E0-294: Systems for Machine Learning

Assignment-1

Akash Maji
CSA, 24212

Tuesday 28th January, 2025

Problem 1:

Consider a convolution operation with the following parameters.

- $N = 8, C = 4, H = 32, W = 32, R = 5, S = 5, M = 32$

Part (a):

Implement the 7-loop naïve implementation of convolution. (5)

Answer(a)

The naive implementation of convolution uses a 7-layer nested for loop. The 'OUTPUT' is obtained by convolving 'FILTER' and 'INPUT'. All the symbols have their usual meanings. The 'OUTPUT' is of size N * M * E * F. Here, E = (H-S+1) and F = (W-R+1). All the values for INPUT, FILTER and BIAS are randomly chosen from [-1.0, 1.0] range. The snippet of naive convolution is given as under. For details, refer **part01.cpp** file.

```
/* implementation idea taken from course slides */
void naive_convolution(){
    // Naive 7-layer loop implementation
    for (int n = 0; n < N; n++) {
      for (int m = 0; m < M; m++) {
        for (int x = 0; x < E; x++) {
          for (int y = 0; y < F; y++) {
            // Initialize the output value with bias
              OUTPUT[n][m][x][y] = BIAS[m];
              for (int i = 0; i < R; i++) {
                for (int j = 0; j < S; j++) {
                  for (int k = 0; k < C; k++) {
                    OUTPUT[n][m][x][y] += INPUT[n][k]
                    [Ux*x+i][Uy*y+j] * FILTER[m][k][i][j];
                  }
                }
                }
            }
        }
    }
  }
}
```

Part (b):

Implement the convolution by flattening. (5)

Answer(b)

We begin by flattening INPUT and FILTER. The flattened FILTER is of size M * CRS, while flattened INPUT is of size N * CRS * EF. The resultant INPUTS_FLAT is Toeplitz matrix. Sample code snippets are given as under. For details, refer **part02.cpp** file.

```
/* 4-layer loop implementation */
void toeplitz_multiply() {
    for (int n = 0; n < N; ++n) {
      for (int m = 0; m < M; ++m) {
        for (int e = 0; e < E * F; ++e) {
            OUTPUT_FLAT[n][m][e] = BIAS[m];
            for (int crs = 0; crs < C * R * S; ++crs) {
              OUTPUT_FLAT[n][m][e] += FILTERS_FLAT[m][crs] *
                                        INPUTS_FLAT[n][crs][e];
            }
        }
      }
    }
}
```

```
void flatten_filters(){
  for(int m = 0; m < M; m++){
      int crs = 0;
      for(int c = 0; c < C; c++){
        for(int r = 0; r < R; r++){
          for(int s = 0; s < S; s++ ){
            FILTERS_FLAT[m][crs++] = FILTER[m][c][r][s];
          }
        }
      }
  }
}
void flatten_inputs(){
  for (int n = 0; n < N; ++n) {
    for (int c = 0; c < C; ++c) {
      for (int r = 0; r < R; ++r) {
        for (int s = 0; s < S; ++s) {
          for (int x = 0; x < E; ++x) {
            for (int y = 0; y < F; ++y) {
              int i = c * R * S + r * S + s;
              int j = x * F + y;
              INPUTS_FLAT[n][i][j] = INPUT[n][c][x+r][y+s];
            }
          }
        }
      }
    }
  }
}
```

Part (c):

Show that both cases produce equal output. (5)

Answer(c)

The output from **part01** and **part02** are stored in the files **OUTPUT1.txt**(generated by ‘part01‘) and **OUTPUT2.txt** (generated by ‘part02‘) respectively. We then run a program **checker** (compiled from **checker.cpp**) to see that both files indeed produce the same results for OUTPUT maps. In part(a), the OUTPUT map is of size N*M*E*F, while in part(b), the OUTPUT map is of size N * M * EF, indicating the same size.

