APAI2025 - LAB04  
PULP\_NN

short line

*Authors: Davide Nadalini, Alberto Dequino, Francesco Conti*

*Contacts: d.nadalini@unibo.it, alberto.dequino@unibo.it*

***Links:*** [***GitHub Link (code)***](https://github.com/EEESlab/APAI25-LAB04-PULP-NN)

# **Summary**

1. Subject(s):
   * Parallelization on the PULP architecture
   * Matrix-multiplication
   * 2D conv
   * profiling code execution
2. Programming Language: C
3. Lab duration: 3h
4. 4. Assignment:
   * Time for delivery: 1 week
   * **Submission deadline: Oct 31st 2025**

# 

# **How to deliver the assignment**

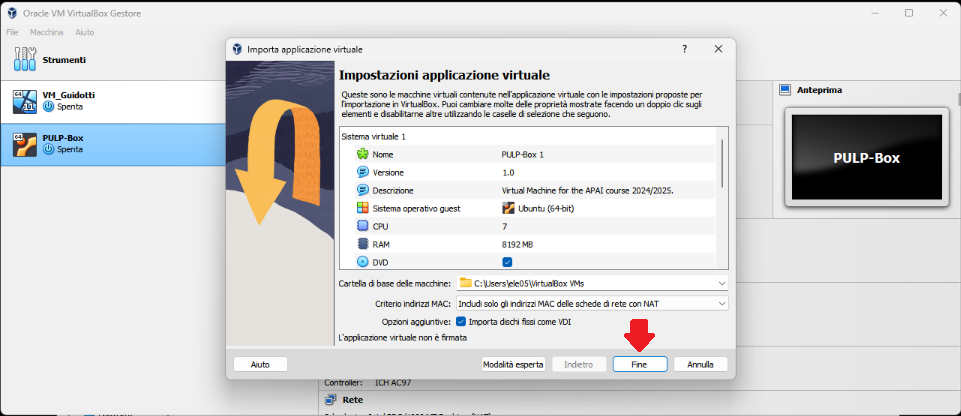
You will deliver ONLY THIS TEXT FILE, no code

* Download this file.
* Fill in the required results.
* 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

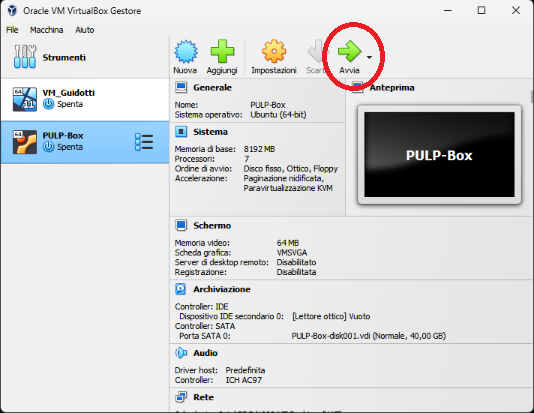
# **LAB STARTS HERE**

# **0. (ONLY LAB1) Access to the local VM**

* On the lab’s PCs, open the file explorer and go to This PC, C:/VM\_APAI
* Double click on PULP-box.ova
* VirtualBox opens, just click on “Fine”

