* Memory allocation on both CPU and GPU. Because, as have discussed before, GPU is accelerator, and it can not be act host machine. So therefore, the computation has to be initiated via CPU. That means, we need to fist initialize the data on the host, that is CPU. At the same time, we also need to initialize the memory allocation on the GPU. Because, we need to transfer the data from CPU to GPU.
* Allocating the CPU memory for a, b, and out vector

// Initialize the memory on the host  
float \*a, \*b, \*out;  
  
// Allocate host memory  
a = (float\*)malloc(sizeof(float) \* N);  
b = (float\*)malloc(sizeof(float) \* N);  
out = (float\*)malloc(sizeof(float) \* N);

* Allocating the GPU memory for d\_a, d\_b, and d\_out matrix

// Initialize the memory on the device  
float \*d\_a, \*d\_b, \*d\_out;  
  
// Allocate device memory  
cudaMalloc((void\*\*)&d\_a, sizeof(float) \* N);  
cudaMalloc((void\*\*)&d\_b, sizeof(float) \* N);  
cudaMalloc((void\*\*)&d\_out, sizeof(float) \* N);

* Now we need to fill the values for the array a and b.

// Initialize host arrays  
for(int i = 0; i < N; i++)  
 {  
 a[i] = 1.0f;  
 b[i] = 2.0f;  
 }

* Transfer initialized value from CPU to GPU

// Transfer data from host to device memory  
cudaMemcpy(d\_a, a, sizeof(float) \* N, cudaMemcpyHostToDevice);  
cudaMemcpy(d\_b, b, sizeof(float) \* N, cudaMemcpyHostToDevice);

* Creating a 2D thread block

// Thread organization   
dim3 dimGrid(1, 1, 1);   
dim3 dimBlock(16, 16, 1);

* Calling the kernel function

// execute the CUDA kernel function   
vector\_add<<<dimGrid, dimBlock>>>(d\_a, d\_b, d\_out, N);

* Vector addition kernel function call definition
* ??? “vector addition function call”
* === "Serial-version"  
   ```c  
   // CPU function that adds two vector   
   float \* Vector\_Add(float \*a, float \*b, float \*out, int n)   
   {  
   for(int i = 0; i < n; i ++)  
   {  
   out[i] = a[i] + b[i];  
   }  
   return out;  
   }  
   ```  
    
  === "CUDA-version"  
   ```c  
   // GPU function that adds two vectors   
   \_\_global\_\_ void vector\_add(float \*a, float \*b,   
   float \*out, int n)   
   {  
   int i = blockIdx.x \* blockDim.x \* blockDim.y +   
   threadIdx.y \* blockDim.x + threadIdx.x;   
   // Allow the threads only within the size of N  
   if(i < n)  
   {  
   out[i] = a[i] + b[i];  
   }  
    
   // Synchronice all the threads   
   \_\_syncthreads();  
   }  
   ```
* Copy back computed value from GPU to CPU

// Transfer data back to host memory  
cudaMemcpy(out, d\_out, sizeof(float) \* N, cudaMemcpyDeviceToHost);

* Deallocate the host and device memory

// Deallocate device memory  
cudaFree(d\_a);  
cudaFree(d\_b);  
cudaFree(d\_out);  
  
// Deallocate host memory  
free(a);   
free(b);   
free(out);

### Questions and Solutions

??? example “Examples: Vector Addition”

=== "Serial-version"  
  
