APAI2023 - LAB04

PULP_NN

Authors: Lorenzo Lamberti, Nazareno Bruschi, Davide Nadalini, Luka Macan, Alberto Dequino, Manuele Rusci, Francesco Conti
Contacts: lorenzo.lamberti@unibo.it, d.nadalini@unibo.it, alberto.dequino@unibo.it

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

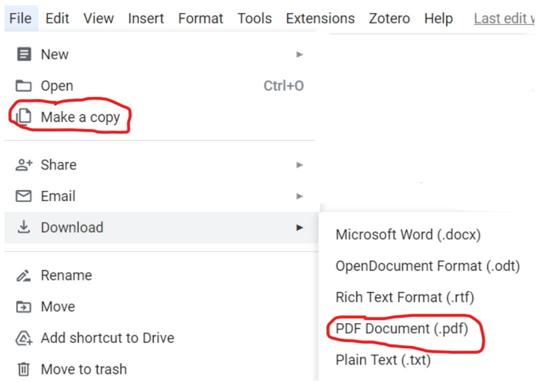
Summary

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: Nov 2nd 2023**

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#1>_<your_name#2>.pdf
- If you are in a group with > 1 people, only deliver 1 file
- Use Virtuale platform to load ONLY your .pdf file



LAB STARTS HERE

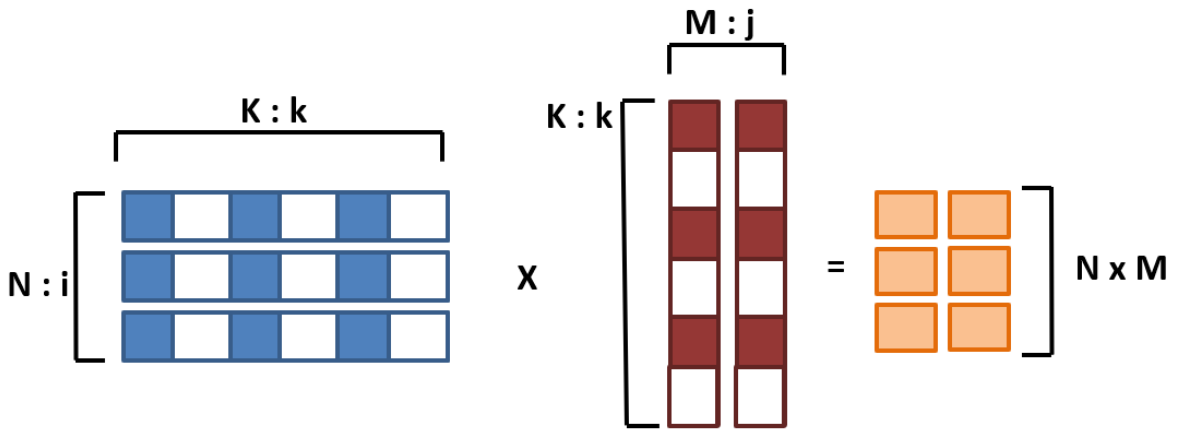
0. Access to the remote server, and setup

- Open this web page: https://compute.eees.dei.unibo.it:8443/guacamole/ **(works only from ALMA WIFI NETWORK!)**
- Login. We distribute credentials by hand.
- Open a terminal (right click – open a new terminal)
- Clone: `git clone https://github.com/EEESlab/APAI23-LAB05-PULP_NN.git`
- `module load pulp-sdk`

Download the repo

```
cd <work_dir>
git clone https://github.com/EEESlab/APAI23-LAB04-PULP-NN.git
```

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.
- 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:
  `module load pulp-sdk`

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 number 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

```
pi_perf_conf(
    1 << PI_PERF_CYCLES
    | 1 << PI_PERF_INSTR
    // /*YOUR CODE HERE*/ //(add load stall)
);
```

Put your solution below (code)

[HERE]

