Nov 23, 2023

APAI2023 - LAB07

PULP Tiling - part2

Authors: Luka Macan, Lorenzo Lamberti, Alessio Burrello, Francesco Conti
Contacts: luka.macan@unibo.it, lorenzo.lamberti@unibo.it

Links: ***GitHub Link (code)*** ***GDOC link (assignment)***

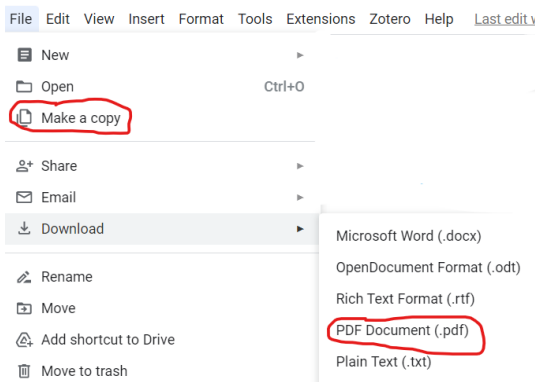
Summary

1. Subject(s):
    - Double buffering
    - Tiling on Conv3x3
2. Programming Language: C
3. Lab duration: 3h
4. Assignment:
    - Time for delivery: 1 week
    - **Submission deadline:** Nov 30, 2023 at 16:00

How to deliver the assignment

You will deliver ONLY THIS TEXT FILE, no code

- Copy this google doc to your drive, so that you can modify it. (`File -> make a copy`)
- Fill the tasks on this google doc.
- Export to pdf format.
- Rename the file to: LAB<number_of_the_lesson>_APAI_<your_name>.pdf
- Use Virtuale platform to load ONLY your .pdf file



Setup

Access to the remote server, and setup

- Open this web page: https://compute.eees.dei.unibo.it:8443/guacamole/ **(works only from ALMA WIFI NETWORK!).**
- Login. Use the credentials provided during the last labs;
- Open a terminal (right click – open a new terminal)

Download the repo

```
$ cd <work_dir>
$ git clone <repo_link_here>
```

Load modules

Load appropriate modules to be able to compile and run the code:

```
$ module load pulp-sdk
$ module load dory-conda
```

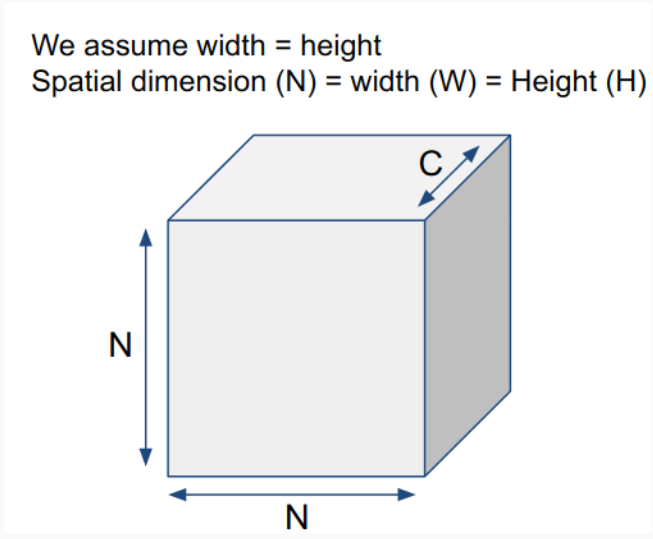
Note: Always do this when opening a new terminal session.

How to run the code

```
1. cd APAI23-LAB07-PULP-Tiling-part2/double-buffering
```

```
2. python3 parameters_generate.py --kernel-shape=1 --channels=32
--output-spatial-dimensions=32
3. make clean all run
```

“What is the spatial dimension?”



