

```

#include <iostream>
#include <cuda_runtime.h>

using namespace std;

__global__ void addVectors(int* A, int* B, int* C, int n)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
    {
        C[i] = A[i] + B[i];
    }
}

int main()
{
    int n = 1000000;
    int* A, * B, * C;
    int size = n * sizeof(int);

    // Allocate memory on the host
    cudaMallocHost(&A, size);
    cudaMallocHost(&B, size);
    cudaMallocHost(&C, size);

    // Initialize the vectors
    for (int i = 0; i < n; i++)
    {
        A[i] = i;
        B[i] = i * 2;
    }
    // Allocate memory on the device
    int* dev_A, * dev_B, * dev_C;
    cudaMalloc(&dev_A, size);
    cudaMalloc(&dev_B, size);
    cudaMalloc(&dev_C, size);

    // Copy data from host to device
    cudaMemcpy(dev_A, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_B, B, size, cudaMemcpyHostToDevice);

    // Launch the kernel
    int blockSize = 256;
    int numBlocks = (n + blockSize - 1) / blockSize;
    addVectors<<<numBlocks, blockSize>>>>(dev_A, dev_B, dev_C, n);

    // Copy data from device to host
    cudaMemcpy(C, dev_C, size, cudaMemcpyDeviceToHost);

    // Print the results

```

```
for (int i = 0; i < 10; i++)
{
    cout << C[i] << " ";
}
cout << endl;

// Free memory
cudaFree(dev_A);
cudaFree(dev_B);
cudaFree(dev_C);
cudaFreeHost(A);
cudaFreeHost(B);
cudaFreeHost(C);

return 0;
}
```