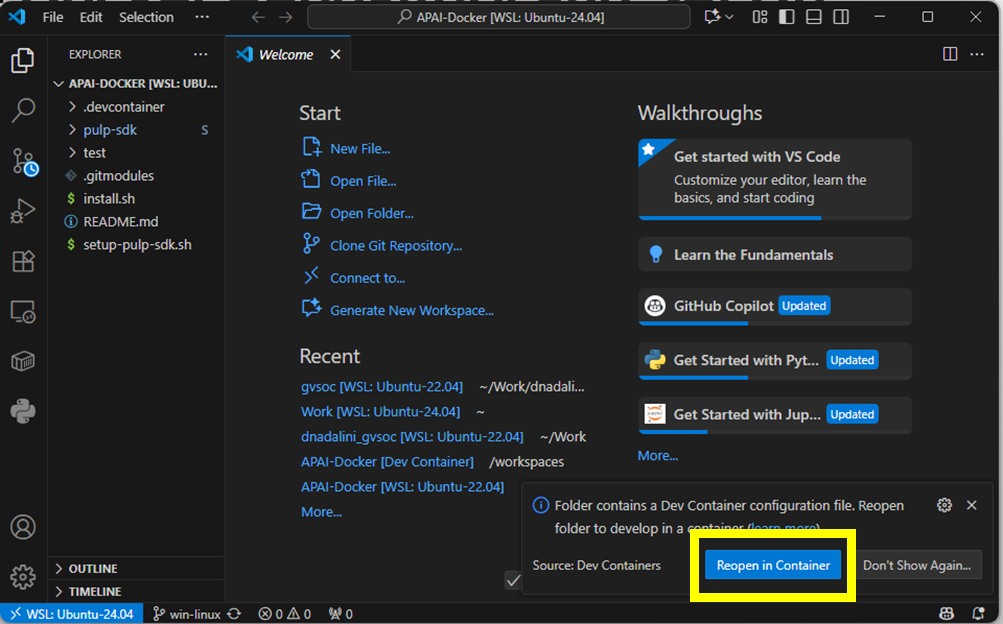


* Wait for the VM to be imported
* Open the VM with “Avvia”

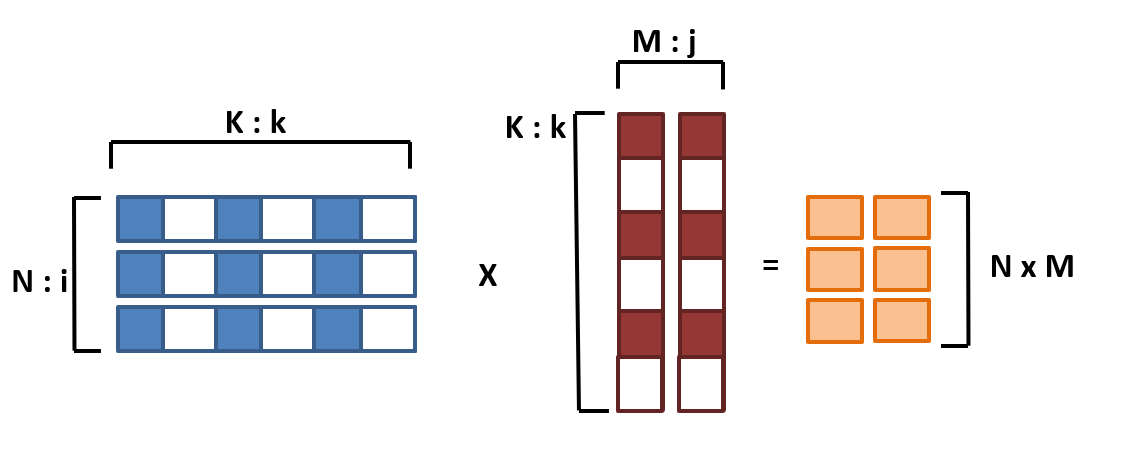


* Password is ‘pulp’

# **1. (ALL) Open Docker Container with VSCode**

* Open a terminal (right click – open a new terminal)
* Go in the APAI-Docker folder: cd APAI-Docker
* Open Docker with VSCode: code .
* Select the pop-up option “Reopen in container”
* Setup PULP-SDK: source setup-pulp-sdk.sh
* Clone GitHub repository of today’s lab: git clone https://github.com/EEESlab/APAI25-LAB01-PULP-Embedded-Programming
* cd APAI25-LAB01-PULP-Embedded-Programming
* cd <folder\_you\_want>
* make clean all run

# **Task 1: matrix-multiplication**

****

**Matrices sizes: N x K \* K x M = N x M**

**Initial sizes: N=32, M=16, K=16**

### 

### 1.0. Setup:

* Open VSCode (*code .*)
* Go to matmul\_parallelization/ folder
* Every time you want to run the code, **SAVE your file** and write in the terminal :

make clean all run

* Don’t forget to source the sdk on new terminals:

source setup\_pulp\_sdk.sh

### 1.1. SPEED-UP and Amdhal's law:

*Task Location: matmul\_parallelization/cluster.c*

**Setup**: N=32, M=16, K=16

**Sub-tasks:**

* Enable performance counters. You fill find them in the code → uncomment them
* Fill table below: *Calculate execution cycles, and calculate speedup w.r.t. using only one core.*

