Università della Svizzera italiana

Institute of Computing
CI

High-Performance Computing Lab **Institute of Computing**

Student: Metehan Dündar Discussed with: Yusuf Eren Tunç

Solution for Project 1

HPC Lab — 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. Identifying Parameters of the Memory Hierarchy

Table 1: Memory hierarchy on the compute node of Intel(R)Apple M1 CPU

Main memory	8 GB
L3 cache	8 MB
L2 cache	12 MB (per 4 cores)
L1I cache	192 KB
L1D cache	128 KB

1.2. Running Membench Program

In this exercise, we were expected to execute the membench software on our own workstation. Because I have a laptop with an arm64-based CPU. I have to modify it to x8664 architecture. Figure 1 was created on my own system using an Apple M1 CPU running macOS Sonoma.

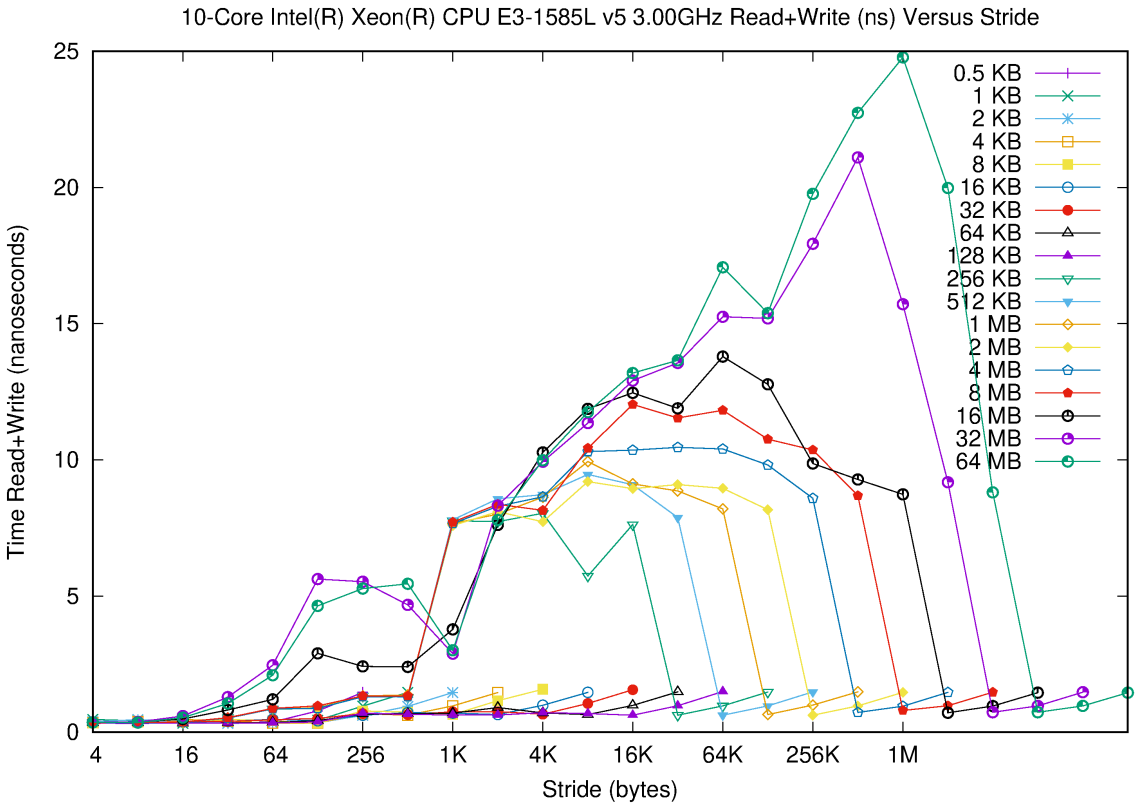


Figure 1: Memory hierarchy on Apple M1

As shown in Figure 1, the curves can be classified into four groups according to their working set size C:

