TDT4102 — CMB Challenge 2023

Version 1.2

Sofiya Sianiuta, Maja Grauff Erntsen, Erling Syversveen Lie, Lasse Natvig

March 2023

Contents

1 Introduction

1.1 The Climbing Mont Blanc Project

Climbing Mont Blanc (CMB) is an open online judge used for training in energy efficient programming of state-of-the-art heterogeneous multi-cores. It uses a three-way heterogeneous processor containing 14 different cores of three different types. The system supports C++ (C++14 standard) and C (C99 standard). The source files are compiled using the GNU G++ compiler version 7.3, with pthreads, OpenMP v4.5 and OpenCL v1.2 support. The -O2 optimization flag is used. Uploaded programs are evaluated with respect to time, energy used, and energy-efficiency (EDP). Many online programming judges exist, but we are not aware of any similar system that also reports energy-efficiency.

Figure 1 shows the back end of the CMB system in action. It is an Odroid XU3 board[1] with a Samsung Exynos 5 Octa SoC (System-on-Chip)[2] with integrated power sensors. It contains 4 ARM Cortex™-A15 cores at 2 Ghz and 4 ARM Cortex™-A7 cores at 1.3 GHz, an ARM Mali-T628 GPU with 6 cores, and 2 GB RAM at 933 MHz. It is accessed via the CMB server and a graphical user interface available at `https://climb.idi.ntnu.no`

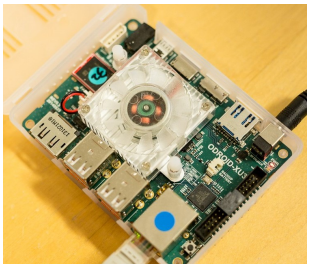

Figure 1: Odroid XU3 used to execute uploaded CMB programs.

1.2 Project Webpage, Documentation and Support

The main official webpage of the CMB project is:

`https://www.ntnu.edu/idi/lab/cal/cmb`

At the end of the descriptional text at that webpage there is a link to more documentation, including a very brief getting-started-guide in English and an short getting-started video in Norwegian. We will, as usual, try to give excellent support to users of the system. Students taking part in the CMB Challenge 2023 can contact us via the TDT4102 Piazza-class (please use the tag CMB), or by e-mail to Lasse Natvig.

The CMB system has been developed by students as part of their project and master thesis work. (The Mali GPU used in the Exynos chips does also origin from NTNU students[3].)

[1] Odroid-XU3 Wiki, http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu3

[2] Exynos Wiki, https://en.wikipedia.org/wiki/Exynos

[3] A Brief History of Mali,
https://www.anandtech.com/show/8234/arms-mali-midgard-architecture-explored/2

1.3 CMB Webpage

In order to participate in the competition you have to sign up on the CMB web-site (https://climb.idi.ntnu.no/) accessible on eduroam networks and through NTNU VPN from home), and join the group "TDT4102 CMB Challenge 2023". **Note that the code you upload can be used in anonymised form for research purposes.** If you need some help to get started, see the getting started guides mentioned above and the **the How To tab** at the CMB web-site.

If you experience technical difficulties with the system, e.g. system crash or messages that ask you to contact the system administrator, you can send an email to Lasse@computer.org — please give enough details (username, problem name, code) about your problem in order for us to help you faster. Thanks for helping to improve the CMB system!

1.4 Scoring Rules

a. 1 point for every problem with a valid solution.

b. 3 points for the fastest solution to a problem, 2 points for the second fastest and 1 point for the third fastest solution to a problem (exceptions, see below).

c. 3 points for the most energy-efficient solution to a problem (measured by the EDP-value), 2 points for the second best and 1 point for the third most energy-efficient solution (exceptions see below).

d. Points can be shared among multiple students if the difference in one of the competition metrics is smaller than the precision of that CMB metric. E.g., if no 1 and no 2 are very close, the judges might decide that both will get $(3 + 2)/2 = 2.5$ points.

e. Some problems give points only for a valid solution, and the last problem give *bonus points for solutions that maximize student happiness!* (See section 2.7)

1.5 Technical Note

The system will test your uploaded solution by running two separate test cases, one big and one small test case. The energy and time measurements are performed while running the program on the big test case. The results will be displayed if you pass both test cases. Please note that we might change the test cases during the competition, so make sure that your solution works on a general case!

