Convolution
Convolution – Outline

• To learn convolution, an important parallel computation pattern
  – Widely used in signal, image and video processing
  – Foundational to stencil computation used in many science and engineering applications
Convolution – Outline

• To learn convolution, an important parallel computation pattern
  – Widely used in signal, image and video processing
  – Foundational to stencil computation used in many science and engineering applications

• To learn important techniques for
  – *Tiling* data for more intricate access patterns
  – Taking advantage of cache memories
Convolution Applications

• Used as a **Filter** that transforms signals and pixels into more desirable values.
Convolution Applications

• Used as a **Filter** that transforms signals and pixels into more desirable values.
  – Some filters smooth out the signal values so that one can see the big-picture trend
Convolution Applications

• Used as a **Filter** that transforms signals and pixels into more desirable values.
  
  – Some filters smooth out the signal values so that one can see the big-picture trend
  
  – Gaussian filters can be used to sharpen boundaries and edges of objects in images.
Convolution

- An array operation where each output data element is a weighted sum of a collection of neighboring input elements
Convolution Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

MW = 5
Convolution Example

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\[
\begin{array}{cccc}
3 & 4 & 5 & 4 & 3 \\
\end{array}
\]
Convolution Example

Convolution Example

Mask size is usually an odd number of elements for symmetry (5 in this example)
Convolution Applications

- An array operation where each output data element is a weighted sum of a collection of neighboring input elements

- Input mask array: *convolution kernel* or *convolution mask*
Convolution Applications

- An array operation where each output data element is a weighted sum of a collection of neighboring input elements.
- Input mask array: *convolution kernel* or *convolution mask*.
- Value pattern of the mask array elements defines the type of filtering done.
- Eg. Image blur.
Convolution – Boundary Condition

Filled in

\[ \text{N} = [0, 1, 2, 3, 4, 5, 6, 7] \]

\[ \text{M} = [3, 4, 5, 4, 3] \]
Convolution – Boundary Condition
Convolution – Boundary Condition

Filled in

\[ \begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\end{array} \]

\[ \begin{array}{cccccc}
3 & 4 & 5 & 4 & 3 \\
\end{array} \]

\[ \begin{array}{cccccc}
38 & & & & & \\
\end{array} \]
Convolution – Boundary Condition

Calculation of output elements near the boundaries (beginning and end) of the input array need to deal with “ghost” elements.

Filled in (ghost)
Convolution – Boundary Condition

Calculation of output elements near the boundaries (beginning and end) of the input array need to deal with “ghost” elements.

Different policies (0, replicates of boundary values, etc.)

Filled in
1D Convolution Kernel with Boundary Condition Handling

```c
__global__
void convolution_1D_basic_kernel(float *N, float *M, float *P,
                                  int Mask_Width, int Width)
{
    # Which element in the output array to update? i

    # Corresponding starting point in the input array
    # wrt mask width

    # For each element corresponding to the mask,
    # accumulate inner product

    # Update the element in the output array.
}
```
1D Convolution Kernel

```c
__global__ void convolution_1D_basic_kernel(float *N, float *M, float *P, int Mask_Width, int Width)
{
    # Which element in the output array to update?
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    # Corresponding starting point in the input array wrt mask width
    # For each element corresponding to the mask,
    # accumulate inner product

    # Update the element in the output array.
}
```
1D Convolution Kernel

```c
__global__ void convolution_1D_basic_kernel(float *N, float *M, float *P, int Mask_Width, int Width)
{
    # Which element in the output array to update?
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    # Corresponding starting point in the input array wrt mask width
    int N_start_point = i - (Mask_Width/2);

    # For each element corresponding to the mask, accumulate inner product
    # Update the element in the output array.
}
```

__global__ void
convolution_1D_basic_kernel(float *N, float *M, float *P, int
Mask_Width, int Width)
{
    # Which element in the output array to update?
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    # Corresponding starting point in the input array
    # wrt mask width
    int N_start_point = i - (Mask_Width/2);

    #Foreach element corresponding to the mask, accumulate inner product
    for (int j = 0; j < Mask_Width; j++) {
        if (N_start_point + j >= 0 && N_start_point + j < Width) {
            Pvalue += N[N_start_point + j]*M[j];
        }
    }

    # Update the element in the output array.
}
1D Convolution Kernel

```c
__global__ void
convolution_1D_basic_kernel(float *N, float *M, float *P, int Mask_Width, int Width)
{
    # Which element in the output array to update?
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    # Corresponding starting point in the input array
    # wrt mask width
    int N_start_point = i - (Mask_Width/2);

    #Foreach element corresponding to the mask, accumulate inner product
    for (int j = 0; j < Mask_Width; j++) {
        if (N_start_point + j >= 0 && N_start_point + j < Width) {
            Pvalue += N[N_start_point + j]*M[j];
        }
    }

    # Update the element in the output array.
    P[i] = Pvalue;
}
```
2D Convolution Example

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M
2D Convolution Example
2D Convolution Example

N

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
2 & 3 & 4 & 5 & 6 \\
3 & 4 & 5 & 6 & 7 \\
4 & 5 & 6 & 7 & 8 \\
5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

P

\[
\begin{array}{cccccc}
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0 \\
\end{array}
\]