1. $C \leq 128\text{KB}$
   - Because the working set fits fully in the L1D cache, each element is loaded precisely once, resulting in a short latency per read/write.
2. $128\text{KB} < C \leq 3\text{MB}$
   - The working set exceeds the L1D cache limit but fits in the L2 cache.
   - When $stride < 8\text{KB}$, the data inside the same 8KB page may be prefetched by hardware, which can disguise the latency of loading from L2 cache to L1D cache, and therefore the curves steadily grow owing to the prefetch technique. Peaks arise at $stride = 8\text{KB}$ where data prefetching is no longer activated.
   - When $8\text{KB} \leq stride < C/32$, all accessible entries are from the same set of L1D cache, and more than 8 cache lines must be loaded, resulting in conflict misses. Before the next access, each cache line is thrashed from an 8-way associative L1D.
   - When $stride \geq C/32$ is used, no more than 8 elements are accessed, and they all belong to the same L1D set. Each cache line must be loaded just once from L2 to L1D. As a result, the time per read/write decreases dramatically.

3. 3MB $< C \leq$ 8MB
    - The working set exceeds the L1D and L2 cache capacities but fits in the L3 cache. Because L3 cache has lower bandwidth and greater latency, and there may be TLB misses, time per read/write is longer than in the second group.
    - Peaks emerge around $stride = 16$KB, when prefetchers stop working.
    - L2 conflict misses appear at 512KB $\leq stride < C/8$. However, before the removal of L2 conflict miss, the accessible pieces become totally fitting in one set of L1D. Because the L2 cache is non-inclusive, such behaviour is permissible, resulting in a sharp decline at $stride = C/16$. As a result, the curve form after that turning point is identical to that of the second group.
    - Because there may be additional data in L3, $C = 8$MB cannot use the whole L3 cache, and this curve reveals a pattern in between the third and fourth groups.
4. $C >$ 8MB
    - The working set exceeds cache capacity. Because memory has lesser bandwidth and greater latency, the time per read/write is longer in the first group, and there may be TLB misses as well.
    - When stride is small, the time per read/write is substantially higher than in other curves, which are dominated by memory bandwidth and latency.
    - A hash function distributes the data among L3 cache slices, and the pattern of memory access cannot be described by the standard N-way cache paradigm. The plot shows that beyond some point, the curves have comparable characteristics to the curves of the third group, which are primarily influenced by the properties of the L1D and L2 cache.

1.3. Characterizing the Memory Access Patterns

- $csize = 128$ and $stride = 1$
  Because one integer equals four bytes, we get $128 * 4 = 512$. As a result, we must examine the purple line with the straight line equivalent to 0.5 kB. Because the array fits fully in L1D cache, each cache line is loaded precisely once. The L1D cache is accessed sequentially using the stride-1 reference pattern.

- $csize = 2^{20}$ and $stride = csize/2$
  This time we get $csize = 2^{22}$ Byte (4 MB) and $stride = 2^{21}$ Byte (2 MB). As a result, we must examine the light blue line with an empty circle. In an alternate pattern, only `x[0]` and `x[csize/2-1]` are accessible. Because the L1D cache is not directly mapped, the two cache lines containing these data are loaded precisely once into the L1D cache.

1.4. Good Temporal Locality

Because they fit into L1D, arrays of size $csize = C \leq 128$K exhibit acceptable temporal locality for all strides, as discussed in Section 1.2. For array of size $csize = C > 128$K, it shows good temporal locality when $stride \geq csize/8$ since no more than 8 elements are accessed and these elements fit into the same set of L1D.

