```
%%writefile add.cu

#include <iostream>
#include <cuda_runtime.h>  // Provides necessary functions and macros to work with CUDA.

using namespace std;

__global__ void addVectors(int* A, int* B, int* C, int n) //__global__: Specifies that this function is a CUDA kernel, meaning it runs on
{
    int i = blockIdx.x * blockDim.x + threadIdx.x; // blockIdx.x: The index of the block within the grid. blockDim.x: The number of threa
    if (i < n) // Ensures that threads do not access memory beyond the allocated array.
    {
        C[i] = A[i] + B[i]; // Adds corresponding elements from vectors A and B, storing the result in C.
    }
}

int main()
{
    int n = 1000000; // The size of the vectors (one million elements).
    int* A, * B, * C; // Pointers for the host (CPU) memory.
    int size = n * sizeof(int); // The memory size required for each vector in bytes.

    // Allocate memory on the host
    cudaMallocHost(&A, size); //  Allocates pinned (page-locked) memory on the host. This improves the speed of memory transfer between h
    cudaMallocHost(&B, size);
    cudaMallocHost(&C, size);

    // Initialize the vectors
    for (int i = 0; i < n; i++)
    {
        A[i] = i; // Fills A with values [0, 1, 2, ..., n-1].
        B[i] = i * 2;  // Fills B with values [0, 2, 4, ..., 2*(n-1)].
    }
    // Allocate memory on the device
    int* dev_A, * dev_B, * dev_C;
    cudaMalloc(&dev_A, size); // cudaMalloc(&dev_A, size): Allocates size bytes of GPU memory for A.
    cudaMalloc(&dev_B, size); // cudaMalloc(&dev_B, size): Allocates size bytes of GPU memory for B.
    cudaMalloc(&dev_C, size); // cudaMalloc(&dev_C, size): Allocates size bytes of GPU memory for C.

    // Copy data from host to device
    cudaMemcpy(dev_A, A, size, cudaMemcpyHostToDevice); // cudaMemcpy(dest, src, size, cudaMemcpyHostToDevice): Copies data from host mem
    cudaMemcpy(dev_B, B, size, cudaMemcpyHostToDevice);

    // Launch the kernel
    int blockSize = 256; // Defines 256 threads per block.
    int numBlocks = (n + blockSize - 1) / blockSize; // Ensures that all elements are covered
    addVectors<<<numBlocks, blockSize>>>(dev_A, dev_B, dev_C, n); // This launches the kernel with numBlocks blocks and blockSize threads

    // Copy data from device to host
    cudaMemcpy(C, dev_C, size, cudaMemcpyDeviceToHost); // Copies the computed results from device memory (dev_C) to host memory (C).

    // Print the results
    for (int i = 0; i < 10; i++)
    {
        cout << C[i] << " "; // Prints the first 10 elements of C to verify the computation.
    }
    cout << endl;

    // Free memory
    cudaFree(dev_A); // releases memory on the GPU.
    cudaFree(dev_B);
    cudaFree(dev_C);
    cudaFreeHost(A); //  releases pinned memory on the CPU.
    cudaFreeHost(B);
    cudaFreeHost(C);

    return 0;
}
```

```
Overwriting add.cu
```

```
!rm -rf /usr/local/cuda           # Removes any previous CUDA installations (only needed in certain environments).
!ln -s  /usr/local/cuda-12.5 /usr/local/cuda        # Links to CUDA 12.2.
!nvcc -arch=sm_75 add.cu -o add         # Compiles the CUDA program (nvcc is the CUDA compiler).
```

```
!./add // Each element of C is the sum of the corresponding elements of A and B:C[0] = 0 + 0 = 0    C[1] = 1 + 2 = 3   C[2] = 2 + 4 = 6
```

```
0 3 6 9 12 15 18 21 24 27
```