Final assignment of "Management and Analysis of Physics Datasets"- Module B

June 18, 2021

Gerardo Carmona

Saverio Monaco You might have seen this kind of puzzles on social media. It should be straightforward for you to understand which operations connect the two numbers before the '=' sign. What is the result of the **?** field? Show proof for each line!

$$
\begin{aligned}
1+2 &= 3 \\
2+5 &= 7 \\
3+7 &= 4 \\
4+5 &= \mathbf{?} \\
5+9 &= 12
\end{aligned}
$$

SOLUTION If we rewrite the digits as binary, we can easily guess the actual operation:

$$
\begin{array}{lcll}
1+2=3 & : & f(01,10) & = 11 \\
2+5=7 & : & f(010,101) & = 111 \\
3+7=4 & : & f(011,111) & = 100 \\
4+5=? & : & f(100,101) & = ? \\
5+9=12 & : & f(0101,1001) & = 1100
\end{array}
$$

For all expression, we can replace $f(a,b)$ with $a \oplus b$, thus obtaining $100 \oplus 101 = 001 = 1$ for "$4+5$".

1 Redundancy

We are programming a file based RAID-4 software algorithm. For this purpose we are converting a single input (**raid4.input**) file into four data files `raid4.0`,`raid4.1`,`raid4.2`,`raid4.3` and one parity file `raid4.4` - the four data and one parity file we call 'stripe files'.

The input file can be downloaded from: http://apeters.web.cern.ch/apeters/pd2021/raid4.input

To do this we are reading in a loop sequentially blocks of four bytes from the input file until the whole file is read: * in each loop we write one of the four read bytes round-robin to each data file, compute the parity of the four input bytes and write the result into the fifth parity file. (see the drawing for better understanding)

- we continue until all input data has been read. If the last bytes read from the input file are not filling four bytes, we consider the missing bytes as zero for the parity computation.

Input File (horizontal) **raid4.input - total size 170619 bytes** (number in cell = byte offset in file)

0	1	2	3	4	5	6	7	8	9	10	11	12	...	...	170618

Output File (vertical) (number in cell = byte offset in file, p0,1,2... are the row-wise parities)

raid4.0	raid4.1	raid4.2	raid4.3	raid4.4
0	1	2	3	p0
4	5	6	7	p1
8	9	10	11	p2
12	13	14	15	p3
...	...	...	...	...

Stripe parity (column wise parity)

q0=0^4^8^12	q1	q2	q3	q4

[1]:
```
import numpy as np

def get_arr(path):
    '''Reads a file as bytes'''
    with open(path,'r+b') as file:
        v = np.frombuffer(file.read(),dtype = np.uint8)
    return v


def post_arr(arr, path):
    '''Writes an array to a file, converting it to bytes'''
    with open(path,'w+b') as file:
        file.write(arr.tobytes())
    return


def xor_arr(matrix):
    '''Applies the bitwise xor function to a collection of arrays, given as rows␣
 ↪in a matrix.'''
    # Neutral xor array
    v = np.zeros(len(matrix[0]),dtype = np.uint8)
    # Applying bitwise_xor between the matrix rows
    for arr in matrix:
```

```
        v = np.bitwise_xor(v,arr)
    return v
```

1.1 Write a program (C,C++, R or Python), which produces four striped data and one parity file as described above using the given input file.

hint: if you have a problem programming this yourself, you can download the core program in C++ from http://apeters.web.cern.ch/apeters/pd2021/raid4.cSee the explanations in the beginning how to compile and run it. You have to add the parity computations at the IMPLEMENT THIS sections! If you can't compile or run it, you can still fill in the missing implementation!

[2]:
```
n = 4
src_file = 'raid4.input'
all_bytes = get_arr(src_file)
all_bytes

# We calculate how many zeros have to be added in the end
extra = n - len(all_bytes)%n
bytes_arr = np.hstack([all_bytes,np.zeros(extra, dtype = np.uint8)])

# Reshaping array
nrows = len(bytes_arr)//n
ncols = n
bytes_arr = bytes_arr.reshape((nrows,ncols))
## Rendered as unsigned integers of 8 bits.
bytes_arr
```

[2]:
```
array([[ 37,  80,  68,  70],
       [ 45,  49,  46,  51],
       [ 10,  37, 196, 229],
       ...,
       [ 55,  55,  51,  55],
       [ 10,  37,  37,  69],
       [ 79,  70,  10,   0]], dtype=uint8)
```

[3]:
```
# Iterate over the columns and generate the files
for i, v in enumerate(bytes_arr.T):
    post_arr(v, f'raid{n}.{i}')

# Evaluating parity between stripes
row_p = xor_arr(bytes_arr.T)

#Store the row parity
post_arr(row_p,f'raid{n}.{n}')
```

[4]:
```
## Unsigned 8bit integer representation of the parity stripe
row_p.T
```