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

2.1. Optimizing the matrix multiplication

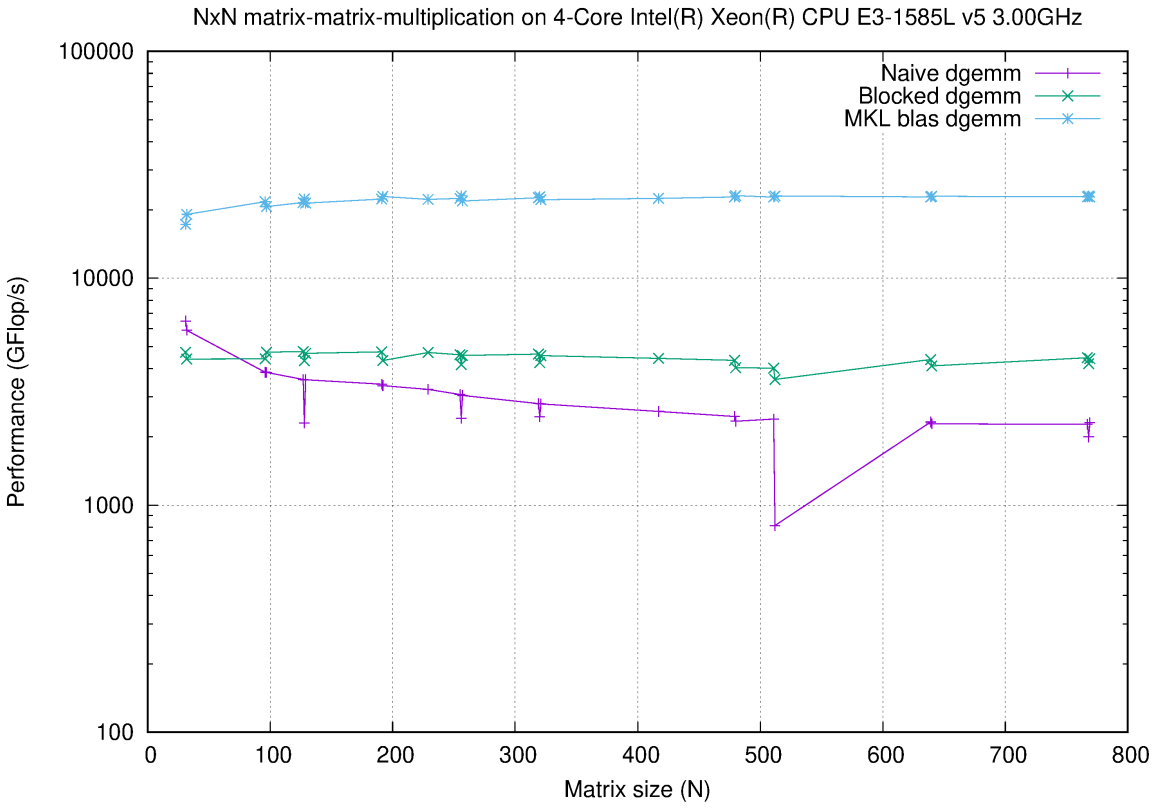
Cache blocking is a mechanism that can improve cache performance by accelerating its activities. This is accomplished by minimising the quantity of data retrieval from main memory by bringing data blocks into cache and storing them in the cache for speedier retrieval.

This blocking procedure will then increase temporal locality by ensuring that data remains in cache to be accessed several times. If the matrices are stored in rows in main memory, there would be many more cache misses, making data retrieval and storage process slow.

The Naive approach iterates over the rows from i = 0 to N, then through the columns for each row from j = 0 to n. k iterates across the rows of the first matrix A and the columns of the second matrix B for each i, j element. This is how k carries out this procedure.
C[i,j]+= A[i,k] * B[k,j].

The blocking implementation differs in that the matrix must first be partitioned into blocks before the matrices A and B are multiplied to form matrix C. To do this, the A and B matrices must first be separated into blocks. Once matrices has been divided into blocks, the code reads each block of B, each block of A, and read and write each block of C.

The procedure iterates to ensure that each block is handled. The figure below depicts the Naive implementation, which performs poorly. Implementing blocking can increase its performance. When the blocked matrix multiplication is completed, the cache's speed improves since the data may be reused. This means that the computer can accomplish many more operations per second with this approach. The greater the number of Gigaflops per second, the better.



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