INFO0027-1: Project 1

Pr Laurent MATHY, Sami BEN MARIEM

February 26, 2021

1 Context

Your Professor needs help!

As part of a common work on the creation of a software Internet router, a PhD student and himself designed and studied a GPU-based forwarding algorithm. The general idea is that Internet packets are processed by the CPU, but the forwarding computation, that is the computation of which port each packet should be sent out, is done on a GPU.

The internal structure of a GPU is irrelevant to our problem, but suffice it to say that GPUs are extremely good at *simultaneously* applying the same computation to many pieces of data. In our case, the input (the pieces of data) is the destination address of Internet packets.

Thus, to efficiently use the GPU, the CPU prepares batches of packets (each of these packets has a potentially different destination address) and sends the destination addresses, corresponding to the packets in the batch, as an array to the GPU. While the GPU is computing the corresponding ports to which the packets must be sent, the CPU is physically sending the packets from the previous batch, as well as already preparing the next batch. There are therefore 3 batches of packets being processed at anyone time in the machine: a batch of packets being sent out by the CPU, a batch of packets whose forwarding decisions are being computed by the GPU, and a batch of packets being built by the CPU.

When its job is done, the GPU sends back an array to the CPU, containing the port number that the input destination address at the same index must be sent to (the output array thus have exactly the same size as the input array). For instance, if the input to the GPU is [d_0, d_1, d_0, d_2, d_2, d_0, d_3], then the output might be [p_2, p_0, p_2, p_2, p_2, p_2, p_1].

There are a few observations that can be made:

- in a batch, a same destination address will always be forwarded to the same port;
- a same destination address may appear several times within a batch.

Given that the GPU input and output may have to travel on the PCIe bus (depending on the specific architecture of the machine), we thought it may be advantageous to only send unique destination addresses to the GPU.

To do so, the batch of packets would still be represented as an array (with duplicate destination addresses), but an auxiliary array of unique destinations would be built: whenever a new destination address (in this batch) is encountered, it is added to the next available position in the auxiliary array, while the index of a destination address in the auxiliary is recorded alongside the packet in the batch array.

For instance, the above batch would now be represented by two arrays:

1. the batch array: [0, 1, 0, 2, 2, 0, 3]

2. the auxiliary array: $[d_0, d_1, d_2, d_3]$

The auxiliary array is then sent to the GPU, whose output would then be $[p_2, p_0, p_2, p_1]$. By then using the index recorded in the batch array, the CPU can readily find which port to send each packet to, by direct lookup in the output array.

In this assignment, your task is to design and implement a data structure that will represent this auxiliary array as well as some functions to use it.

2 Interface

The implementation of your **A**bstract **D**ata **T**ype should be named `MAGIC` and should at least support the following 3 functionnalities:

```
MAGIC MAGICinit(int maxSize, int addrSize);
int MAGICindex(MAGIC m, char* dest);
void MAGICreset(MAGIC m);
```

- the `MAGICinit` function initializes the data structure, with a view to hold up-to `maxSize` elements and where `MAGIC` is a pointer type identifying the created instance of the `MAGIC` ADT. The size of the destination keys that are held by the `MAGIC` data structure is given by `addrSize`. Note that all the keys will be `addrSize`-byte long.
- the `MAGICindex` function returns the index of the destination (`dest`) in the auxiliary array. This function must therefore automatically assign the next available index to a new destination address.
- the `MAGICreset` start the magic afresh (i.e. "forget" all the addresses seen in the previous batch) at the beginning of a new batch.

The performance of both the `MAGICindex` and the `MAGICreset` operations is critical. Indeed, your data structure will operate in a context such that the `MAGICindex` function is called for every packet in a batch, while the `MAGICreset` function is called at the beginning of every new batch (when packets keep arriving into the router). On the other hand, the performance of the `MAGICinit` function is not so important, as this function is only called at creation time.

It is worth mentionning that the size parameter passed to `MAGICinit` is a strict maximum, and you can assume that there will never be more packets than this maximum in any batch.

3 Implementation & Performance study

You are asked to provide **two implementations**, of the above ADT, in **C**. Each of those should be based on a different data structure.

For each implementation, function prototypes, as well as the `MAGIC` ADT, must be contained in a header file called `magic.h`. This file may also contain other lines of code, *but only the strict minimum required* to support the use of this interface.

All the code required for the implementation of this interface must be provided in a separate file called `magic.c`.

Then, your two files should be added to a directory called `MAGIC1` for your first implementation and `MAGIC2` for your second.

You must then realize a **performance study**, comparing your two solutions to the problem. What to measure and how is left to your discretion.

4 Remarks

4.1 Respect the interface

Submission must scrupulously **follow the interface** defined above. You can of course define auxiliary functions as you see fit, but these should be properly hidden from the users of the interface.

4.2 Readability

Your code must be readable:

- Make the organisation of your code as obvious as possible. Remember you can create as many auxiliary functions as you see fit.
- Complement your self-documenting code with comments, where appropriate.
- Choose a coding convention, and stick to it. *Consistency* is key.
- Don't forget to use indentation.

4.3 Warnings

Your code must compile **without error or warning** with `gcc -Wall -pedantic -std=c99` on `ms8??` machines.

However, we advise you to check your code also with `gcc -Wall -Wextra -pedantic -std=c99`.

4.4 Evaluation

This project counts towards 20% of your final mark. In this project, you will be evaluated on: your algorithms and their efficiency, the organization of your code, respect of the defined API, correctness, your performance study and its analysis. Failure to comply with any of the points mentioned in here can (and most often will) result in a mark of zero!

5 Submission

Projects must be done in groups of 2 students.

Projects must be submitted through the submission platform before **Monday April 19$^{\text{th}}$**, 23:59 CET. After this time, a penalty will be applied to late submissions. This penalty is calculated as a deduction of 2^{N-1} marks (where N is the number of started days after the deadline).

You will submit a `<ID>.tar.xz` archive of a `<ID>` folder, where `<ID>` is your group ID given by the submission platform. This folder should contain the folder `MAGIC1` and `MAGIC2` as well as your report (max. 3 pages) explaining your algorithms and presenting your performance study.

Your code must compile with `gcc`, without error or warning. Failure to compile will result in an awarded mark of 0. Likewise, poor performance when run over large input will result in an awarded mark of 0.

The submission platform will do basic checks on your submission, and you can submit multiple times, so check your submission early!

Bon travail...