***Tip****: for executing with different numbers of cores, use the “CORES” flag.*

*Example: make clean all run CORES=8*

|  |  |  |
| --- | --- | --- |
| CORES | Cycles | Speedup (w.r.t. CORES=1) |
| 1 |  |  |
| 2 |  |  |
| 4 |  |  |
| 8 |  |  |

### 1.2. Explore different input sizes:

*Task Location: matmul\_parallelization/cluster.c*

**Setup**:

* 8 CORES
* N=<varying> M=16 K=16

**Sub-tasks:**

* measure performance of each individual core *(Execution cycles, and IPC)*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Matrix size: N | Instructions executed (each individual core) | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |
| 81 |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Matrix size: N | IPC (each individual core) | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |
| 81 |  |  |  |  |  |  |  |  |

Answer the following questions:

* Is the workload equally balanced with N=4? Why ?
* Is the workload equally balanced with N=8? Why ?
* Is the IPC higher for N=80 or N=81? Why ?

**[ANSWER]:**

* measure the overall performance of the GEMM: Cycles, MACs, MAC/Cycles

***Tip:*** *Calculate the MACs by hand*

|  |  |  |  |
| --- | --- | --- | --- |
| N | MACs | Cycles (equal to all cores) | MAC/cycles (equal to all cores) |
| 4 |  |  |  |
| 8 |  |  |  |
| 80 |  |  |  |
| 81 |  |  |  |

Answer the following questions:

* Why when N=81 the MAC/cycles is lower than when N=80?

**[ANSWER]:**

### 1.3. Load stalls & unrolling the MatMul

*Task Location:*

* *matmul\_parallelization/cluster.c*
* *matmul\_parallelization/matmuls.c*

**Setup**:

* 8 CORES
* N=32 M=16 K=16

**Sub-task:**

* Enable the performance counters of our interest. We want to profile:
  + Execution cycles (total)
  + N° instructions executed
  + Load Stalls → missing

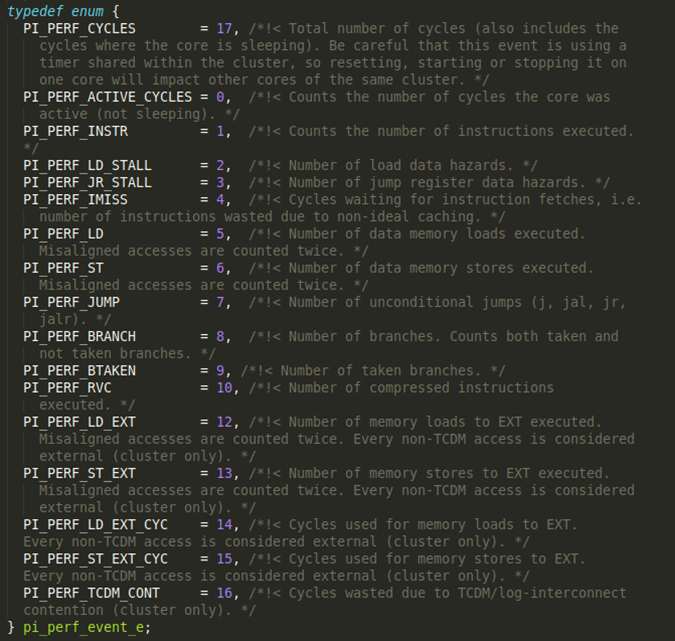
Complete the code where you find /\* YOUR CODE HERE \*/ with the right performance counters



Put your solution below (code)

**[HERE]**

***Tip****: Here’s the full list of the performance counters*



*Ref: /rtos/pmsis/pmsis\_api/include/pmsis/chips/default.h*

* Implement missing code on the gemm\_unroll(). Then manually fix the code in order to unroll 2-4-8-16 operations.
* Fill in now the table, comparing the naive gemm() vs. the unrolled version gemm\_unroll()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **stalls** | **Instructions** | **Cycles** | **IPC** | **MACs** | **MAC/Cycles** |
| naive |  |  |  |  |  |  |
| Unrolled 2 |  |  |  |  |  |  |
| Unrolled 4 |  |  |  |  |  |  |
| Unrolled 8 |  |  |  |  |  |  |
| Unrolled 16 |  |  |  |  |  |  |

Answer the following questions:

* Compare the stalls of “naive” vs. unrolled (2-4-8). Who has more? Why?
* Why Unrolled16 has more stalls than Unrolled8?

