Assignment 6 – Optimised Parallel Reduction

Let us consider the dot product operation of two vectors A and B, each of dimension N. Implement a CUDA program which uses the 1D CUDA kernel dotproduct() launched with parameters <<<m,k>>> such that

i) m * k = N,

ii) each block of k threads of dotproduct() is responsible for computing the pairwise products of k successive elements in A and B and finally returning the partial sum of those k products. Use shared memory while computing partial sums.

Implement and call the reduction kernel repeatedly on these k partial sums until you are left with p partial sums where p <1024. For summing up elements less than 1024, it makes sense to execute reduction on the CPU, since the GPU is highly underutilized in this case. Once the partial sums have been calculated, implement a sequential CPU function that will reduce these elements to yield the final scalar value of the dot product.

Note for the reduction GPU kernel, implement a version that -

i) avoids branch divergence,
ii) avoids shared memory bank conflicts,
iii) performs first add during load and
iv) uses loop unrolling.

Implement a complete CUDA program which takes as input

i) the number of test cases
ii) for each test case the value of N and
iii) 2 lines of N space separated values representing the 2 vectors A and B.

Assume that N is a power of 2 and is very large (of the order in the range 2^18 -- 2^23.). Your task would be to generate random floating point arrays of this order and test your implementation while adhering to the input format discussed above. The code should be general enough to handle large matrices. Your program should print the required dot product i.e. a scalar value for each test case.