Università della Svizzera italiana

Institute of Computational Science ICS

**High-Performance Computing Lab** **2020**

Student: Gabriele Berra Discussed with: -

## Solution for Project 1

Due date: 13.10.2020 (midnight)

**HPC Lab 2020 — Submission Instructions**
**(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)**

- Assignments must be submitted to Icorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Matlab). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  *Project_number_lastname_firstname*
  and the file must be called:
  *project_number_lastname_firstname.zip*
  *project_number_lastname_firstname.pdf*
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

In this first assignment I will tackle - fist theoretically and then practically - one of the most important aspect to take in consideration when coding in order to reach the highest performance out of our machines, the memory hierarchies.

# 1. Explaining Memory Hierarchies *(30 Points)*

## 1.1. Introduction

In order to understand the memory hierarchies we first have to consider which types of memory exists and the relationship between each one. Basically, we can distinguish memories for their speed in accessing the data (i.e. the time - usually in nanoseconds - between the request of the data and when the data is received) and for their capacity of storing a large amount of them. Starting from the farthest from the Arithmetic Unit (AU) - which is the unit that receives data from the register and performs operations on them - we find the main memory, the biggest and also the slowest, capable of storing large amount of data. The second memories are the cashes which are smaller than the main memory but are fastest since they are closer to the AU. Starting from the biggest - and also the closest (physically) to the main memory - we find: Level 3 cache (L3), Level 2 cache (L2), Level 1 cache (L1). Last but not least, a fundamental memory very close to the AU - and, for this reason, the fastest - is represented by the registers.

Once we have understood the differences between the various types of memories, both on the basis of their size and their speed in accessing the data, we will focus on the relationship between

them. As said before, the AU is the unit that sends the request for a specific data. The latter is searched, firstly, in the registers and, if it is not present there, the AU searches for it in the L1 cache, then in L2 and so on. Finally, the last memory in which the AU has to search is the main memory. Now, it is easy to understand that, in order to reach high performance, it is necessary to load the data that we are going to use in the closest memory, so that the AU spends less time searching for it, thus improving the performance. Once understood what the different memories are and how they work, we can focus our attention on how to use them efficiently considering their different capacity. The perfect - and also utopistic - situation is when all the data that we are going to use in our computation are all in the L1 cache. Unfortunately for us, we can load only few data in L1, but if we can manage to use as much as possible the data loaded in it, then the AU does not have to search in other memories and, thus, the computational speed increases. Every time that the data that we search is present in the caches is called a *cache hit*. In contrast, if the data that we need is not in the caches, that is called a *cache miss* and, then, the AU has to load it directly from main memory, compromising the performance.

At the end of this introduction we can state that, if we want to reach the higher performance possible, we have to minimize the *cache misses* during the process, by using as much as possible the data already loaded in the caches. We could also consider the time of the AU in performing the computations but, since that time cannot be modified, we focus our attention only on minimizing the *cache misses* or, conversely, on maximizing the *cache hits*.

## 1.2. Identify the parameters of the memory hierarchy on the compute node of the ICS cluster

The ICS cluster is a collection of computers called *nodes* with different characteristics. In the ICS cluster there are 32 nodes CPU only, 8 with only one GPU, and 2 multi GPU.

With the aim of obtain the specifics of a node we have to:

- Run the command *likwid-topology* that gives as output the sizes for each cache and,
- run the command *cat /proc/meminfo* that gives more information regarding the main memory, the memory available, and other useful parameters.

In my case I took in consideration two nodes to better understand the differences. In the specific, those nodes are the *login node* and the *node07*. I considerate the latter for the only reason that, as we can see with the command *sinfo*, has a big memory, for that, I think it is appropriate to include it in my report regarding the memory hierarchies.

To be more precise, and also to answer the first question, I reported below two tables contain the sizes of the caches and main memory of, on the left hand side, the *login node* and, on the right hand side, the *node07*.

| ICS login node | |
|---|---|
| Main memory | 62.5443 GB |
| L3 cache | 25 MB |
| L2 cache | 256 kB |
| L1 cache | 32 kB |

| ICS node07 | |
|---|---|
| Main memory | 503.5439 GB |
| L3 cache | 25 MB |
| L2 cache | 256 kB |
| L1 cache | 32 kB |

Since the data gathered from the cluster regarding the main memory are in kB, for the computation of the size of it in GB I used the formula 1 GB = 1024 MB and 1 MB = 1024 kB.

At a first glimpse at the tables the two nodes differs only in the size of the main memory and the amount of data that can be stored in the caches are the same for each node taken in consideration. In conclusion, if we have to deal with an enormous amount of date we are going to use the ICS node07 instead of any other node.

## 1.3.

The two figures below were given by running the $run_membench.shfile, thefirstoneonmylaptopand, thesecondon$

## 2. Optimize Square Matrix-Matrix Multiplication *(70 Points)*