LAB B

Introduction to the algorithm

CRS：compressed row storage，舉例如下

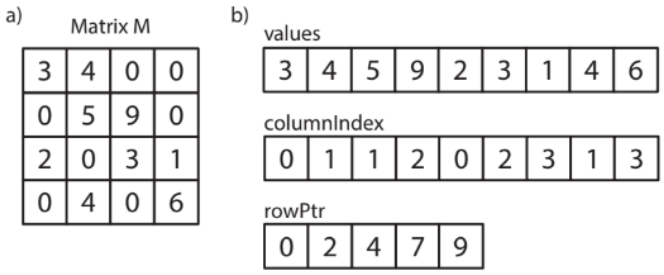


Figure 6.1: A 4 × 4 matrix **M** represented in two different ways: as a 'dense' matrix stored in a two-dimensional array, and as a sparse matrix stored in the compressed row storage (CRS) form, a data structure consisting of three arrays.

values 數組儲存非零值，columnIndex 儲存了非零值所對應的 column 值，rowPtr 則對應了非零值起始位置的 index。利用這種 CRS 的方法減少計算量，使 sparse matrix 的運算提高效率。

The original code explanation

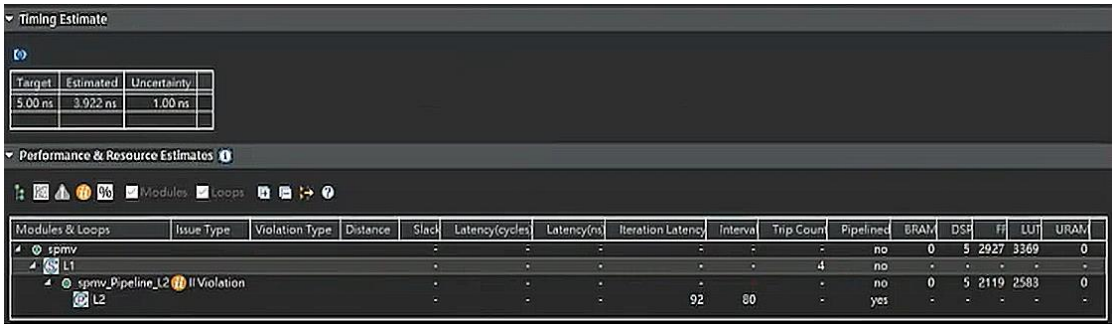
```
#include "spmv.h"

void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
        DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE])
{
#pragma HLS TOP name=spmv
#pragma HLS INTERFACE mode=s_axilite port=rowPtr
#pragma HLS INTERFACE mode=s_axilite port=columnIndex
#pragma HLS INTERFACE mode=s_axilite port=values
#pragma HLS INTERFACE mode=s_axilite port=y
#pragma HLS INTERFACE mode=s_axilite port=x
#pragma HLS INTERFACE mode=s_axilite port=return

#pragma HLS AGGREGATE compact=auto variable=rowPtr
L1: for (int i = 0; i < NUM_ROWS; i++) {
        DTYPE y0 = 0;
    L2: for (int k = rowPtr[i]; k < rowPtr[i+1]; k++) {

#pragma HLS unroll factor=8
#pragma HLS pipeline

            y0 += values[k] * x[columnIndex[k]];
        }
        y[i] = y0;
    }
}
```