 ```c   
 //-\*-C++-\*-  
 // Vector-addition.c  
   
 #include <stdio.h>  
 #include <stdlib.h>  
 #include <math.h>  
 #include <assert.h>  
 #include <time.h>  
   
 #define N 5120  
 #define MAX\_ERR 1e-6  
  
 // CPU function that adds two vector   
 float \* Vector\_Add(float \*a, float \*b, float \*out, int n)   
 {  
 for(int i = 0; i < n; i ++)  
 {  
 out[i] = a[i] + b[i];  
 }  
 return out;  
 }  
  
 int main()  
 {  
 // Initialize the memory on the host  
 float \*a, \*b, \*out;   
  
 // Allocate host memory  
 a = (float\*)malloc(sizeof(float) \* N);  
 b = (float\*)malloc(sizeof(float) \* N);  
 out = (float\*)malloc(sizeof(float) \* N);  
  
 // Initialize host arrays  
 for(int i = 0; i < N; i++)  
 {  
 a[i] = 1.0f;  
 b[i] = 2.0f;  
 }  
  
 // Start measuring time  
 clock\_t start = clock();  
  
 // Executing CPU funtion   
 Vector\_Add(a, b, out, N);  
  
 // Stop measuring time and calculate the elapsed time  
 clock\_t end = clock();  
 double elapsed = (double)(end - start)/CLOCKS\_PER\_SEC;  
   
 printf("Time measured: %.3f seconds.\n", elapsed);  
  
 // Verification  
 for(int i = 0; i < N; i++)  
 {  
 assert(fabs(out[i] - a[i] - b[i]) < MAX\_ERR);  
 }  
  
 printf("out[0] = %f\n", out[0]);  
 printf("PASSED\n");  
  
 // Deallocate host memory  
 free(a);   
 free(b);   
 free(out);  
  
 return 0;  
 }  
 ```  
  
=== "CUDA-template"  
  
 ```c   
 //-\*-C++-\*-  
 // Vector-addition-template.cu  
  
 #include <stdio.h>  
 #include <stdlib.h>  
 #include <math.h>  
 #include <assert.h>  
 #include <time.h>  
 #include <cuda.h>  
  
 #define N 5120  
 #define MAX\_ERR 1e-6  
  
  
 // GPU function that adds two vectors   
 \_\_global\_\_ void vector\_add(float \*a, float \*b,   
 float \*out, int n)   
 {   
 // allign your thread id indexes   
 int i = ........  
   
 // Allow the threads only within the size of N  
 if------  
 {  
 out[i] = a[i] + b[i];  
 }  
  
 // Synchronice all the threads   
  
 }  
  
 int main()  
 {  
 // Initialize the memory on the host  
 float \*a, \*b, \*out;  
  
 // Allocate host memory  
 a = (float\*)......  
   
 // Initialize the memory on the device  
 float \*d\_a, \*d\_b, \*d\_out;  
  
 // Allocate device memory  
 cudaMalloc((void\*\*)&d\_a,......  
  
 // Initialize host arrays  
 for(int i = 0; i < N; i++)  
 {  
 a[i] = ....  
 b[i] = ....  
 }  
  
 // Transfer data from host to device memory  
 cudaMemcpy.....  
   
 // Thread organization   
 dim3 dimGrid....   
 dim3 dimBlock....  
  
 // execute the CUDA kernel function   
 vector\_add<<< >>>....  
  
 // Transfer data back to host memory  
 cudaMemcpy....  
   
 // Verification  
 for(int i = 0; i < N; i++)  
 {  
 assert(fabs(out[i] - a[i] - b[i]) < MAX\_ERR);  
 }  
  
 printf("out[0] = %f\n", out[0]);  
 printf("PASSED\n");  
  
 // Deallocate device memory  
 cudaFree...  
  
  
 // Deallocate host memory  
 free..  
  
 return 0;  
 }  
 ```  
   
=== "CUDA-version"  
  
