling

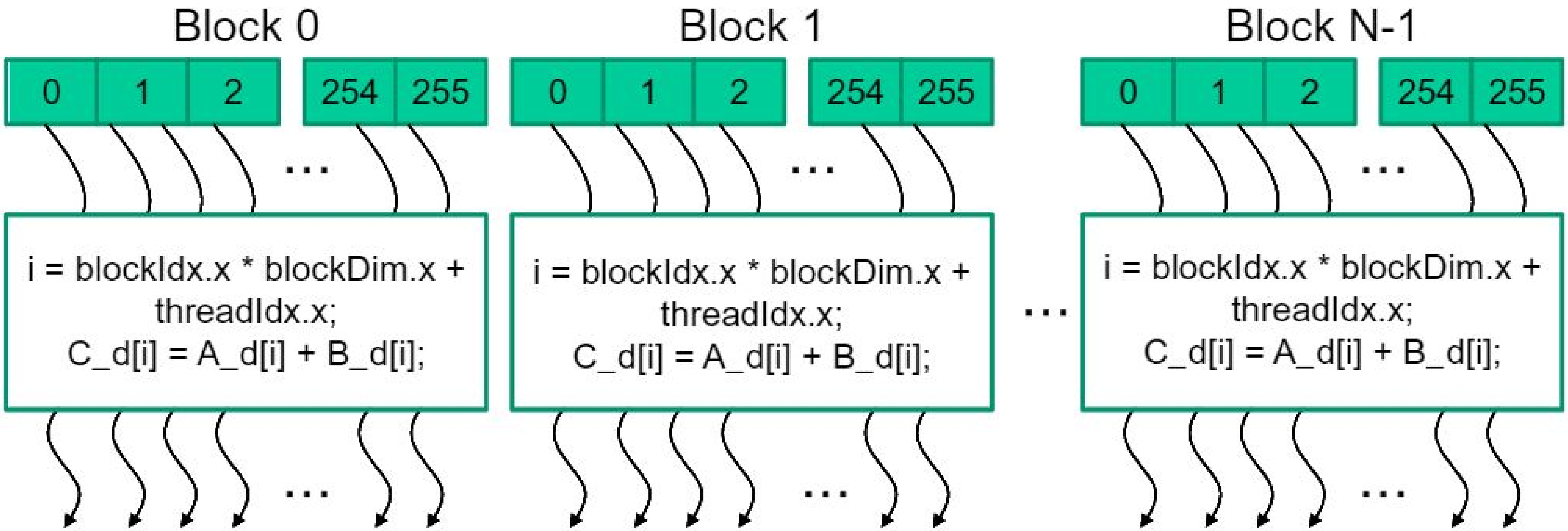


```
cudaError_t err = cudaMalloc((void **) &d_A, size);
```

