高階合成技術於應用加速 (11020EE521800)

Lab B #5 - Convolution Filtering 110061639 呂易縉

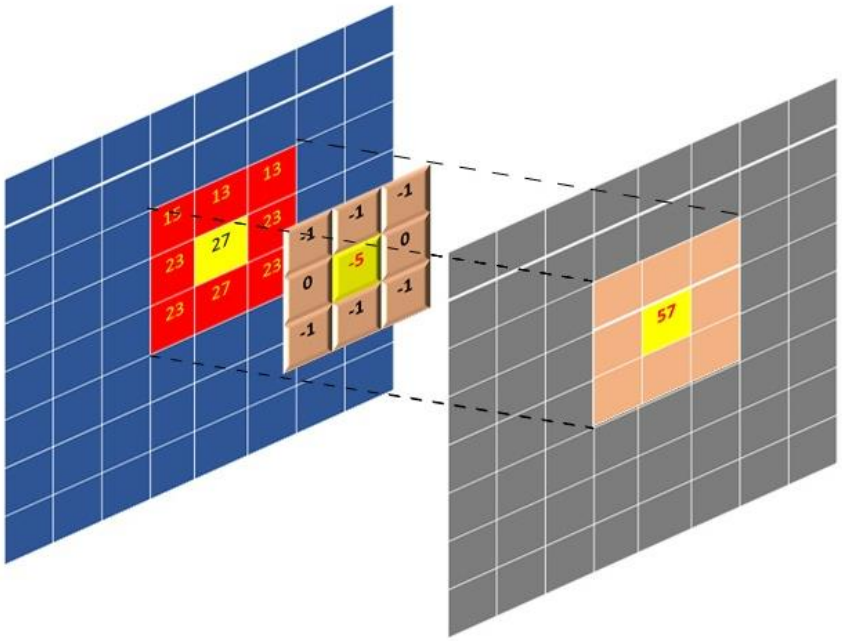
Algorithm introduction :

影音應用程式廣泛使用不同類型的濾波器，於不同的情況，像是濾除雜訊，操縱動態模糊，增強顏色和對比度，邊緣檢測等。

卷積濾波器的核心觀念是圍繞在對像素進行某種形式的數據加權，這種濾波器是對影音中每一幀中所有像素進行的，它進而重新定義像素與其周圍區域相關的數量和類型。

卷積運算本質上是在像素集和係數矩陣上執行的加權乘積的總和，下圖說明了如何計算像素的卷積，濾波器為大小 3x3 的係數矩陣，計算過程包括：

- 選擇以黃色標註顯示的輸入像素
- 提取大小與濾波器相同的矩陣
- 計算提取的矩陣和係數矩陣的元素乘積和
- 將乘積和放置在與前面所選輸入像素相同的位置上



1080p 高清影片的性能要求 ：

根據 1080p 高清 （HD） 的標準規格，可以計算出實際應用的性能要求，對於每秒 60 幀 （FPS） 的 1080p 高清影片，下面列出了規格以及以每秒像素為單位所需的輸送量：

```
Video Resolution         = 1920 x 1080
Frame Width (pixels)     = 1920
Frame Height (pixels)    = 1080
Frame Rate(FPS)          = 60
Pixel Depth(Bits)        = 8
Color Channels(YUV)      = 3
Throughput(Pixel/s)    = Frame Width * Frame Height * Channels * FPS
Throughput(Pixel/s)    = 1920*1080*3*60
Throughput (MB/s)      = 373 MB/s
```

Software Implementation :

卷積濾波器在軟體中是使用典型的多層 for loops 結構來實現。外面兩層 for loop 用於選擇要處理的像素 ； 內兩層 for loop 用於執行與係數矩陣的乘積和運算。 (P.s. 本演算法假設圖片邊界之外的所有像素值皆為零)

```
void Filter2D(
        const char           coeffs[FILTER_V_SIZE][FILTER_H_SIZE],
        float                factor,
        short                bias,
        unsigned short       width,
        unsigned short       height,
        unsigned short       stride,
        const unsigned char *src,
        unsigned char       *dst)
{
    for(int y=0; y<height; ++y)
    {
        for(int x=0; x<width; ++x)
        {
            // Apply 2D filter to the pixel window
            int sum = 0;
            for(int row=0; row<FILTER_V_SIZE; row++)
            {
                for(int col=0; col<FILTER_H_SIZE; col++)
                {
                    unsigned char pixel;
                    int xoffset = (x+col-(FILTER_H_SIZE/2));
                    int yoffset = (y+row-(FILTER_V_SIZE/2));
                    // Deal with boundary conditions : clamp pixels to 0 when outside of image
                    if ( (xoffset<0) || (xoffset>=width) || (yoffset<0) || (yoffset>=height) ) {
                        pixel = 0;
                    } else {
                        pixel = src[yoffset*stride+xoffset];
                    }
                    sum += pixel*coeffs[row][col];
```

```
            }

            // Normalize and saturate result
            unsigned char outpix = MIN(MAX((int(factor * sum)+bias), 0), 255);

            // Write output
            dst[y*stride+x] = outpix;
        }
    }
}
```

