

```

// includes, system
#include <stdio.h>
#include <assert.h>

// Simple utility function to check for CUDA runtime errors
void checkCUDAError(const char* msg);

// Part 2 of 2: implement the kernel
__global__ void reverseArrayBlock( )
{
}

////////////////////////////////////
////////
// Program main
////////////////////////////////////
////////
int main( int argc, char** argv)
{
    // pointer for host memory and size
    int *h_a;
    int dimA = 256 * 1024; // 256K elements (1MB total)

    // pointer for device memory
    int *d_b, *d_a;

    // define grid and block size
    int numThreadsPerBlock = 256;

    // Part 1 of 2: compute number of blocks needed based on array size
and desired block size
    int numBlocks = ;

    // allocate host and device memory
    size_t memSize = numBlocks * numThreadsPerBlock * sizeof(int);
    h_a = (int *) malloc(memSize);
    cudaMalloc( (void **) &d_a, memSize );
    cudaMalloc( (void **) &d_b, memSize );

    // Initialize input array on host
    for (int i = 0; i < dimA; ++i)
    {
        h_a[i] = i;
    }

    // Copy host array to device array
    cudaMemcpy( d_a, h_a, memSize, cudaMemcpyHostToDevice );

    // launch kernel
    dim3 dimGrid(numBlocks);
    dim3 dimBlock(numThreadsPerBlock);
    reverseArrayBlock<<< dimGrid, dimBlock >>>( d_b, d_a );

    // block until the device has completed
    cudaThreadSynchronize();

    // check if kernel execution generated an error
    // Check for any CUDA errors
    checkCUDAError("kernel invocation");
}

```

```

// device to host copy
cudaMemcpy( h_a, d_b, memSize, cudaMemcpyDeviceToHost );

// Check for any CUDA errors
checkCUDAError("memcpy");

// verify the data returned to the host is correct
for (int i = 0; i < dimA; i++)
{
    assert(h_a[i] == dimA - 1 - i );
}

// free device memory
cudaFree(d_a);
cudaFree(d_b);

// free host memory
free(h_a);

// If the program makes it this far, then the results are correct and
// there are no run-time errors.  Good work!
printf("Correct!\n");

return 0;
}

void checkCUDAError(const char *msg)
{
    cudaError_t err = cudaGetLastError();
    if( cudaSuccess != err)
    {
        fprintf(stderr, "Cuda error: %s: %s.\n", msg, cudaGetErrorString(
err) );
        exit(EXIT_FAILURE);
    }
}

```