**[ANSWER]:**

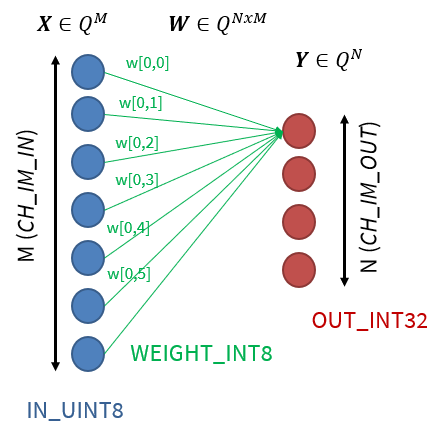
### 1.4. (optional) Data Reuse

Read instructions inside the code

|  |  |  |
| --- | --- | --- |
| **FC** | **IPC** | **MACs/cycle** |
| naive |  |  |
| Reuse 2 |  |  |
| Reuse 4 |  |  |
| Reuse 8 |  |  |

# 

# **Task 2: Fully Connected Layer**

****

### 2.0. Setup:

* Open VSCode.
* Go to **fully\_connected**/ folder
* Every time you want to run the code, **SAVE your file** and write in the terminal :

make clean all run

* Don’t forget to source the sdk on new terminals:

source setup\_pulp\_sdk.sh

### 2.1. MACs: FullyConnected without vs. with SIMD

Compare the number of executed instructions in the normal FullyConnected layer and the SIMD version.

1. Calculate the number of MACs by hand
2. Use perf counter to measure the number of executed instructions
3. Change the *dotp\_u8\_i8\_i32* function to the *dotp\_u8\_i8\_i32\_simd* and measure the number of executed instructions again

**Note:** the number of executed MAC operations is dictated by the size of the FullyConnected layer, not the way operations are implemented (non-SIMD/SIMD). The formula to calculate the total number of MAC operations is:

You can find the FullyConnected layer dimensions in the *data\_allocation.h* header file.

**TIP:** 8-bit SIMD instructions perform 4 MACs in 1 cycle

|  |  |  |  |
| --- | --- | --- | --- |
| **Cores (#N)** | **SIMD** | **Operations (MAC)** | **Instructions (#N)** |
| 1 | No |  |  |
| 1 | Yes |  |  |

What is the ratio of instructions before/after the new SIMD implementation? Why?

**[ANSWER]:**

### 2.2. Calculate speedup

Measure the latency of a FullyConnected layer and fill out the table below.

From the measured latency calculate the performance and speedup with regard to the single core latency.

**Note:** to calculate the performance you will have to divide the total number of MAC operations with the measured latency.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cores (#N)** | **SIMD** | **Latency (cycles)** | **Performance (MAC/cycle)** | **Speedup w.r.t #Cores=1 SIMD=No** |
| 1 | No |  |  | 1.0x |
| 1 | Yes |  |  |  |
| 8 | No |  |  |  |
| 8 | Yes |  |  |  |

# **Task 3: Convolution Layer**

### 3.0. Setup

* Open VSCode.
* Go to **convolution/** folder
* Every time you want to run the code, **SAVE your file** and write in the terminal :

make clean all run

* Don’t forget to source the sdk on new terminals:

source setup\_pulp\_sdk.sh

### 3.1. Speedup over multiple cores

Measure the latency of a Convolution layer and fill out the table below. From the measured latency calculate the performance and speedup with regard to the single core latency.

**Note:** to calculate the performance you will have to divide the total number of MAC operations with the measured latency. The formula to calculate the total number of MAC operations is:

You can find the Convolution layer dimensions in the *data\_allocation.h* header file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cores (#N)** | **MACs** | **Latency (cycles)** | **Performance (MAC/cycle)** | **Speedup w.r.t #Cores=1 SIMD=No** |
| 1 |  |  |  | 1.0x |
| 2 |  |  |  |  |
| 4 |  |  |  |  |
| 8 |  |  |  |  |