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

```
typedef enum {
  PI_PERF_CYCLES        = 17, /*!< Total number of cycles (also includes the
    cycles where the core is sleeping). Be careful that this event is using a
    timer shared within the cluster, so resetting, starting or stopping it on
    one core will impact other cores of the same cluster. */
  PI_PERF_ACTIVE_CYCLES = 0,  /*!< Counts the number of cycles the core was
    active (not sleeping). */
  PI_PERF_INSTR         = 1,  /*!< Counts the number of instructions executed.
  */
  PI_PERF_LD_STALL      = 2,  /*!< Number of load data hazards. */
  PI_PERF_JR_STALL      = 3,  /*!< Number of jump register data hazards. */
  PI_PERF_IMISS         = 4,  /*!< Cycles waiting for instruction fetches, i.e.
    number of instructions wasted due to non-ideal caching. */
  PI_PERF_LD            = 5,  /*!< Number of data memory loads executed.
    Misaligned accesses are counted twice. */
  PI_PERF_ST            = 6,  /*!< Number of data memory stores executed.
    Misaligned accesses are counted twice. */
  PI_PERF_JUMP          = 7,  /*!< Number of unconditional jumps (j, jal, jr,
    jalr). */
  PI_PERF_BRANCH        = 8,  /*!< Number of branches. Counts both taken and
    not taken branches. */
  PI_PERF_BTAKEN        = 9, /*!< Number of taken branches. */
  PI_PERF_RVC           = 10, /*!< Number of compressed instructions
    executed. */
  PI_PERF_LD_EXT        = 12, /*!< Number of memory loads to EXT executed.
    Misaligned accesses are counted twice. Every non-TCDM access is considered
    external (cluster only). */
  PI_PERF_ST_EXT        = 13, /*!< Number of memory stores to EXT executed.
    Misaligned accesses are counted twice. Every non-TCDM access is considered
    external (cluster only). */
  PI_PERF_LD_EXT_CYC    = 14, /*!< Cycles used for memory loads to EXT.
  Every non-TCDM access is considered external (cluster only). */
  PI_PERF_ST_EXT_CYC    = 15, /*!< Cycles used for memory stores to EXT.
  Every non-TCDM access is considered external (cluster only). */
  PI_PERF_TCDM_CONT     = 16, /*!< Cycles wasted due to TCDM/log-interconnect
  contention (cluster only). */
} pi_perf_event_e;
```

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?

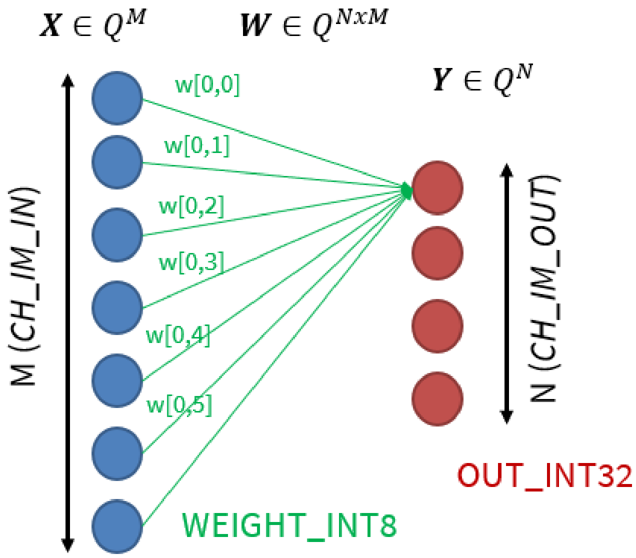
[ANSWER]:

1.4. (optional) Data Reuse

Read instructions inside the code

FC	IPC	MACs/cycle
`naive`		
`Reuse 2`		
`Reuse 4`		
`Reusue 8`		

Task 2: Fully Connected Layer



IN_UINT8

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:
  `module load pulp-sdk`

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:

$$MACs = Channels_{in} * Channels_{out}$$

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 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			

2.3. Implement missing code

GOAL: We want to count how many dotproducts operations we perform

Put your solution below (code)

[HERE]

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:
  `module load pulp-sdk`

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 divide the total number of MAC operations with the measured latency. The formula to calculate the total number of MAC operations is:

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

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				