The CMB system runs on an Ubuntu Linux system and compiles the code using a gcc/g++ compiler. The compiler **uses the C++14 standard**, so code using C++17/20 features will *not* compile. (Further details at the How-To-page).

The problems and solutions have been tested on different computers, and your code should work well if you follow the C++14 standard and have included all necessary headers (include too many, rather than too few). The std_lib_facilities.h-header used in TDT4102 is *not* available in the system, so you will receive compilation errors if you try to use it.

To avoid possibilities of cheating, the feedback from the CMB system is short and sometime to little help. Therefore, it is wise to test your code locally on the given examples before uploading — and remember that you can ask for help. In case we find weaknesses we might publish a revised version of this document or inform all students via piazza and blackboard.

2 Problems for CMB Challenge 2023

The competition is only for students in the TDT4102 course spring 2023. The different problems are independent of each other. Their order follow a "story" at campus and is more or less in increasing order of difficulty. You are free to choose the order in which you solve the problems.

2.1 Sleepy student

Level of difficulty: ★
After spending the whole week doing voluntarily work at the Studentersamfundet, you fall asleep with a joyful feeling of content and fulfilment of your social needs. You have had great fun, so your dreams are sweet, well in fact they are cute, puffy, tiny little sheep. You see them hopping up and down, one.., twooo , threeee.

2.1.1 Task

You count 100 sheep in total. The output is `i sheep`, written one hundred times, line by line. `i` is the count of each sheep. Write the output to STDOUT in the format shown in the example. (There is no input for this task).

2.1.2 Example

Output:

```
1 sheep
2 sheep
3 sheep
...
100 sheep
```

2.1.3 Note

This task is not evaluated on time or energy. You will only need to produce a valid solution, and will be given no extra points for optimizing the task.

2.2 Call from rector

Level of difficulty: ★
Rrrrrriiiiiing!!!
Suddenly all the sheep disappear and you wake up abruptly. Rector Anne Borg is at the other end of the call. The feeling of being star-struck quickly disappears as you realize that she is shaky and upset about something. An important report is missing from her office and someone has left a strange letter next to where it was earlier. The rector contacted you as she has seen this as a problem only you, with your tremendous programming skills, can help her solve.

2.2.1 Letter

```
5
Cd oczmz, kvmoizm.
D rmjoz di ocdn ajmh nj ocvo ocz kmdixdkgz rdgg ijo wpno jpm
hznnvbz viy ocz diajmhvodji rdgg novt wzorzzi pn.
Nj avm D cvqz mzhjqzy vgg kmjja ja ocvo D cvqz wzzi czmz.
Ocz jigt dnnpz D cvy rvn oj mzhjqz ht odhznovhk amjh ocz wpdgydib.
D omdzy jqzmgjvydib ocz ntnozh nj ocvo oczt rdgg izqzm wjoczm
gjjfdib ocmjpbc vgg.
Hzzo hz vo jpm mzbpgvm kgvxz, viy D rdgg nzz tjp gvozm. Ocvifn.
```

2.2.2 Task

From the expectation that the person writing this letter could have several other letters laying around at campus, the rector wants you to write a program which can translate this type of coded letters in general. The first line of the input explains the *shift* as a number needed to decipher the letter. The rest of the lines is the letter itself, where the letters needs to be shifted. It is only the letters 'a' – 'z' and 'A' – 'Z' that are shifted. As an example, a shift of the letter 'A' of value 2 gives 'C'. The shift wraps around, so a shift of 'z' with value 1 gives 'a'.

2.2.3 Example

Input:

```
1
Sgzmj xnt enq ozqshbhozshmf hm BLA 2023!
```

Output:

```
Thank you for participating in CMB 2023!
```

2.2.4 Note

This task is not evaluated on time or energy. You will only need to produce a valid solution, and will be given no extra points for optimizing the task.

