Università
della
Svizzera
italiana

**Institute of
Computing
CI**

High-Performance Computing Lab **2022**

Student: Claudio Maggioni Discussed with: —

Solution for Project 1

Due date: 12.10.2022 (midnight)

HPC 2022 — 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 project you will practice memory access optimization, performance-oriented programming, and OpenMP parallelizaton on the ICS Cluster .

1. Explaining Memory Hierarchies *(25 Points)*

1.1. Memory Hierarchy Parameters of the Cluster

By identifying the memory hierarchy parameters through `likwid-topology` for the cache topology and `free -g` for the amount of primary memory I find the following values:

Main memory	62 GB
L3 cache	25 MB per socket
L2 cache	256 kB per core
L1 cache	32 kB per core

All values are reported using base 2 IEC byte units. The cluster has 2 sockets and a total of 20 cores (10 per socket). The cache topology diagram reported by `likwid-topology -g` is the following:

```
Socket 0:
+-------------------------------------------------------------------------------------------------------------------------+
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |    0    | |    1    | |    2    | |    3    | |    4    | |    5    | |    6    | |    7    | |    8    | |    9    | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------------------------------------------------------------------------------------------------------------------+ |
| |                                                        25 MB                                                        | |
| +---------------------------------------------------------------------------------------------------------------------+ |
+-------------------------------------------------------------------------------------------------------------------------+
Socket 1:
+-------------------------------------------------------------------------------------------------------------------------+
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |   10    | |   11    | |   12    | |   13    | |   14    | |   15    | |   16    | |   17    | |   18    | |   19    | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | |
| +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ +---------+ |
| +---------------------------------------------------------------------------------------------------------------------+ |
| |                                                        25 MB                                                        | |
| +---------------------------------------------------------------------------------------------------------------------+ |
+-------------------------------------------------------------------------------------------------------------------------+
```

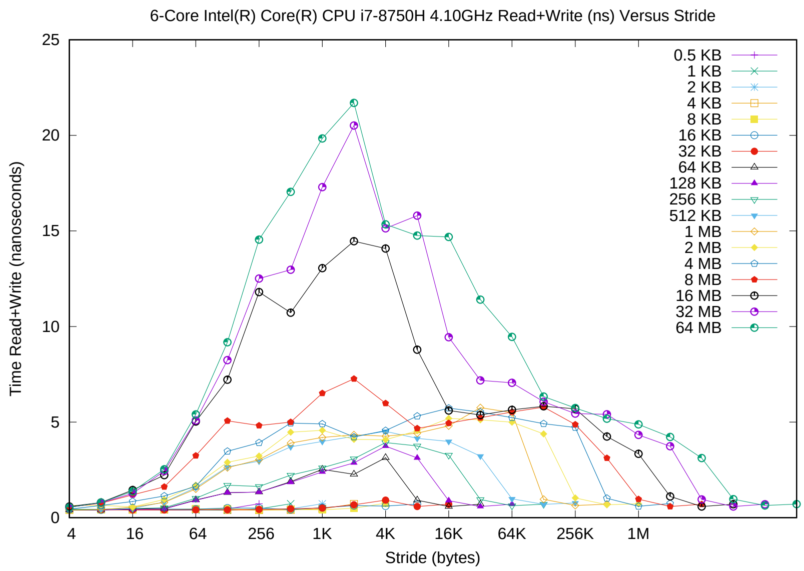
1.2. Memory Access Pattern of `membench.c`

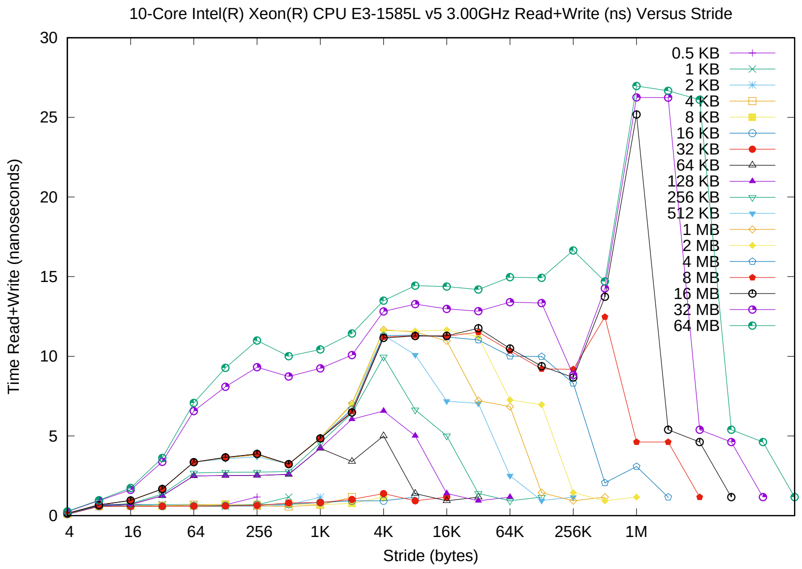
The benchmark `membench.c` measures the average time of repeated read and write overations across a set of indices of a stack-allocated array of 32-bit signed integers. The indices vary according to the access pattern used, which in turn is defined by two variables, `csize` and `stride`. `csize` is an upper bound on the index value, i.e. (one more of) the highest index used to access the array in the pattern. `stride` determines the difference between array indexes over access iterations, i.e. a `stride` of 1 will access every array index, a `stride` of 2 will skip every other index, a `stride` of 4 will access one index then skip 3 and so on and so forth.

Therefore, for `csize = 128` and `stride = 1` the array will access all indexes between 0 and 127 sequentially, and for `csize =` 2^{20} and `stride =` 2^{10} the benchmark will access index 0, then index $2^{10} - 1$, and finally index $2^{20} - 1$i.

1.3. Analyzing Benchmark Results

The `membench.c` benchmark results for my personal laptop (Macbook Pro 2018 with a Core i7-8750H CPU) and the cluster are shown below respectively:





The memory access graph for the cluster's benchmark results shows that temporal locality is best for small array sizes and for small `stride` values. In particular, for array memory sizes of 16MB or lower (`csize` of $4 \cdot 2^{20}$ or lower) and `stride` values of 2048 or lower the mean read+write time is less than 10 nanoseconds. Temporal locality is worst for large sizes and strides, although the largest values of `stride` for each size (like `csize / 2` and `csize / 4`) achieve better mean times due to the few elements accessed in the pattern (this observation is also valid for the largest strides of each size series shown in the graph).

2. Optimize Square Matrix-Matrix Multiplication *(60 Points)*

3. Quality of the Report *(15 Points)*