Note: in Lab06 you had to change by hand the size of the input in Src/main.c after generating the network with parameters_generate.py.

```
#define CHANNELS 32
#define SPATIAL_DIM 32
```

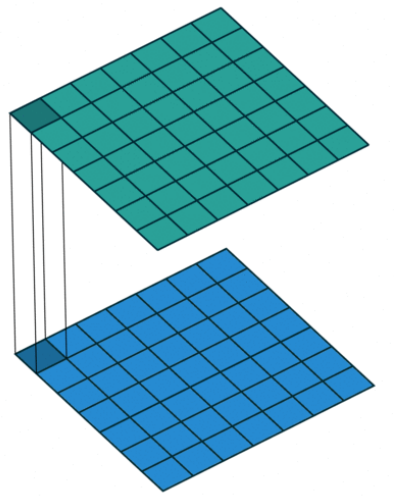
In this lab07 the change in main.c is done automatically for you!

LAB STARTS HERE

Exercise 4: Tiling with double-buffering on Conv1x1

We tackle a 2D convolution with this size: Kernel = 1x1, Stride = 1, Padding = 0

1x1 convolution:



Task 4.1. Implementing missing code:

- Implement double buffering File: `double-buffering/Src/layer_execution.c/`

Instructions in the code.

Search and complete these fields: `/* YOUR CODE HERE */`

```
const int nb_tiles_total = nb_h_tile * nb_w_tile;
int buffer_idx = 0;

/*
* Load tile 0's input data (outside the for loop)
*/
/* YOUR CODE HERE */;

for (h_tile_idx = 0; h_tile_idx < nb_h_tile; h_tile_idx++) {
  for (w_tile_idx = 0; w_tile_idx < nb_w_tile; w_tile_idx++) {
    const int next_buffer_idx = (buffer_idx + 1) % 2;
    const int next_w_tile_idx = (w_tile_idx + 1) % nb_w_tile;
    const int next_h_tile_idx = h_tile_idx + (next_w_tile_idx == 0 ? 1 : 0);  // If the next_w_tile is 0, that means we went into a new row, so the next_h_tile is h_tile + 1
    const int next_tile_idx = next_h_tile_idx * nb_w_tile + next_w_tile_idx;

    /*
     * Wait for the current tiles data
     * Note: we are waiting here and not right before the kernel, because
     * otherwise we would wait for the next tile too and ruin the point
     * of double buffering.
     */
    /* YOUR CODE HERE */;

    /*
    * Load next tiles input data from L2 into L1
    */
    if (next_tile_idx < nb_total_tiles) { // Check if there exists a 'next' tile
      /* YOUR CODE HERE */;
    }

    /*
     * Kernel execution
     */
    /* YOUR CODE HERE */;

    /*
     * Store output data back into L2 from L1
     */
    /* YOUR CODE HERE */;

    buffer_idx = next_buffer_idx;
  }
}

/*
* Last wait for the DMA transfers
*/
/* YOUR CODE HERE */;
```

Answer (code implementation):

....

Task 4.2. Compare performances of single buffer vs. double-buffering:

N.B. We assume the input to be squared, so only one spatial dimension is required

We already give to you the results for the single buffering (same as last laboratory!)

You can find the code for double buffering in this folder:

- `double-buffering/`

Fill the following table:

	Channels (C)	Spatial Dim. (N)	MAC	Cycles	MAC/cycles
single-buffer	32	32	1048576	138045	7.6
double-buffering	32	32			

Note: The formula to calculate the total number of MAC operations is:

$$MACs = Kernel\,Height * Kernel\,Width * Channels_{in} * Height_{out} * Width_{out} * Channels_{out}$$

Reply to the following questions

- How many cycles do we save with double-buffering? What's the speedup?
  Answer:
- What's the MAC/cycles improvement ?
  Answer:

Task 4.3. Find the minimum Tiling factor to fit L1 with double buffering:

- Find the minimum tiling factor for which the spatial dimension is divided to fit the layer in L1 (64kB).
  NOTE: Keep in mind we now use two buffers for the input and output feature maps, while we only store the kernel weights once.
- Compute memory occupation in L1

C are the channels,
N is the spatial size (N=input_width=input_height)

Dimensions (C, N)	Total L2 Memory Occupation	Tiling Factor	L1 occupation **Inputs-outputs**	L1 occupation **Weights**	L1 occupation **Total**	Cycles

16, 64						
32, 64						
64, 32						
128, 32						

(Optional Question):

- If you completed task3.3 from lab06 (which was still conv1x1, but without double buffering), which tiling factors are now different? Why?
  This is the old table

Task 3.3. Find the optimal Tiling factor to maximize performance:

- Test the four layers specified in the table.
- Try different tiling factor. Find the optimal one.

