Parallel Programming

- Parallel programming is like getting help from friends to clean up a big mess. Instead of one person doing all the work, everyone works together at the same time. Each person cleans a small part, and when everyone finishes, the whole place is clean faster. This way, the job is done quicker because many people are helping at once.

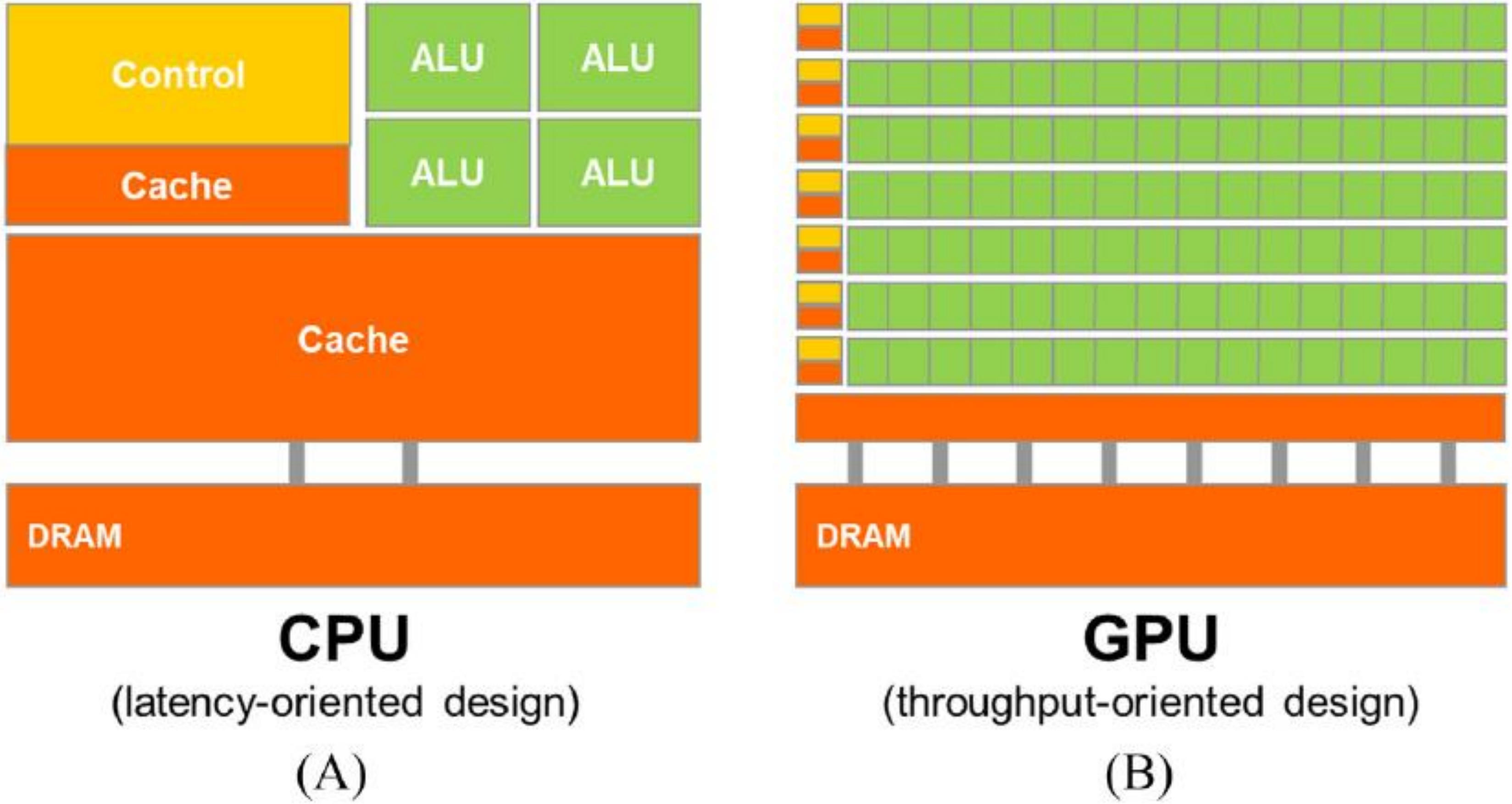
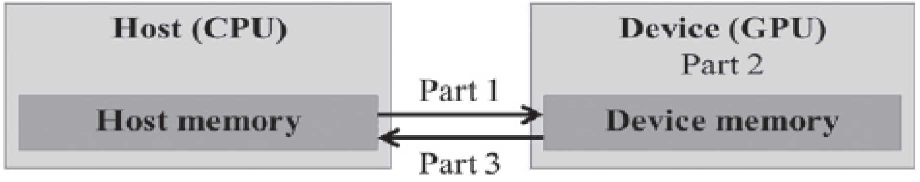


