### [^^Unified Memory Programming^^](https://docs.nvidia.com/cuda/profiler-users-guide/index.html#migrating-to-nsight-tools-from-visual-profiler-and-nvprof)

* Unified memory simplifies the explicit data movement from host to device by programmers.
* Just one memory allocation is enough [cudaMallocManaged()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html#group__CUDART__MEMORY_1gd228014f19cc0975ebe3e0dd2af6dd1b)

|  |  |
| --- | --- |
| Without unified memory | With unified memory |
| Allocate the host memory | ~~Allocate the host memory~~ |
| Allocate the device memory | Allocate the device memory |
| Initialize the host value | Initialize the host value |
| Transfer the host value to the device memory location | ~~Transfer the host value to the device memory location~~ |
| Do the computation using the CUDA kernel | Do the computation using the CUDA kernel |
| Transfer the data from the device to host | ~~Transfer the data from the device to host~~ |
| Free device memory | Free device memory |
| Free host memory | ~~Free host memory~~ |

### Questions and Solutions

??? example “Examples: Unified Memory - Vector Addition”

=== "Without Unified Memory"  
  
 ```c  
 //-\*-C++-\*-  
 // Without-unified-memory.cu  
   
 #include <stdio.h>  
 #include <stdlib.h>  
 #include <math.h>  
 #include <assert.h>  
 #include <time.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);  
 c = (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;  
 }  
 ```  
  
=== "With Unified Memory - template"  
  
 ```c  
 //-\*-C++-\*-  
   
 #include <stdio.h>  
 #include <stdlib.h>  
 #include <math.h>  
 #include <assert.h>  
 #include <time.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);  
 c = (float\*)malloc(sizeof(float) \* N);  
 \*/  
  
 // Initialize the memory on the device  
 float \*d\_a, \*d\_b, \*d\_out;  
  
 // Allocate device(unified) memory  
 cudaMallocManaged......  
  
 // Initialize host arrays  
 for(int i = 0; i < N; i++)  
 {  
 d\_a[i] = ...  
 d\_b[i] = ...  
 }  
  
 /\*  
 // 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...   
 dim3 dimBlock...  
  
 // execute the CUDA kernel function   
 vector\_add<<<dimGrid, dimBlock>>>(d\_a, d\_b, d\_out, N);  
   
 // synchronize if needed  
 ......  
   
 /\*  
 // Transfer data back to host memory  
 cudaMemcpy(out, d\_out, sizeof(float) \* N, cudaMemcpyDeviceToHost);  
 \*/  
  
 // Verification  
 for(int i = 0; i < N; i++)  
 {  
 assert(fabs(d\_out[i] - d\_a[i] - d\_b[i]) < MAX\_ERR);  
 }  
  
 printf("out[0] = %f\n", d\_out[0]);  
 printf("PASSED\n");  
  
 // Deallocate device(unified) memory  
 cudaFree...  
   
  
 /\*  
 // Deallocate host memory  
 free(a);   
 free(b);   
 free(out);  
 \*/  
  
 return 0;  
 }  
 ```  
   
=== "With Unified Memory-version"  
  
 ```c  
 //-\*-C++-\*-  
 // With-unified-memory.cu  
   
 #include <stdio.h>  
 #include <stdlib.h>  
 #include <math.h>  
 #include <assert.h>  
 #include <time.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);  
 c = (float\*)malloc(sizeof(float) \* N);  
 \*/  
  
 // Initialize the memory on the device  
 float \*d\_a, \*d\_b, \*d\_out;  
  
 // Allocate device memory  
 cudaMallocManaged(&d\_a, sizeof(float) \* N);  
 cudaMallocManaged(&d\_b, sizeof(float) \* N);  
 cudaMallocManaged(&d\_out, sizeof(float) \* N);   
  
 // Initialize host arrays  
 for(int i = 0; i < N; i++)  
 {  
 d\_a[i] = 1.0f;  
 d\_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);  
 cudaDeviceSynchronize();  
 /\*  
 // Transfer data back to host memory  
 cudaMemcpy(out, d\_out, sizeof(float) \* N, cudaMemcpyDeviceToHost);  
 \*/  
  
 // Verification  
 for(int i = 0; i < N; i++)  
 {  
 assert(fabs(d\_out[i] - d\_a[i] - d\_b[i]) < MAX\_ERR);  
 }  
  
 printf("out[0] = %f\n", d\_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”

=== "Without-unified-memory.cu"  
 ```c  
 // compilation  
 $ nvcc -arch=compute\_70 Without-unified-memory.cu -o Without-Unified-Memory  
   
 // execution   
 $ ./Without-Unified-Memory  
   
 // output  
 $ ./Without-Unified-Memory  
 out[0] = 3.000000  
 PASSED  
 ```  
   
=== "With-unified-memory"  
 ```c  
 // compilation  
 $ nvcc -arch=compute\_70 With-unified-memory.cu -o With-Unified-Memory  
   
 // execution  
 $ ./With-Unified-Memory  
  
 // output  
 $ ./With-Unified-Memory   
 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?