\_\_global\_\_ void nw\_1(unsigned char\* sequence1\_d, unsigned char\* sequence2\_d, int\* scores\_d) {

    unsigned int segment = SEQUENCE\_LENGTH\*blockIdx.x;

    unsigned int tidx = threadIdx.x;

    int row, col, top, left, topleft, insertion, deletion, match, max;

**Initialize containers to store the sequence pair in shared memory**

    \_\_shared\_\_ unsigned char sequence1\_s[SEQUENCE\_LENGTH];

    \_\_shared\_\_ unsigned char sequence2\_s[SEQUENCE\_LENGTH];

**Initialize the three buffers that we need for storing the previous values across the anti-diagonal**

    \_\_shared\_\_ int buffer1\_s[SEQUENCE\_LENGTH + 1];

    \_\_shared\_\_ int buffer2\_s[SEQUENCE\_LENGTH + 1];

    \_\_shared\_\_ int buffer3\_s[SEQUENCE\_LENGTH + 1];

**Initialize pointers to the above buffers**

    int \* buffer1 = buffer1\_s;

    int \* buffer2 = buffer2\_s;

    int \* buffer3 = buffer3\_s;

    sequence1\_s[tidx] = sequence1\_d[segment + tidx];

    sequence2\_s[tidx] = sequence2\_d[segment + tidx];

**Initializing the entries at the top leftmost entry in the “matrix”**

    if (tidx == 0) {

        buffer1[tidx] = 0;

        buffer2[tidx] = INSERTION;

        buffer2[tidx+1] = DELETION;

    }

    \_\_syncthreads();

    for(int i=0; i<SEQUENCE\_LENGTH-1; ++i)

    {

**If the element is at the boundaries, calculate based on the boundary rules**

        if (tidx == 0) {

            buffer3[tidx] = (i+2)\*INSERTION;

            buffer3[i+2] = (i+2)\*DELETION;

        }

        \_\_syncthreads();

        if(tidx <= i)

        {

            row = i-tidx;

            col = tidx;

**access the elements stored in the iteration one step before**

            top = buffer2[tidx+1];

            left = buffer2[tidx];

**access the element stored in the iteration two steps before**

            topleft = buffer1[tidx];

            insertion = top + INSERTION;

            deletion = left + DELETION;

            match = topleft + ((sequence1\_s[col] == sequence2\_s[row])?MATCH:MISMATCH);

            max = (insertion > deletion)?insertion:deletion;

            max = (match > max)?match:max;

**store the element computed for the subsequent steps**

            buffer3[tidx+1] = max;

        }

        \_\_syncthreads();

**SWAP THE BUFFERS SUCH THAT THE CURRENT BECOMES THE PREVIOUS (2), THE PREVIOUS BECOMES THE SECOND PREVIOUS (1), AND THE CURRENT IS SET TO (1) BUT WILL BE UPDATED SUBSEQUENTLY**

        int \* temp = buffer1;

        buffer1 = buffer2;

        buffer2 = buffer3;

        buffer3 = temp;

    }

    row = SEQUENCE\_LENGTH - 1 - tidx;

    col = tidx;

    top  = buffer2[tidx+1];

    left = buffer2[tidx];

    topleft = buffer1[tidx];

    insertion = top + INSERTION;

    deletion = left + DELETION;

    match = topleft + ((sequence1\_s[col] == sequence2\_s[row])?MATCH:MISMATCH);

    max = (insertion > deletion)?insertion:deletion;

    max = (match > max)?match:max;

    buffer3[tidx] = max;

    \_\_syncthreads();

    int \* temp = buffer1;

    buffer1 = buffer2;

    buffer2 = buffer3;

    buffer3 = temp;

**Doing the second half of the matrix**

    for(int i=SEQUENCE\_LENGTH-1; i>0; --i)

    {

**Same steps as above.**

        if(tidx < i)

        {

            row = SEQUENCE\_LENGTH - tidx - 1;

            col = SEQUENCE\_LENGTH + tidx - i;

            top  = buffer2[tidx+1];

            left = buffer2[tidx];

            topleft = buffer1[tidx+1];

            insertion = top + INSERTION;

            deletion = left + DELETION;

            match = topleft + ((sequence1\_s[col] == sequence2\_s[row])?MATCH:MISMATCH);

            max = (insertion > deletion)?insertion:deletion;

            max = (match > max)?match:max;

            buffer3[tidx] = max;

        }

        \_\_syncthreads();

        int \* temp = buffer1;

        buffer1 = buffer2;

        buffer2 = buffer3;

        buffer3 = temp;

    }

    if(tidx == 0)

    {

        scores\_d[blockIdx.x] = buffer2[tidx];

    }

}

void nw\_gpu1(unsigned char\* sequence1\_d, unsigned char\* sequence2\_d, int\* scores\_d, unsigned int numSequences) {

    assert(SEQUENCE\_LENGTH <= 1024); // You can assume the sequence length is not more than 1024

    unsigned int numThreadsPerBlock = SEQUENCE\_LENGTH;

    unsigned int numBlocks = numSequences;

    nw\_1 <<<numBlocks, numThreadsPerBlock>>> (sequence1\_d, sequence2\_d, scores\_d);

}