2.3 Timestamp validator

Level of difficulty: ★
Following the hints from the previous letter you find out that you want to check the IDs for the people that has scanned their card into the building door locks today. The file containing all of the timestamps for that day is secured such that no one can remove their timestamp. Unfortunately, it seems as though a hacker has been able to fill the file with several false timestamps. However, the hacker was either in much hurry, or very lazy in the process, so the code generating the false timestamps does not take the correct syntax convention into account — a miss that you want to exploit. The convention for a valid timestamp is: `DDMMYYNNXXZZ` where the format is as follows:

`DD` - date, range: $01 \leq$ `DD` ≤ 30
`MM` - month, range: $01 \leq$ `MM` ≤ 12
`YY` - year, range: $00 \leq$ `YY` ≤ 99
`NN` - initials, range: A-Z
`XX` - hour, range: $00 \leq$ `XX` ≤ 23
`ZZ` - minutes, range: $00 \leq$ `ZZ` ≤ 59

We assume for simplicity that the days span from `01` to `30` regardless of the month, and that only years between 1924 and 2023 will be accepted. Also, only uppercase letters A-Z are considered valid. You are zooming in on the suspect, so be careful!

2.3.1 Task

You may assume that all IDs are listed with one ID per line. You should read the input from STDIN. Your task is to count all the invalid IDs, and all the valid IDs, using the given convention. The output is two lines; one with the number of valid IDs followed by one with the number of invalid IDs in the input file. You should write the output to STDOUT.

2.3.2 Example I

Input:
```
210297AR1730
430297BBR1630
21029eth77
190501SS0000
```

Output:
```
2
2
```

2.3.3 Example II

Input:
```
350701CTE2730
031268km1355
031268KM1355
10577NM0559
010577NM0559
```

Output:
```
2
3
```

2.4 Code cracker

Level of difficulty: ★★
After the thief had broken into the room, the person had to break through a lock before getting access to the valuable item. In order to gain knowledge about the burglary, you want to figure out how long time the thief had to turn the lock when you know which combination it had initially. From this knowledge it is possible to calculate how long time the thief minimum had to use while in the room.

2.4.1 Task

The lock has **N** different digits where you have to turn each digit, one position at a time. Each digit has to be rotated separately, and from 0 you can move both to 1 and 9. Given an **N** digit solution and a **N** digit starting combination, what is the minimum number of changes needed before getting the right code? One turn of the lock equals to one change, so 1 - 3 is 2 changes and 0 - 7 is three. You can assume that **N** is smaller than 6000000.

2.4.2 Input

First line is the solution, second line the starting combination. You should read the input from STDIN.

2.4.3 Output

The minimum required steps. The answer should be written to the console output with STDOUT.

2.4.4 Example

Input:

```
106517
021263
```

Output:

```
20
```

2.5 Calendar hot spot

Level of difficulty: ★★

Your lightning sharp intuition tells you that it could be valuable to inspect the camera footage from the rector's office. However, you have been informed that gaining such access could take some time, as both GDPR and security risks must be considered. Tired of being bored with waiting for such slow, bureaucratic processes, you decide to prepare your search a bit.

Even though the thief was a bit lazy with generating false timestamps, it could be reasonable to think that the person wanted to falsify a huge number of timestamps in the same time interval as when the mischief was conducted in the rector's office. This would make it more difficult to identify the right ID in the list of all timestamps, and thus reduce the list of suspects. You decide to go through the huge timestamp file that you analyzed earlier, and check what time period is the most busy — i.e. with most timestamps. Hopefully, the camera footages will be ready for you by then, and you can jump right to the timeslot of the crime.

2.5.1 Task

In this task you get an unknown number of valid timestamps from one day. From this you should find out which *hour* within that day contains the most timestamps in total. An *hour* in this context could start at any 10'th minute, that is from :00, :10, :20, :30, :40, or :50. The hour is calculated with the first minute being added, so that for example 17:00 to 17:59 is calculated as one entire hour.