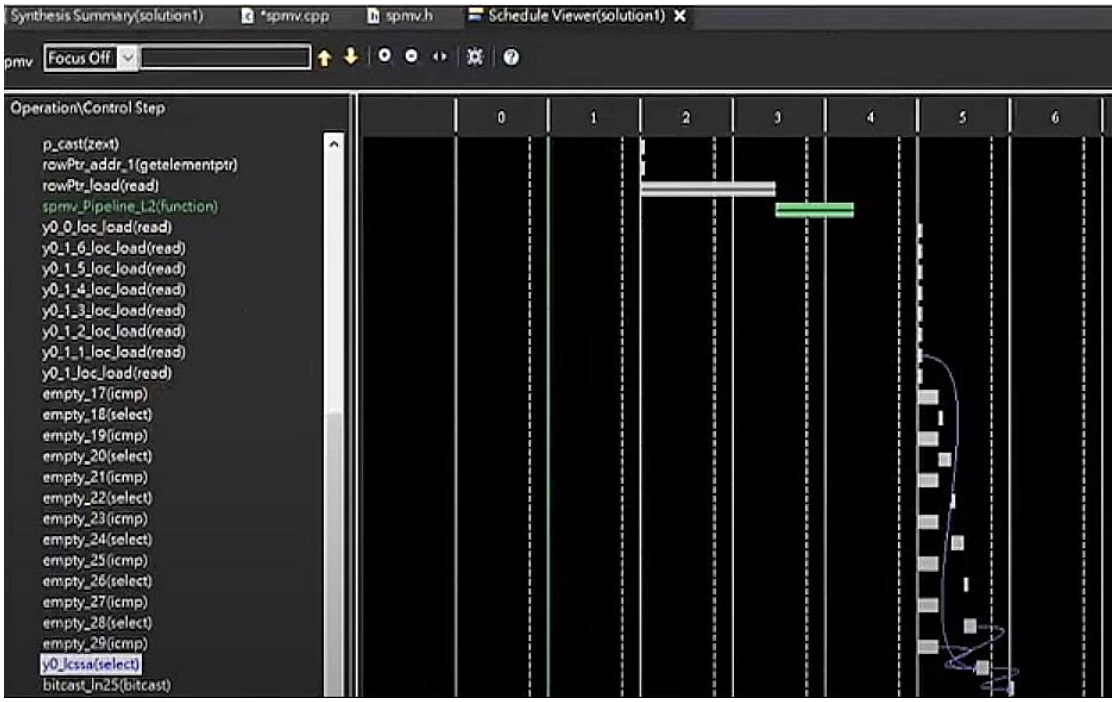
首先我是用 AXI-Lite 的 interface 來簡單跑一下 source code。

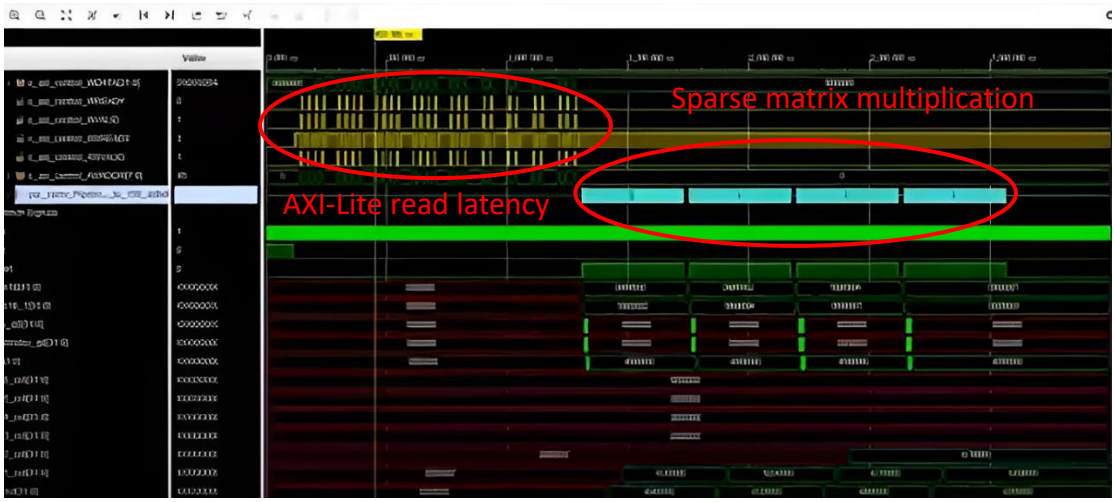
然後可以看到我用了兩個 loop，特別要注意的是我框起來的 L2 這個 loop。我們利用 column matrix 其中一個 vector 的 index 去 access 其它兩個 vector 從而定位 sparse matrix 中非零值的位置。所以這個 L2 的 for-loop 的 loop bound 不是一個定值，而是一個變數。所以在使用 HLS 的優化方法就會遇到不便性。

最後還可能會遇到一個問題就是，在做 matrix multiplication 時會有 MAC 的問題，如果直接跑 HLS 可能會遇到 RAW dependence 的問題。但是 code 中有做 unroll 的處理或許可以解決以上問題，但會影響 cycle 的最小值。

Result analysis







由第一張圖的 report 可知硬體資源的利用狀況，然後也看到有出現 II Violation 的問題，但實際再用此 code 在 PYNQ 跑的時候卻能跑出正確結果。