Part (d):

Obtain and report the number of instructions and execution time for both (a) and (b) using the perf command. (5)

Answer(d)

We run the following commands and obtain the below measurements for the two parts.

```
$ perf stat -e instructions,cycles  ./part01
$ perf stat -e instructions,cycles  ./part02
```

Table 1: Performance Metrics for part(a) and part(b)

Metric	**part01**	**part02**
Instructions	2,32,90,50,892	1,58,42,57,791
Cycles	64,42,45,838	43,95,64,370
Instructions per cycle (IPC)	3.62	3.60
Time elapsed (seconds)	0.156478924	0.113216107
User time (seconds)	0.149440000	0.099209000
System time (seconds)	0.007020000	0.008016000

Part (e):

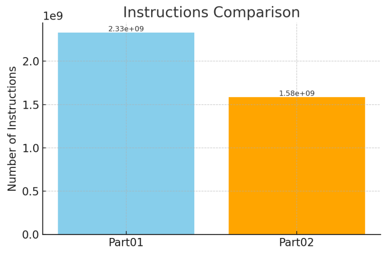
Analyze and comment on the above results. (5)

Answer(e)

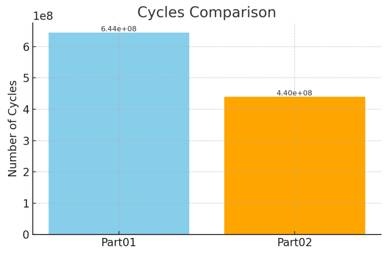
The utility ‘perf‘ is a Linux tool that analyzes system performance by accessing the processor’s performance monitoring unit (PMU). It’s used to record and display events like number of instructions, cycles, time taken etc.
We can see from the comparison graphs that the performance is significantly better in part02 than in part01 in given parameters. Hence, Toeplitz convolution achieves better performance, although taking up more space to store INPUT. Because it allows for significantly faster computation by transforming the convolution operation into a simple matrix multiplication with a Toeplitz matrix.

Metric	Improvement
Instructions	31.96% improvement
Cycles	31.73% improvement
Time Elapsed	27.63% improvement

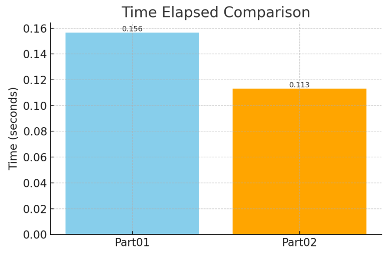
Table 2: Improvement Metrics



(a) Instructions Comparison



(b) Cycles Comparison



(c) Time Elapsed Comparison

Figure 1: Performance Comparison for Instructions, Cycles, and Time Elapsed

The following system was used in Problem1:

Component	Specification
System	$Ryzen7 - 5800H$
Cores	8
Threads	16
Base Clock	$3.2 - GHz$
RAM	$24 - GB$

The following code files and README are available in Problem1: GitHub

Problem 2:

Given an input feature X whose dimensions are $[h = 3, w = 3, i = 2]$:

$$X = \begin{bmatrix} -8 & 4 & 8 \\ -2 & -9 & -7 \\ 7 & -5 & 7 \end{bmatrix}, \quad \begin{bmatrix} -7 & 6 & 7 \\ -9 & -3 & -8 \\ 1 & -3 & 0 \end{bmatrix}$$

Given a convolution weight whose dimensions are $[h = 2, w = 2, i = 2, o = 3]$:

$$W_{\text{out channel 1}} = \begin{bmatrix} -8 & 1 \\ -2 & 3 \end{bmatrix}, \quad \begin{bmatrix} -6 & -6 \\ 2 & 5 \end{bmatrix}$$

$$W_{\text{out channel 2}} = \begin{bmatrix} -6 & 9 \\ -7 & 5 \end{bmatrix}, \quad \begin{bmatrix} -4 & 7 \\ -1 & 0 \end{bmatrix}$$

$$W_{\text{out channel 3}} = \begin{bmatrix} -2 & -3 \\ -2 & 1 \end{bmatrix}, \quad \begin{bmatrix} -3 & -9 \\ 1 & 0 \end{bmatrix}$$