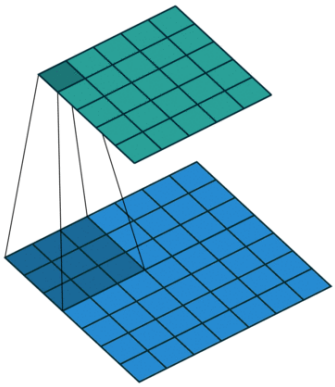
Dimensions (C, N)	Spatial Dim. L2	Spatial Dim. L1	Tiling Factor
16, 64	131328	33024	2
32, 64	263168	17408	4
64, 32	135168	36864	2
128, 32	278528	32768	4

Answer:

Error1: *when you* ***overflow the L1 memory*** *available you will get this:*

Exercise 5: Tiling with 3x3 convolution

3x3 convolution:



Task 5.1. Calculate convolution input sizes by hand

Let's assume PADDING=0 and STRIDE=1

In the case of 1x1 convolution, the input and output sizes were the same. With the kernel dimensions changed to 3x3, the output will be reduced. The 'C' (channels) variable will stay the same, but the 'N' (spatial) variable will now denote the **output spatial dimension**.

This is the formula

$$W_{out} = floor\left(\frac{W_{in} - k + 2P}{S} + 1\right)$$

$$H_{out} = floor\left(\frac{H_{in} - k + 2P}{S} + 1\right)$$

Fill out the table with the missing information: what's the input size that gets us the desired output size?

Output spatial dimension N	Kernel Height	Kernel Width	Input feature Height	Input feature Width

16	3	3		
32	3	3		
48	3	3		
64	3	3		

1. How did the input feature dimensions change? **Answer:**
2. If we changed the kernel size to 5x5, how would they change (just the amount)?
   Answer:
3. Write down the general formula to calculate the change for any kernel size.
   Answer:

Task 5.2. Tile overlapping

The receptive field in a 1x1 convolution is of the size 1x1 so there was no overlap between adjacent tiles. In the 3x3 convolution there is some overlap because of a larger receptive field.

Exercises:

1. If we look at adjacent tiles in one dimension (e.g. horizontal), how much do they overlap in that dimension? *N.B. Think about the border between tiles in the output and which pixels do they need from the input.*
   Task: Write down a generic formula for any kernel size.

   Answer

The current implementation doesn't account for this overlap when calculating the pointer for the input of the tile, so implement the code yourself:

2. Change the `tile_overlap_x_in` and `tile_overlap_y_in` calculation in the function `tile_load()` to account for the overlap.
   You can find the function in the file `double-buffering/Src/kernel.c`

```
54  const int tile_overlap_x_in = 0;  // /* YOUR CODE HERE */ - change this value,
55  const int tile_overlap_y_in = 0;  // /* YOUR CODE HERE */ - change this value,
```

Answer (code implementation)

Task 5.3. Find the optimal tiling factor for the 3x3 convolution to maximize L1 memory occupation:

- Test the four layers specified in the table. *N.B. N is the output spatial dimension*
- Before trying out a tiling factor, calculate the L1 memory occupation and see whether it will fit or not (try it out too).

Dimensions (C, N)	Total L2 Memory Occupation	Tiling Factor	L1 occupation **Inputs-outputs**	L1 occupation **Weights**	L1 occupation **Total**	Cycles
16, 64						
32, 64						
64, 32						
128, 32						

1. Which numbers changed compared to the 1x1 convolution from task 4.3? **Answer:**
2. Since we are not tiling channels, what is the largest channel size we can use so that the weights still fit into L1? (ignore the input, output, im2col buffers) **Answer**