2.5.2 Input

The input is read from STDIN and is a .txt file where all IDs are listed with one ID per line. All the IDs listed will be valid timestamps with the original format as described in subsection 2.3.

2.5.3 Output

The first line should be the number of timestamps in the most busy hour (period of 60 minutes) as defined under Task above. The second line should be the starting point of the hour that was busy, and the example below shows the format.

2.5.4 Example

Input:

```
000086GA1709
130408PJ1628
250585UC0240
221187AQ0410
210781ZM0629
130991JR0155
240871UP1955
260080BB1835
100113KW1616
050873TH2108
```

Output:

```
3
16:10
```

2.6 Mystery solver

Level of difficulty: ★★

As a good detective, you have constructed a table for each of your suspects to solve the mystery. It incorporates a score on means, motive, opportunity and other critical parameters. But you know much better than simply comparing the results amongst each other by hand. As a big guru of effectivization and optimization algorithms, you lay down a plan for finding the suspect, which essentially boils down to multiplication of matrices.

Each matrix represents a bunch of given suspects, one per row, and the columns represent the crime-related parameters. Each suspect is given a score on each parameter. Who has most motive, means and opportunity? Well, you are just about to solve the mystery.

2.6.1 Input and task

The input has **N** + 1 lines. First line contains two integers **N** and **S**, where **N** is the number of matrices in the product and **S** is the seed to the pseudo-random number generator. The next N lines contain two integers: **r** - the number of rows and **c** - the number of columns. The order of lines corresponds to the order of multiplication, so the first line with dimensions corresponds to the first (rightmost) matrix in the product.

Generate **N** matrices $\{A_1, A_2, ..., A_N\}$, with corresponding dimensions from input, assuming that the dimensions $p, q \in [1, 50000]$ — and multiply them $\mathbf{M} = A_N A_{N-1} A_{N-2} ... A_1$. The matrices must be filled with random 32-bit floating point numbers in the range $[-0.5, 0.5]$, generated sequentially, matrix by matrix, and for each matrix row by row. For example, 2 x 2 matrix must be generated in the following order: `(1,1)`, `(1,2)`, `(2,1)`, `(2,2)`.

You are supposed to use the function

```
int get_random(int from, int to)
```

described under the CMB HowTo tab to generate the random floating point numbers needed to fill the matrices. You may use the sentence

```
float realNum =  0.5 - 1.0 * (get_random(0,100)/100.0);
```

to generate the numbers.

Note that in order to produce the correct result, you must initialize the matrices in the order of multiplication. I.e. begin to fill A_1 with random numbers, then A_2 and A_3 and so on.

For the big test case hidden in CMB you can assume that both dimensions of all matrices are multiples of 8 (This assumption might be useful if you decide to use tiling, but you can also choose to ignore it). Even a straightforward matrix multiplication will solve the problem within the time limits, but you can get more ideas from these sources:

- https://en.wikipedia.org/wiki/Loop_nest_optimization
- https://en.wikipedia.org/wiki/Matrix_chain_multiplication
- https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm

2.6.2 Output

Write the resulting matrix **M** to STDOUT. The format should be the float numbers with a newline after each row and a blank-space between the numbers. We recommend to use `setprecision(6)` to get numbers that resemble those shown below, but do *not* expect to get identical values!. Your solution will be compared with the CMB solution and relative differences[4] smaller than 0.01 will be accepted.

[4] Calculated as abs((your result - expected result)/expected result).

2.6.3 Examples

Input:

```
3 4
3 2
3 3
2 3
```

Output:

```
-0.019398 -0.188008
-0.034079 -0.024902
```

Note: Here **N** is 3 and **S** is 4

Input:

```
4 777
16 8
640 16
32 640
8 32
```

Output:

```
4.51555 -4.03261 1.5672 -1.76524 -3.15909 0.64387 -4.01473 0.670606
10.6847 4.5529 -2.94728 5.37495 -1.44017 -3.43169 -2.55809 1.90042
-2.11201 -3.09145 -6.03958 -0.780988 -1.25857 3.3519 -5.09616 1.79759
-4.96459 -4.97667 -5.10007 -4.6546 -0.892383 -1.78725 -0.411896 -2.87651
0.936217 -0.45678 4.80988 5.40307 -5.30995 -0.814619 5.73566 7.75997
-6.74114 0.54747 7.60792 1.3539 8.28187 6.4683 6.79328 6.06388
1.19645 4.38832 -4.29697 3.85732 5.90794 1.53554 -0.290198 2.42656
-4.11833 -5.05589 -2.18385 -5.31018 -0.313578 1.49114 2.29119 -6.04179
```