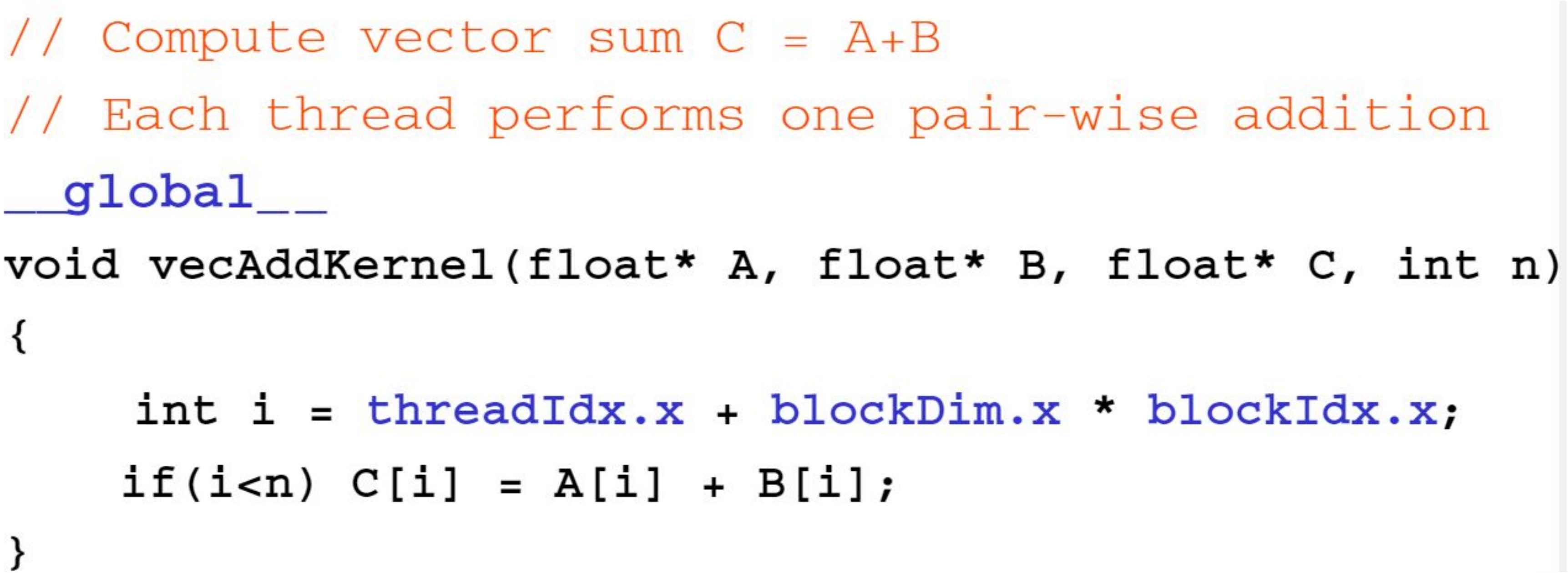


```
if (err != cudaSuccess) {
printf(“%s in %s at line %d\n”, cudaGetErrorString( err),
  __FILE__, __LINE__);
exit(EXIT_FAILURE);
}
```

Threads Organization


Block 0
0
1
2
254
255
...
i = blockIdx.x * blockDim.x + threadIdx.x;
C_d[i] = A_d[i] + B_d[i];
...
Block 1
0
1
2
254
255
...
i = blockIdx.x * blockDim.x + threadIdx.x;
C_d[i] = A_d[i] + B_d[i];
...
...
Block N-1
0
1
2
254
255
...
i = blockIdx.x * blockDim.x + threadIdx.x;
C_d[i] = A_d[i] + B_d[i];
...

Vector Addition Kernel



```
// 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];
}
```

Kernel Launch

```
int vectAdd(float* A, float* B, float* C, int n)
{
//  d_A, d_B, d_C allocations and copies omitted
// Run ceil(n/256) blocks of 256 threads each
    vecAddKernel<<<ceil(n/256.0), 256>>>(d_A, d_B, d_C, n);
}
```

CUDA C Keywords Function Declaration

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

Complete Function

```
void vecAdd(float* A, float* B, float* C, int n)
{
    int size = n * sizeof(float);
    float *d_A, *d_B, *d_C;

     cudaMalloc((void **) &d_A, size);
     cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice);
     cudaMalloc((void **) &B_d, size);
     cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice);

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

     vecAddKernel<<<ceil(n/2560), 256>>>(d_A, d_B, d_C, n);

     cudaMemcpy(C, d_C, size, cudaMemcpyDeviceToHost);
        // Free device memory for A, B, C
      cudaFree(d_Ad); cudaFree(d_B); cudaFree (d_C);
}
```

TODO (1/1)

- Make your own code to add vectors
- Try to measure times
- Use error handling - “Saves a lot of work”
- ENJOY :)

Bibliography (1/1)

- Programming Massively Parallel Processors

THANKS

john@sirius.utp.edu.co