下圖顯示了 top function 如何為具有三個不同部分或通道的圖片調用卷積濾波器函數。這裡 OpenMP 編譯指示用於並行化使用多個線程的軟體執行。

```
  #pragma omp parallel for num_threads(3)
for(int n=0; n<numRunsSW; n++)
{
  // Compute reference results
  Filter2D(filterCoeffs[filterType], factor, bias, width, height, stride, y_src, y_ref);
  Filter2D(filterCoeffs[filterType], factor, bias, width, height, stride, u_src, u_ref);
  Filter2D(filterCoeffs[filterType], factor, bias, width, height, stride, v_src, v_ref);
}
```

Running the Software Application :

```
------------------------------------------------
Number of runs     : 60
Image width        : 1920
Image height       : 1080
Filter type        : 6

Generating a random 1920x1080 input image
Running Software version on 60 images

CPU  Time          :     23.6291 s
CPU  Throughput    :     15.0644 MB/s
------------------------------------------------
```

測得的性能僅為"2.42FPS"，而所需的輸送量為 60 FPS，因此為了滿足所需的性能，軟體實現需要加速 25 倍。(373 / 15.0644 = 24.76)

Hardware Implementation :

在給定性能限制，進而了解要用哪種硬體實現前，可以先檢查卷積 kernel：

- Core 的計算在 4 層 for loop 中完成，可將其分解成每個要輸出的像素計算
- 從濾波器原始碼中可以看出，當兩個內部迴圈完成一次執行時，將生成單個輸出像素
- 這兩個迴圈就是表示係數矩陣和圖像子矩陣上執行乘積和，而矩陣大小由係數矩陣定義，即 15x15
- 兩個內部迴圈的執行大小為 225（15x15），換句話說，每個輸出像素共執行了 225 次乘法累加（MAC）運算

Baseline Hardware Implementation Performance :

每個週期只可執行一個 MAC，可以根據 MAC 估計性能，此處假定硬體時鐘頻率為 300MHz，如下所示：

```
MACs per Cycle = 1
Hardware Fmax(MHz) = 300
Throughput  = 300/225 = 1.33 (MPixels/s) =  1.33 MB/s
```

根據原始碼，還可估計輸入和輸出端需要多少記憶體頻寬才能實現。從上所示，兩個內部迴圈在計算單個輸出像素時，在輸入端執行 225（15 * 15）讀取，因此：

```
Output Memory Bandwidth = Throughput = 1.33 MB/s
Input Memory Bandwidth  = Throughput * 225 = 300 MB/s
```

由上可知，記憶體頻寬的要求非常小，而 60FPS 1080p 高清影片所需的輸送量為 373 MB/s。下圖所示要滿足性能要求，需分別加速多少倍：

```
Acceleration Factor to Meet 60FPS Performance = 373/1.33 = 280x
Acceleration Factor to Meet SW Performance    = 14.5/1.33 = 10.9x
```

要將硬體實現的需將性能提高 280 倍才能處理 60 FPS，可採取的方法之一是將內部迴圈和 channel unroll。例如，通過 unroll 最裡面的迴圈，可將性能提高 15 倍。通過這一更改，硬體性能已經比單純使用軟體實現更好，但還不足以滿足所需影片的性能。進一步，可將內部兩個迴圈都 unroll，即可將性能提高 225 倍，這意味著每個週期的輸送量為 1 輸出像素，性能和記憶體頻寬要求如下：

```
Throughput  = Fmax * Pixels produced per cycle = 300 * 1 = 300 MB/s
Output Memory Bandwidth = Fmax * Pixels produced per cycle =  300 MB/s
Input Memory Bandwidth = Fmax * Input pixels read per output pixel = 300 * 225 = 67.5 GB/s
```

所需的輸出記憶體頻寬隨輸送量線性擴展，且輸入記憶體頻寬已大幅上升，可能無法負荷。觀察卷積濾波器可以發現，不需將輸入記憶體中讀取所有 225 像素進行處理。可以構建一種新的緩存方案，以避免如此大量地使用輸入記憶體頻寬。

可對 kernel 進行優化，以增加輸入數據的重複使用次數，這可導致頻寬需求大幅降低。使用這種方法，可使所需的輸入與輸出的頻寬相同，即大約 300 MB/s，當兩個內部迴圈都展開時，只需要讀取 1 個輸入像素，平均即可產生一個輸出像素，因此輸入記憶體頻寬為 300 MB / s。雖然可以降低輸入頻寬，但仍將僅為 300 MB/s，低於所需的 373 MB/s。