FIGURE 1.1

CPUs and GPUs have fundamentally different design philosophies: (A) CPU design is latency oriented; (B) GPU design is throughput-oriented.



```
void vecAdd(float* A, float* B, float* 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: Call kernel – to launch a grid of threads
    // to perform the actual vector addition
    ...

    // Part 3: Copy C from the device memory
    // Free device vectors
    ...
}
```

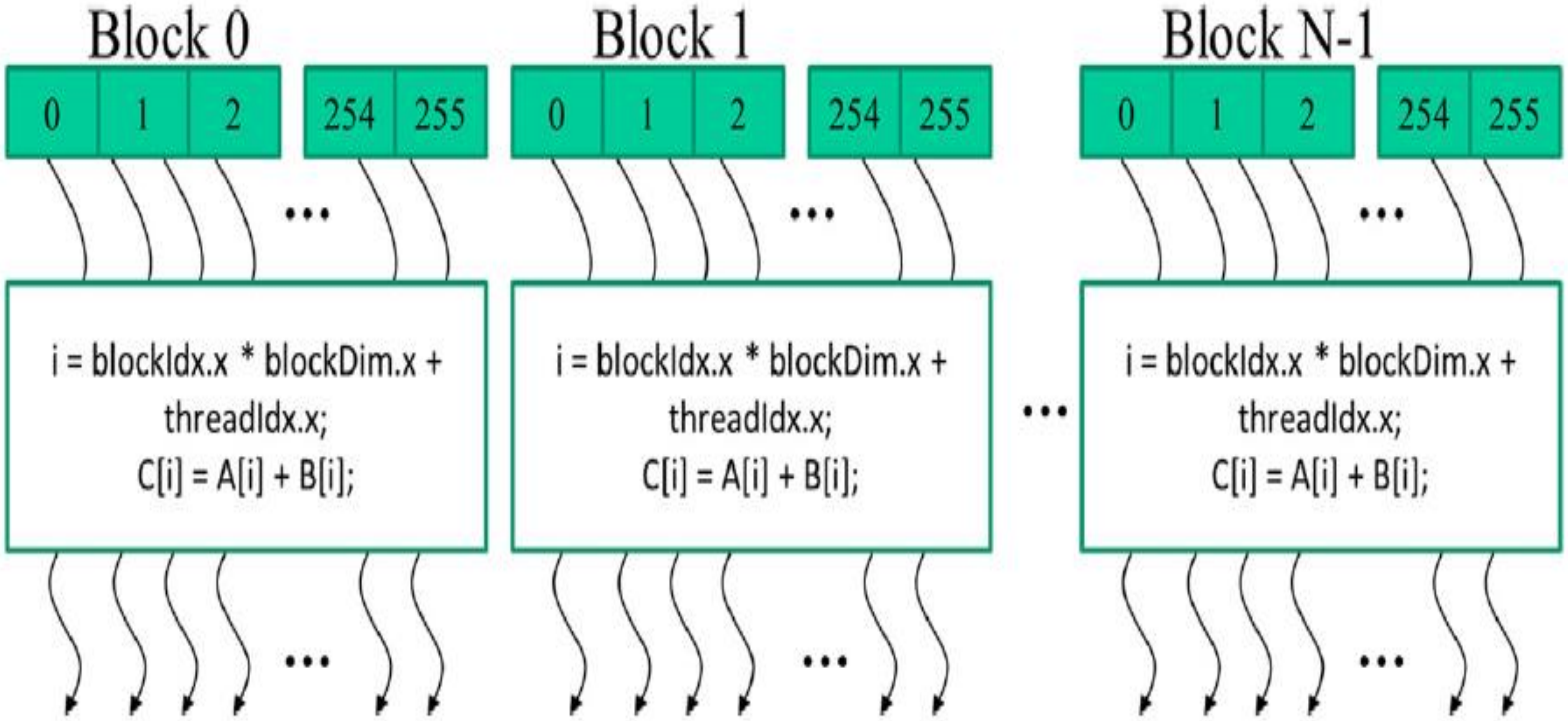
FIGURE 2.5

Outline of a revised vecAdd function that moves the work to a device.

```
void vecAdd(float* A_h, float* B_h, float* C_h, int n) {
    int size = n * sizeof(float);
    float *A_d, *B_d, *C_d;

    cudaMalloc((void **) &A_d, size);
    cudaMalloc((void **) &B_d, size);
    cudaMalloc((void **) &C_d, size);

    cudaMemcpy(A_d, A_h, size, cudaMemcpyHostToDevice);
    cudaMemcpy(B_d, B_h, size, cudaMemcpyHostToDevice);

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

    cudaMemcpy(C_h, C_d, size, cudaMemcpyDeviceToHost);

    cudaFree(A_d);
    cudaFree(B_d);
    cudaFree(C_d);
}
```