 ```c   
 //-\*-C++-\*-  
 // Vector-addition.cu  
   
 #include <stdio.h>  
 #include <stdlib.h>  
 #include <math.h>  
 #include <assert.h>  
 #include <time.h>  
 #include <cuda.h>  
  
 #define N 5120  
 #define MAX\_ERR 1e-6  
  
  
 // GPU function that adds two vectors   
 \_\_global\_\_ void vector\_add(float \*a, float \*b,   
 float \*out, int n)   
 {  
   
 int i = blockIdx.x \* blockDim.x \* blockDim.y +   
 threadIdx.y \* blockDim.x + threadIdx.x;   
 // Allow the threads only within the size of N  
 if(i < n)  
 {  
 out[i] = a[i] + b[i];  
 }  
  
 // Synchronice all the threads   
 \_\_syncthreads();  
 }  
  
 int main()  
 {  
 // Initialize the memory on the host  
 float \*a, \*b, \*out;  
  
 // Allocate host memory  
 a = (float\*)malloc(sizeof(float) \* N);  
 b = (float\*)malloc(sizeof(float) \* N);  
 out = (float\*)malloc(sizeof(float) \* N);  
   
 // Initialize the memory on the device  
 float \*d\_a, \*d\_b, \*d\_out;  
  
 // Allocate device memory  
 cudaMalloc((void\*\*)&d\_a, sizeof(float) \* N);  
 cudaMalloc((void\*\*)&d\_b, sizeof(float) \* N);  
 cudaMalloc((void\*\*)&d\_out, sizeof(float) \* N);   
  
 // Initialize host arrays  
 for(int i = 0; i < N; i++)  
 {  
 a[i] = 1.0f;  
 b[i] = 2.0f;  
 }  
  
 // Transfer data from host to device memory  
 cudaMemcpy(d\_a, a, sizeof(float) \* N, cudaMemcpyHostToDevice);  
 cudaMemcpy(d\_b, b, sizeof(float) \* N, cudaMemcpyHostToDevice);  
  
 // Thread organization   
 dim3 dimGrid(ceil(N/32), ceil(N/32), 1);  
 dim3 dimBlock(32, 32, 1);   
  
 // execute the CUDA kernel function   
 vector\_add<<<dimGrid, dimBlock>>>(d\_a, d\_b, d\_out, N);  
  
 // Transfer data back to host memory  
 cudaMemcpy(out, d\_out, sizeof(float) \* N, cudaMemcpyDeviceToHost);  
   
 // Verification  
 for(int i = 0; i < N; i++)  
 {  
 assert(fabs(out[i] - a[i] - b[i]) < MAX\_ERR);  
 }  
  
 printf("out[0] = %f\n", out[0]);  
 printf("PASSED\n");  
  
 // Deallocate device memory  
 cudaFree(d\_a);  
 cudaFree(d\_b);  
 cudaFree(d\_out);  
  
 // Deallocate host memory  
 free(a);   
 free(b);   
 free(out);  
  
 return 0;  
 }  
 ```

??? “Compilation and Output”

=== "Serial-version"  
 ```c  
 // compilation  
 $ gcc Vector-addition.c -o Vector-Addition-CPU  
   
 // execution   
 $ ./Vector-Addition-CPU  
   
 // output  
 $ ./Vector-addition-CPU   
 out[0] = 3.000000  
 PASSED  
 ```  
   
=== "CUDA-version"  
 ```c  
 // compilation  
 $ nvcc -arch=compute\_70 Vector-addition.cu -o Vector-Addition-GPU  
   
 // execution  
 $ ./Vector-Addition-GPU  
   
 // output  
 $ ./Vector-addition-GPU  
 out[0] = 3.000000  
 PASSED  
 ```

??? Question “Questions”

- What happens if you remove the \*\*`\_\_syncthreads();`\*\* from the \*\*`\_\_global\_\_ void vector\_add(float \*a, float \*b,   
 float \*out, int n)`\*\* function.  
- Can you remove the if condition \*\*`if(i < n)`\*\* from the \*\*`\_\_global\_\_ void vector\_add(float \*a, float \*b,  
 float \*out, int n)`\*\* function. If so how can you do that?  
- Here we do not use the \*\*`cudaDeviceSynchronize()`\*\* in the main application, can you figure out why we  
 do not need to use it.   
- Can you create a different kinds of threads block for larger number of array?