從第二張 schedule viewer 的圖可以看到它同步產生 8 個 read access 的 parallel process。但這 8 個 read 的 process 無法同時執行，而是按時間順序使用頻寬。

所以這裡就可能產生以下 RAW bottleneck 的問題。
從第三張 wave viewer 可以看到 data 讀寫時間幾乎佔了整個執行時間的一半，造成 pipeline 執行的低效。

Optimization

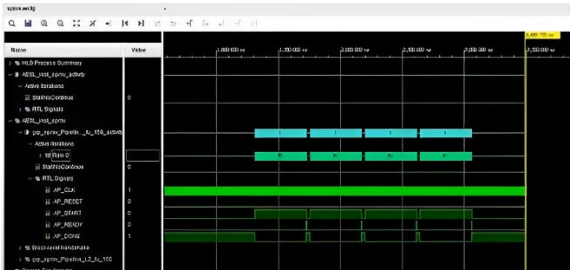
```
#include "spmv.h"

void spmv(int rowPtr[NUM_ROWS+1], int columnIndex[NNZ],
        DTYPE values[NNZ], DTYPE y[SIZE], DTYPE x[SIZE])
{
#pragma HLS TOP name=spmv
#pragma HLS INTERFACE mode=m_axi depth=5 port=rowPtr
#pragma HLS INTERFACE mode=m_axi depth=16 port=columnIndex
#pragma HLS INTERFACE mode=m_axi depth=16 port=values
#pragma HLS INTERFACE mode=m_axi depth=4 port=y
#pragma HLS INTERFACE mode=m_axi depth=4 port=x
#pragma HLS INTERFACE mode=m_axi port=return

#pragma HLS ARRAY_RESHAPE dim=1 variable=rowPtr
#pragma HLS ARRAY_RESHAPE dim=1 variable=columnIndex

L1: for ( __int8 i = 0; i < NUM_ROWS; i++) {
#pragma HLS LOOP_TRIPCOUNT avg=4 max=4 min=4
        DTYPE y0 = 0;
    L2: for (__int8 k = rowPtr[i]; k < rowPtr[i+1]; k+=7) {
#pragma HLS LOOP_TRIPCOUNT avg=2 max=4 min=0
//#pragma HLS unroll factor=7
#pragma HLS PIPELINE
            DTYPE y_par0 = values[k] * x[columnIndex[k]];

        L3: for (int l=1; l<7; l++){//II_com = 7
                if ( l+k < rowPtr[i+1]){
                    y_par0 += values[l+k] * x[columnIndex[l+k]];
                }
            }
            y0 += y_par0;
        }
        y[i] = y0;

    }
```

(1)

(2)

(3)

(1) 首先我把 AXI-Lite 改為 AXI master 的 interface，因為後者支援 DMA 和 Burst access ， 會比前者提供更好的效能，更適合傳送 matrix。
(2) 為了更好的使用記憶體頻寬，也有加入 array reshape 的 paragma 指令對 R/W data 做 reshape。
(3) 將 L2 拆分成 L2 與 L3 ， 在每一個設定多一個 varible 之後，把原本中間要做的事情丟到 L3 裡，多加一變數進行 value 的累加。相較於之前的程式將此變數定為 0 的做法更加有效率。將原本的值改為 partial sum 的結果，提高效率。
(4) 結果如下：執行時間由 7.36us 變為 3.50us。



What to learn

一般拿到一個演算法都會把 memory access 跟 computation 寫在一起，在本 lab 中先要讀進 row，然後再得知 column pointer 要從哪裡開始。一般的演算法也都是按照這個順序來的，所以使用 pipeline 等方法就是沒有辦法簡單的做平行處

理，inner loop 做完下面的才開始。一個 basic 的技巧就是要把抓 data 的部分拉開，output 出去的要分開，再去看 computation。Data 如果總是 variable 的話，其實可以考慮使用 stream 的方法，computation 只要 FIFO stream 有 data 就一直在做。 剩下的一個問題就是需要知道 memory access 需要知道運算的 sequence 是什麼，再做 schedule。以上三件事情 overlap 起來就會更有效率。

Problem encountered

首先在跑 source code 得到的 report 有出現 II violation，但實際上不影響 pynq 的使用，就沒有去在意這個問題。

其次，我也有試過改 unroll factor 來做優化，但是會出現編譯錯誤，所以我後來就沒有使用這種方法。

Github

https://github.com/Barry-Sung/LAB_B