要解決此問題，其中一種方法是複製 kernel instance，也稱為計算單元(compute units)。增加計算單元數，以便平行處理數據。在本例中，可在單獨的計算單元上處理所有顏色通道(YUV)，使用三個計算單元（每個顏色通道一個）時，預期的性能摘要將如下所示：

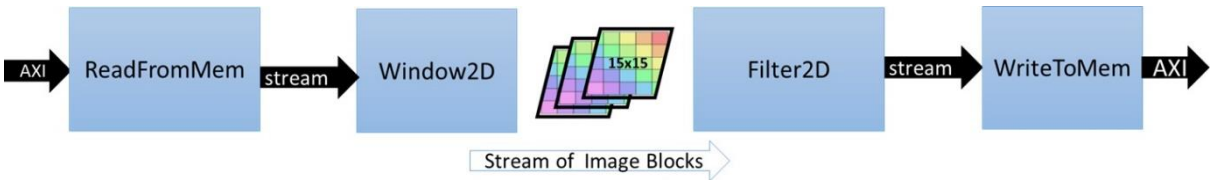
```
Throughput(estimated)  = Performance of Single Compute Unit * No. Compute Units = 300 x 3 = 900 MB/s
Acceleration Against Software Implementation = 900/14.5 = 62x
Kernel Latency ( per image on any color channel ) = (1920*1080) / 300 = 6.9 ms
Video Processing Rate = (1/Kernel Latency) = 144  FPS
```

Top Level Structure of Kernel :

```
void Filter2DKernel(
        const char           coeffs[256],
        float                factor,
        short                bias,
        unsigned short       width,
        unsigned short       height,
        unsigned short       stride,
        const unsigned char  src[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT],
        unsigned char        dst[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT])
  {
    #pragma HLS DATAFLOW

  // Stream of pixels from kernel input to filter, and from filter to output
    hls::stream<char,2>    coefs_stream;
    hls::stream<U8,2>      pixel_stream;
    hls::stream<window,3>  window_stream; // Set FIFO depth to 0 to minimize resources
    hls::stream<U8,64>     output_stream;

  // Read image data from global memory over AXI4 MM, and stream pixels out
    ReadFromMem(width, height, stride, coeffs, coefs_stream, src, pixel_stream);

    // Read incoming pixels and form valid HxV windows
    Window2D(width, height, pixel_stream, window_stream);

  // Process incoming stream of pixels, and stream pixels out
  Filter2D(width, height, factor, bias, coefs_stream, window_stream, output_stream);

  // Write an incoming stream of pixels and write them to global memory over AXI4 MM
  WriteToMem(width, height, stride, output_stream, dst);
  }
```

Dataflow 由以下四種不同的功能組成：

- ReadFromMem：從主記憶體讀取像素數據或影片輸入
- Window2D：本地緩存，在輸出端有大小為 15x15
- Filter2D：核心濾波演算法
- WriteToMem：將輸出數據寫入主記憶體

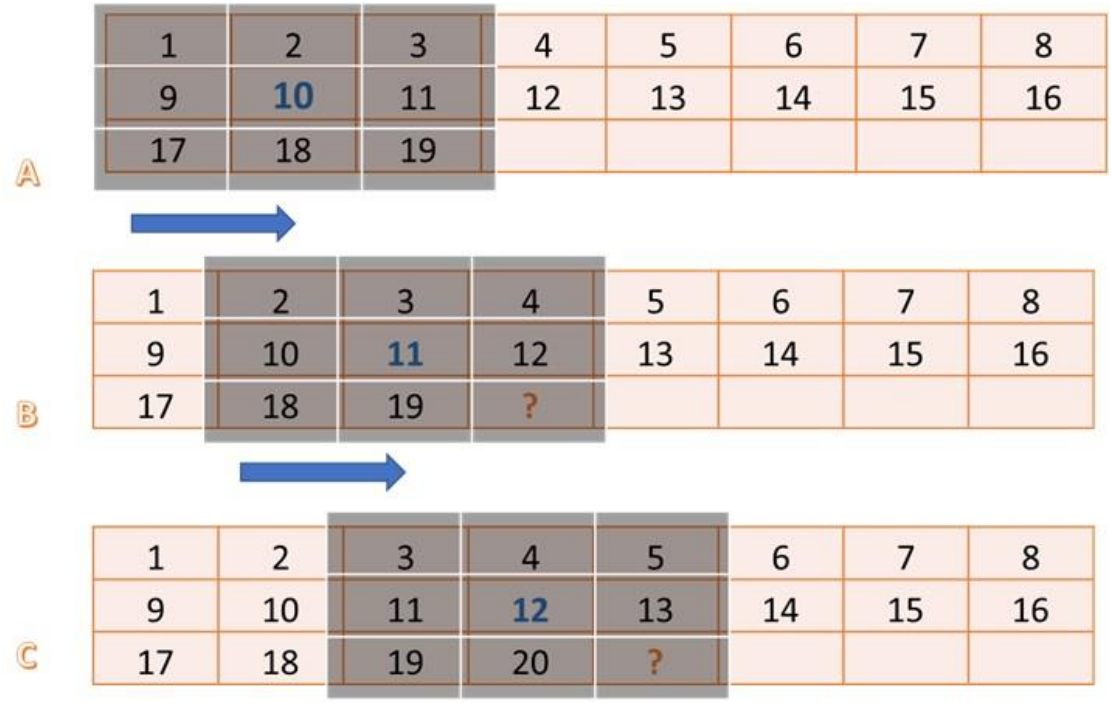


```
Memory Accesses to Read filter Co-efficients = 15x15 = 225
Memory Accesses to Read Neighbouring Pixels  = 15x15 = 225
Memory Accesses to Write to Output           = 1
Total Memory Accesses                       = 451
```