FIGURE 2.8

A more complete version of vecAdd().



```
i = blockIdx.x * blockDim.x +
threadIdx.x;
C[i] = A[i] + B[i];
```

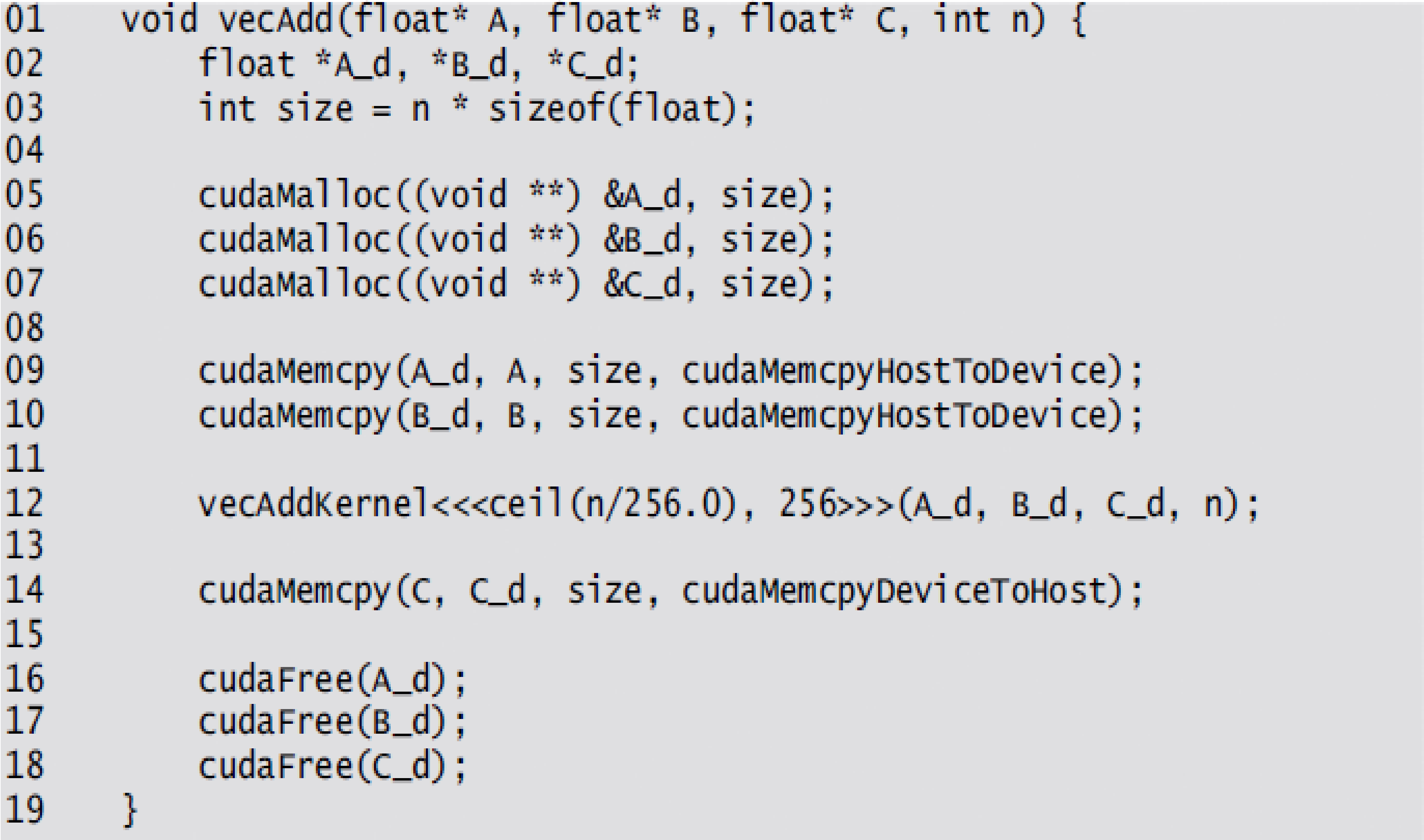
FIGURE 2.9

All threads in a grid execute the same kernel 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];
    }
}
```

FIGURE 2.10

A vector addition kernel function.



```
void vecAdd(float* A, float* B, float* C, int n) {
    float *A_d, *B_d, *C_d;
    int size = n * sizeof(float);

    cudaMalloc((void **) &A_d, size);
    cudaMalloc((void **) &B_d, size);
    cudaMalloc((void **) &C_d, size);

    cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);

    vecAddKernel<<<ceil(n/256.0), 256>>>(A_d, B_d, C_d, n);

    cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);

    cudaFree(A_d);
    cudaFree(B_d);
    cudaFree(C_d);
}
```