Note: Here **N** is 4 and **S** is 777.

2.7 Campus guards

Level of difficulty: ★★★

In order to prevent that this can happens again, the rector has found out that there should be placed several guards at the campus to watch over the area. The guards will be students at NTNU that have volunteered to be a guard many times weekly. Since the students have different time schedules due to diverse subjects and activities, they have different wishes on which time slots they want to be a guard. Therefore the rector has asked you to write a program that takes as input a table of different student wishes, and returns an allocation of students to time slots that fulfil certain criteria.

There is a constraint that each student should guard *at most* **L** times per week and it should be exactly **C** guards allocated to every time slot. Students wishes are numbers in the range 1..5. The number of time slots **N** can be in the range [1, 6000]. Different ways of distributing the students to the time slots according to the criteria will give different *sum of priorities*, here called **sumP**. As an example — if there are three students and they get time slots with priorities (5 and 4), (4 and 4) and (5 and 2) the sumP value will be 24. Your task is to find a distribution of students where many students get their most wanted time slots, i.e. to maximize **sumP**, or in other words — maximize student happiness!.

2.7.1 Task (Slightly revised)

The input consists of 4 integers; **N**, **L**, **C** and **S**, where S is the seed for the random number generator. The seed is used to generate a matrix of student wishes, row by row. Use the random number generator from the CMB HowTo page and generate integers in the range 1..5 row by row. The number of columns in the matrix equals N, and the number of rows (number of student guards) **should be calculated with the formula** $rows = \lceil (N * C)/L \rceil$. Let all students get exactly L time slots, but for some combinations of input parameters there will be a last student that get a lower load (what remains). This might be well deserved, since it is the *student guard superintendent!*

A solution is a list of student allocations to time slots. Time slots are numbered 0, 1, ... (N-1). To be a valid solution — it should be exactly C students allocated to every time slot, and every student should be scheduled to *at most* L time slots. The sum of the priorities achieved by all student guards is called **sumP**, and should be printed after the list of allocations.

2.7.2 Scoring

For this problem CMB will in addition to time and energy report a *score value*. This is calculated by a rather magic but extremely fair formula based on the **sumP** value such that a high value of **sumP** gives a lower (better) score. In addition to the standard scoring of the three fastest (and valid), and the three most energy efficient (and valid) solutions (See Section 1.4), it will be given 6, 4 and 2 extra points for the three best (lowest) results for the score value. Hence, a participator will probably submit several different solutions, since optimizing for fast execution can lead to different strategies than optimizing for a good score.

2.7.3 Examples

Note that we have many possible correct solutions. For each of the input examples below, only one of many possible valid outputs is shown.

Input example A:

```
3 2 2 1
```

Output:

```
0 2 -1
1 2 -1
0 1 -1
Sum priorities: 18
```

Explanation:
For this input your program should generate the following matrix of wishes, but it should not be printed as part of the solution.

```
3 2 2
1 4 5
3 1 3
```

Input example B:

```
6 2 2 1
```

Output:

```
1 2 -1
3 4 -1
0 5 -1
1 2 -1
3 4 -1
0 5 -1
Sum priorities: 37
```

Explanation:
For this exact case the program should generate the following matrix of wishes, but it should not be printed as part of the solution.

```
3 2 2 1 4 5
3 1 3 1 5 1
1 2 1 2 1 4
1 2 3 5 4 2
4 1 4 4 3 3
5 4 1 1 4 5
```

Thank you for taking part in CMB Challenge 2023!