從純軟體實現的角度來看，在輸出端生成單個樣本，需要在輸入端進行 450 次記憶體讀取，對輸出進行 1 次寫入存取。即使許多讀取因緩存而變得快速，但大量的記憶體使用，還是會成為瓶頸。但在 FPGA 上設計可以輕鬆構建高效的數據移動和存取方案。其中一個關鍵和主要優勢是可以對頻寬有自定義的配置。

Window2D: Line and Window Buffers

由兩個基本模塊組成，Line buffer 及 Window，



上圖顯示了生成像素 11 所需的內容，B 中的 window，與之前 A 中的有很大的重疊。在觀察後可發現從輸入端來看，它只需要一個新像素，即可產生一個輸出像素。

```
--------------------------------------------------------------------------------
HLS Testbench for Xilinx 2D Filter Example
Image info
- Width     :       1000
- Height    :         30
- Stride    :       1024
- Bytes     :      30720
Running FPGA accelerator
Comparing results
Test PASSED: Output matches reference
--------------------------------------------------------------------------------
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
```

模擬了寬度 = 1000 且高度 = 30 的圖片

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
Filter2DKernel				0.00	2211165	7.370E6	-	2211166	-	dataflow	44	139	27186	25206	0
ReadFromMem				0.00	2211165	7.370E6	-	2211165	-	no	0	1	2469	1484	0
Window2D				0.00	2087055	6.956E6	-	2087055	-	no	14	1	2237	492	0
entry_proc				-	0	0.0	-	0	-	no	0	0	3	47	0
Filter2D				0.00	2073859	6.912E6	-	2073859	-	no	0	136	16412	17705	0
WriteToMem				0.00	2210837	7.369E6	-	2210837	-	no	0	1	889	1299	0

上圖顯示了 top module 主要使用 139 個 DSP 進行 SOP 操作，以及數據行動器塊使用 14 個 BRAM

還可從 co-simulation 報告中獲得 kernel 延遲的準確測量值，結果顯示如下：

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
Filter2DKernel				38441	38441	38441
ReadFromMem				31129	31129	31129
Window2D				38244	38244	38244
entry_proc				0	0	0
Filter2D				38270	38270	38270
WriteToMem				38440	38440	38440

由於模擬了 1000x30 的圖片，因此延遲應為 30000 + 某些模塊的固定延遲。此處 8441 就是固定延遲，當實際圖像大小為 1920x1080 時，固定延遲將在更多圖像行中攤銷。大圖像會攤銷延遲的事實可以通過使用較大的圖像進行模擬來驗證。

驗證 kernel 是否可實現每個週期一個輸出樣本輸送量，如下所示：

WriteToMem	0.00	2210837	7.369E6	-	2210837	-	no	0	1	889	1299	0
WriteToMem_Pipeline_write_image	0.00	2210763	7.368E6	-	2210763	-	no	0	0	640	692	0
write_image	-	2210761	7.368E6	3	1	2210760	yes	-	-	-	-	-

Running Software Emulation :

```
Xilinx 2D Filter Example Application (Randomized Input Version)

FPGA binary       : ./fpgabinary.sw_emu.xclbin
Number of runs    : 1
Image width       : 1920
Image height      : 1080
Filter type       : 3
Max requests      : 6
Compare perf.     : 1

Programming FPGA device
Generating a random 1920x1080 input image
Running FPGA accelerator on 1 images
  finished Filter2DRequest
  finished Filter2DRequest
  finished Filter2DRequest
Running Software version
Comparing results

Test PASSED: Output matches reference
```

輸入和輸出影像如下所示，將濾波器類型選擇設置為 3，以執行邊緣檢測：