M

\[
\begin{array}{cccc}
0 & 4 & 0 \\
4 & 5 & 6 \\
0 & 6 & 0 \\
\end{array}
\]

25
2D Convolution Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 2 \\
0 & 2 & 0 \\
\end{array}
\]
2D Convolution Example

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 & 5 \\
2 & 3 & 4 & 5 & 6 \\
3 & 4 & 5 & 6 & 7 \\
4 & 5 & 6 & 7 & 8 \\
5 & 6 & 7 & 8 & 9 \\
\end{array} \]
2D Convolution Example

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
2 & 3 & 4 & 5 & 6 \\
3 & 4 & 5 & 6 & 7 \\
4 & 5 & 6 & 7 & 8 \\
5 & 6 & 7 & 8 & 9 \\
\end{array}
\quad \begin{array}{cccccc}
5 & 25 \\
\end{array}
\]

M

N

P

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 2 \\
0 & 2 & 0 \\
\end{array}
\]

5
2D Convolution – Ghost cells

GHOST CELLS
(apron cells, halo cells)
```c
__global__
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {
    int Col  =   blockIdx.x * blockDim.x + threadIdx.x;
    int Row  = blockIdx.y * blockDim.y + threadIdx.y;
}
```
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {

  int Col = blockIdx.x * blockDim.x + threadIdx.x;
  int Row = blockIdx.y * blockDim.y + threadIdx.y;
}

2D Convolution Kernel

```c
__global__
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {

    int Col = blockIdx.x * blockDim.x + threadIdx.x;
    int Row = blockIdx.y * blockDim.y + threadIdx.y;

    if (Col < w && Row < h) {
        // Calculate output image with masked values for all valid pixels
    }
}
```
2D Convolution Kernel

```c
__global__
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {

    int Col = blockIdx.x * blockDim.x + threadIdx.x;
    int Row = blockIdx.y * blockDim.y + threadIdx.y;

    if (Col < w && Row < h) {

        int pixVal = 0;
        // Element to start masking from.
        ..... 

    }
}
```
2D Convolution Kernel

```c
__global__
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {

    int Col = blockIdx.x * blockDim.x + threadIdx.x;
    int Row = blockIdx.y * blockDim.y + threadIdx.y;

    if (Col < w && Row < h) {

        int pixVal = 0;

        N_start_col = Col - (maskwidth/2);
        N_start_row = Row - (maskwidth/2);

        ....
    }
}
```
```c
__global__
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {

    int Col = blockIdx.x * blockDim.x + threadIdx.x;
    int Row = blockIdx.y * blockDim.y + threadIdx.y;

    if (Col < w && Row < h) {
        int pixVal = 0;

        N_start_col = Col - (maskwidth/2);
        N_start_row = Row - (maskwidth/2);

        // Get the of the surrounding box
        .....  

    }
}
```
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {

    int Col = blockIdx.x * blockDim.x + threadIdx.x;
    int Row = blockIdx.y * blockDim.y + threadIdx.y;

    if (Col < w && Row < h) {

        int pixVal = 0;

        N_start_col = Col – (maskwidth/2);
        N_start_row = Row – (maskwidth/2);

        // Get the of the surrounding box
        for(int j = 0; j < maskwidth; ++j) {
            for(int k = 0; k < maskwidth; ++k) {
                // .....  
            }
        }
    }
}
# 2D Convolution Kernel

```c
__global__
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {

    int Col = blockIdx.x * blockDim.x + threadIdx.x;
    int Row = blockIdx.y * blockDim.y + threadIdx.y;

    if (Col < w && Row < h) {

        int pixVal = 0;

        N_start_col = Col – (maskwidth/2);
        N_start_row = Row – (maskwidth/2);

        // Get the of the surrounding box
        for(int j = 0; j < maskwidth; ++j) {
            for(int k = 0; k < maskwidth; ++k) {
                int curRow = N_Start_row + j;
                int curCol = N_start_col + k;

                // Verify we have a valid image pixel
                if(curRow > -1 && curRow < h && curCol > -1 && curCol < w) {
                    pixVal += in[curRow * w + curCol] * mask[j*maskwidth+k];
                }
            }
        }

    }
}
```
2D Convolution Kernel

```c
__global__
void convolution_2D_basic_kernel(unsigned char * in, unsigned char * mask, unsigned char * out, int maskwidth, int w, int h) {

    int Col = blockIdx.x * blockDim.x + threadIdx.x;
    int Row = blockIdx.y * blockDim.y + threadIdx.y;

    if (Col < w && Row < h) {

        int pixVal = 0;

        N_start_col = Col - (maskwidth/2);
        N_start_row = Row - (maskwidth/2);

        // Get the of the surrounding box
        for(int j = 0; j < maskwidth; ++j) {
            for(int k = 0; k < maskwidth; ++k) {
                int curRow = N_Start_row + j;
                int curCol = N_start_col + k;

                // Verify we have a valid image pixel
                if(curRow > -1 && curRow < h && curCol > -1 && curCol < w) {
                    pixVal += in[curRow * w + curCol] * mask[j*maskwidth+k];
                }
            }
        }

        // Write our new pixel value out
        out[Row * w + Col] = (unsigned char)(pixVal);
    }
}
```
Tiled Convolution

- Thread to Output Data Index Mapping