# LAB8 APAI:

Tiling on PULP

***Credits***: *Alessio Burrello, Lorenzo Lamberti, Francesco Conti, Nazareno Bruschi, Davide Nadalini, Alberto Dequino. (University of Bologna)*

***Contacts***: *alessio.burrello@unibo.it*, *nazareno.bruschi@unibo.it*, *f.conti@unibo.it*

## How to deliver the assignment:

**DEADLINE: 23/12/2021 at 4PM**

**Instructions:**

- Use Virtuale platform to load your file: link
- update the C project, named the folder as follows: LAB7_APAI_yourname with the required screenshots

## Requirements:

- VirtualBox or VMWare
- pulp-box: https://github.com/EEESlab/APAI-LAB01-PULP_Embedded . Follow the README to download and import it

## Resources

- https://github.com/pulp-platform/pulp-sdk
- https://github.com/pulp-platform/pulp-sdk/tests
- https://github.com/pulp-platform/pulp-sdk/applications
- https://github.com/pulp-platform/pulp-sdk/rtos
- https://pulp-platform.org/
- https://greenwaves-technologies.com/manuals/BUILD/HOME/html/index.html
- https://greenwaves-technologies.com/manuals/BUILD/PMSIS_API/html/modules.html
- https://greenwaves-technologies.com/manuals/BUILD/PMSIS_BSP/html/index.html
- Material on GitHub

## Lab session summary:

1. Flash the external non-volatile memory with the useful data and parameters;
2. Read the memory and write data in the external volatile memory;
3. Use PULP cluster;
4. Use the performance counters;
5. Tile the input and output data to obtain the best performance from the execution.

# Task 0: Set-up, open and familiarize with C project

a) Open your pulp-box and set it up
   - open a new terminal
   - configure the environment with $source /pulp/sourceme.sh

b) Download from https://www.github.com/EEESlab/APAI-LAB08-Tiling-PULP the assignment using ssh and then compile and run it
   - $git clone git@github.com:EEESlab/APAI-LAB08-Tiling-PULP.git
   - $cd APAI-LAB08-Tiling-PULP
   - $make clean all run

# Task 1: Flash the external non-volatile memory (FLASH)

a) The application is using particular libraries to allow communications between external devices and the PULP chip. These are the ones related to BSP (Board Support Package) and to the filesystem manager
   - include *bsp/bsp.h* and *bsp/fs/readfs.h*

b) PULP-SDK (Software Development Kit) includes the Flasher, a tool to flash the memories with data in hex format. We should just add our intention to use the Flasher to the Makefile and then it will do all for us
   - Define and add the file names (using the relative path) to *FLASH_FILES* Make variable. Files are located in ./Inc/. Example: FLASH_FILE+=Inc/test.hex

# Task 2: Read the FLASH and write data the volatile external memory (RAM)

a) Initialize a new device that is going to describe the RAM (follow the same procedure for the FLASH above in the code), configure it with the specific parameters (chip-select, size, …) and then open it. *ram* device has been already defined in the global area
   - *pi_hyperram_conf_init*, *pi_open_from_conf* and *pi_ram_open*

b) You should allocate the arrays that you want to use in the application as for a normal memory area
   - *pi_ram_alloc*. All the args are already defined (L3_input, L3_…)

c) Copy the data from FLASH to RAM using bsp functions
   - *pi_ram_write*. fs_read reads a portion of the filesystem (128byte) and stores what it has read in flashBuffer. Write the content in the allocated region of RAM

d) Compile the application to check if the code is correct and does not contain errors. If you have any problem you can enable useful *printf*s removing the comment to *DEBUG* define. You can add more information if you want

## Task 3: Use PULP cluster

a) PULP cluster is a device and then you should initialize, configure and open it as for the others. *cluster* struct is already defined in the global area
   - *pi_cluster_conf_init*, *pi_open_from_conf*, *pi_cluster_open*

b) The application to execute on the cluster side is called cluster task. Delegate a cluster initialization to prepare useful structures that describe the layer to be computed
   - *pi_cluster_send_task_to_cl*. Send the function *cluster_init*

c) Compile the application to check if the code is correct and does not contain errors

## Task 4: Use the performance counters

a) PULP cores are augmented with a set of hardware counters to assess some characteristics such as the number of executed instructions, load stalls and branch instructions. Activate and start them on the FC side. YOU MUST ACTIVATE AT LEAST THE NUMBER OF TOTAL CYCLES. Example pi_perf_conf(1 << WHAT_I_WANT_TO_MEASURE)
   - *pi_perf_conf, pi_perf_reset, pi_perf_start*

b) Send the application to the cluster, which is called *layer_run* and after the execution put the evaluation point for the counters at the end of the cluster offloading. Print what they have counted to see the performance
   - *pi_perf_stop, pi_perf_read*

c) Compile the application to check if the code is correct and does not contain errors

## Task 5: Tile the input and the output data and obtain the best performance from the execution

a) The entire layer, now in L2, is bigger than the available L1. TILE IT! To keep it simpler, tile just along H_in, W_in, H_out and W_out. DO NOT TILE CHANNELS.
   - in *cluster_init* select the sizes and assign them to the local structure (*I1_layer*). Try with ⅛, ⅙ and ¼ of the global ones.

b) Implement the tiling loop in the main function (*layer_run*)
   - define the bounds and write the two nested loops

c) Program the DMA to transfer input (*kernel_init*) and output (*kernel_end*) data in L1. To keep it simpler use blocking transfers
   - *pi_cl_dma_cmd_2d*. 2d transfers allow access to the memory like a rectangular matrix and not sequentially. Example:

```
pi_cl_dma_cmd_2d(
  L2_input + tile_index * tile_dim,
  L1_input,
  tot_transfer_size,
  how_many_lines_to_skip,
  size_before_skip_the_line,
  PI_CL_DMA_DIR_EXT2LOC, //or LOC2EXT
  &dma_cmd);
```

d) Profile the execution and see if it is functionally working.
   - Compile and run the final application and take screenshots of the results in the standard output for each tiling size case.

e) What happens if you choose an odd number as a tiling factor? Try with 1/5
   - Propose a solution including leftovers in the tiling loops