Other 2D convolution parameters are:

- No zero padding
- Stride = 1
- Average pooling

Part (a):

What is the dimension of $Y = \text{Conv2D}(W, X)$, and what is the total number of elements in the output feature? (13)

Answer(a)

The dimension of X is H * W * C = 3 * 3 * 2, while that of each W_i is S * R * C = 2 * 2 * 2.
Since, the stride is 1, so the Y will have dimension E * F = (H-S+1) * (W-R+1) = 2 * 2.
Also, there are 3 W maps, so Y will have final dimension E * F * M = 2 * 2 * 3.
Thus, the total number of elements in Y is 2 * 2 * 3 = 12.
Now, if we do 2 * 2 pooling after the convolution of Y, we will get Y with dimension 1 * 1 * 3 = 3 elements.

Part (b):

Convert $Y = \text{Conv2D}(W, X)$ into matrix multiplication $Y' = W' \times X'$ using the Toeplitz matrix generation technique introduced in class. (12)

Answer(b)

In Toeplitz matrix generation for W to get W', we place the M FILTERS in M rows, and each row has all C channels placed one after other. So the dimension of W' is M * CRS, where R*S is the dimension of a FILTER.
Also, for X', we place C channels one after another vertically downwards. The dimension of X' is CRS * EF.

The filters W_1, W_2, W_3 are flattened row-wise into W' as described earlier:

$$W_1' = \begin{bmatrix} -8 & 1 & -2 & 3 & -6 & -6 & 2 & 5 \end{bmatrix}$$
$$W_2' = \begin{bmatrix} -6 & 9 & -7 & 5 & -4 & 7 & -1 & 0 \end{bmatrix}$$
$$W_3' = \begin{bmatrix} -2 & -3 & -2 & 1 & -3 & -9 & 1 & 0 \end{bmatrix}$$
$$W' = \begin{bmatrix} -8 & 1 & -2 & 3 & -6 & -6 & 2 & 5 \\ -6 & 9 & -7 & 5 & -4 & 7 & -1 & 0 \\ -2 & -3 & -2 & 1 & -3 & -9 & 1 & 0 \end{bmatrix}$$

For each 2×2 filter sliding window over X (stride 1), extract patches from both channels and append column-wise.

$$X_1' = \begin{bmatrix} -8 & 4 & -2 & -9 \\ 4 & -2 & -9 & -7 \\ -2 & -9 & 7 & -5 \\ -9 & -7 & -5 & 7 \end{bmatrix}$$

$$X_2' = \begin{bmatrix} -7 & 6 & -9 & -3 \\ 6 & 7 & -3 & -8 \\ -9 & -3 & 1 & -3 \\ -3 & -8 & -3 & 0 \end{bmatrix}$$

$$X' = \begin{bmatrix} -8 & 4 & -2 & -9 \\ 4 & -2 & -9 & -7 \\ -2 & -9 & 7 & -5 \\ -9 & -7 & -5 & 7 \\ -7 & 6 & -9 & -3 \\ 6 & 7 & -3 & -8 \\ -9 & -3 & 1 & -3 \\ -3 & -8 & -3 & 0 \end{bmatrix}$$

Thus, $Y' = W' \times X' = \begin{bmatrix} -8 & 1 & -2 & 3 & -6 & -6 & 2 & 5 \\ -6 & 9 & -7 & 5 & -4 & 7 & -1 & 0 \\ -2 & -3 & -2 & 1 & -3 & -9 & 1 & 0 \end{bmatrix} * \begin{bmatrix} -8 & 4 & -2 & -9 \\ 4 & -2 & -9 & -7 \\ -2 & -9 & 7 & -5 \\ -9 & -7 & -5 & 7 \\ -7 & 6 & -9 & -3 \\ 6 & 7 & -3 & -8 \\ -9 & -3 & 1 & -3 \\ -3 & -8 & -3 & 0 \end{bmatrix}$

And, dimension of Y' is 3 * 4 = 12 elements.