Università della Svizzera italiana

Institute of Computing CI

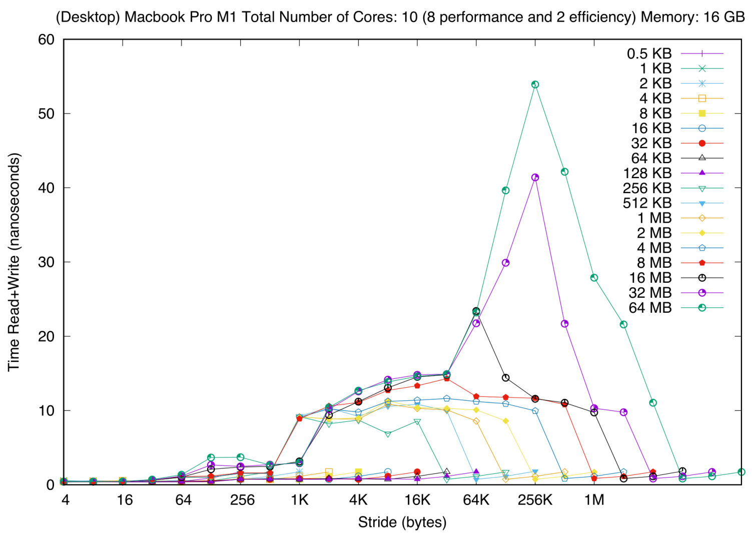
High-Performance Computing Lab **Institute of Computing**

Student: Kaan Egemen Sen Discussed with: Lorenzo VARESE, Vijaya AKAVARAPU

Solution for Project 1

1. Explaining Memory Hierarchies *(25 Points)*

Since the clusters are currently inaccessible due to maintenance, this part of the assignment is omitted in accordance with the instructions from the professor. We will discuss local performance only.



As we can see in the figure above our L1 cache works at most 128Kb which is purple triangles. So this gives us information about the size of the L1 cache entirely. We specifically checked the performances and there is no peak at all. Also when we check the L2 cache and main memory we get the information below :

Table 1: Parameters of the memory hierarchy

Main memory	**16 GB**
L2 cache	**24 MB**
L1 cache	**128 kB**

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

2.1. Implementation

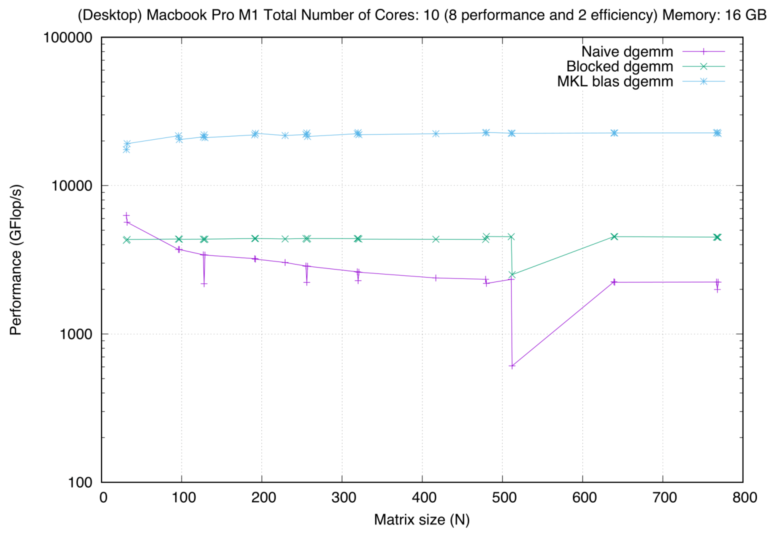
We did our calculations using chunked blocks so that the operations are optimized. The following starting and finishing points are used in the loops:

```
void square_dgemm (int n, double* A, double* B, double* C) {
 const int block_size=36;

  /* For each row i of A */
  for (int i_ = 0; i_ < n; i_ += block_size) {
    for (int j_ = 0; j_ < n; j_ += block_size) {
      for (int k_ = 0; k_ < n; k_ += block_size) {
        for (int i = i_; i < minimum(n,(i_+block_size)); i++) {
          for (int j = j_; j < minimum(n,(j_+block_size)); j++) {
            double cij = C[(i)+(j)*n];
            for (int k = k_; k < minimum(n,(k_+block_size)); k++) {
              cij += A[(i)+(k)*n] * B[(k)+(j)*n];
            }
            C[(i)+(j)*n] = cij;
          }
        }
      }
    }
  }
}
```

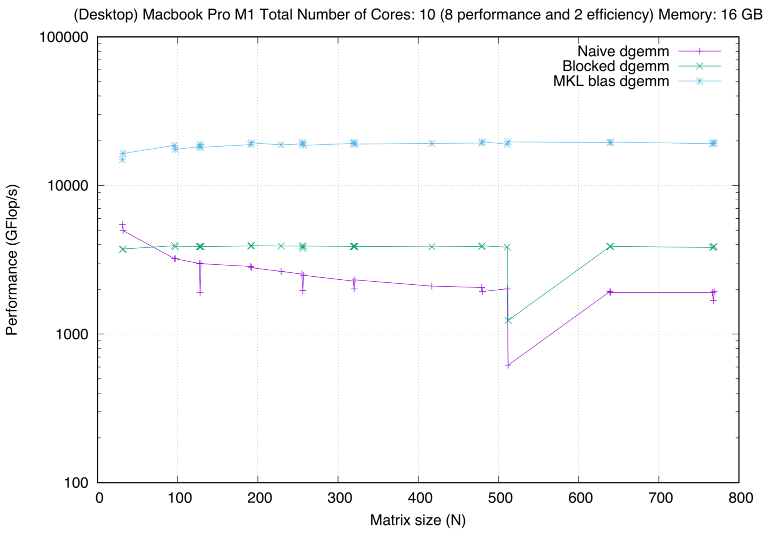
2.2. Runtime Result and Discussion

Through some internet search, we found out the 2021 Macbook Pro M1 has 128Kb. The theoretical ideal block size can be found by s = $\sqrt{M/3}$, where s is the block size and M is the size of the fast memory. The L1 cache in the local machine is 128Kb, so $s = 6$. But it is only theoretical that's why we picked our size as $s = 18$. After that size block, our blocked calculation drops down under the naive version which is something we do not want at all. As we can see in the figure below :



The **BLAS dgemm** is the best-performing method, since **BLAS** improves the performance by running in a multi-threaded way. The **Naive dgemm** has the worst performance because it does not use spatial locality. This causes a larger amount of slow memory access since the data stored in the cache is not used efficiently.

When we change the box size to 24 we can clearly see the difference in the performance. Especially with a matrix of size $N = 500$, the **blocked dgemm** performs significantly worse than it did with the block size 18.



3. References

1) Apple-M1-Pro-Processor-Benchmarks

2) Apple M1 Wiki