TP1 - Optimizing Memory Access

Imad Kissami

EL AMRANI Soufiane

19/02/2025

Exercice 1 :

Stride	Sum	Time (msec)	Rate (MB/s)
1	1000000.000000	1.000000	7629.394531
2	1000000.000000	2.000000	3814.697266
3	1000000.000000	1.000000	7629.394531
4	1000000.000000	3.000000	2543.131510
5	1000000.000000	2.000000	3814.697266
6	1000000.000000	4.000000	1907.348633
7	1000000.000000	4.000000	1907.348633
8	1000000.000000	5.000000	1525.878906
9	1000000.000000	5.000000	1525.878906
10	1000000.000000	5.000000	1525.878906
11	1000000.000000	5.000000	1525.878906
12	1000000.000000	5.000000	1525.878906
13	1000000.000000	5.000000	1525.878906
14	1000000.000000	5.000000	1525.878906
15	1000000.000000	6.000000	1271.565755
16	1000000.000000	8.000000	953.674316
17	1000000.000000	5.000000	1525.878906
18	1000000.000000	6.000000	1271.565755
19	1000000.000000	6.000000	1271.565755
20	1000000.000000	6.000000	1271.565755

Stride	Sum	Time (msec)	Rate (MB/s)
1	1000000.000000	1.000000	7629.394531
2	1000000.000000	1.000000	7629.394531
3	1000000.000000	2.000000	3814.697266
4	1000000.000000	1.000000	7629.394531
5	1000000.000000	2.000000	3814.697266
6	1000000.000000	2.000000	3814.697266
7	1000000.000000	3.000000	2543.131510
8	1000000.000000	3.000000	2543.131510
9	1000000.000000	3.000000	2543.131510
10	1000000.000000	4.000000	1907.348633
11	1000000.000000	4.000000	1907.348633
12	1000000.000000	4.000000	1907.348633
13	1000000.000000	5.000000	1525.878906
14	1000000.000000	5.000000	1525.878906
15	1000000.000000	5.000000	1525.878906
16	1000000.000000	6.000000	1271.565755
17	1000000.000000	6.000000	1271.565755
18	1000000.000000	5.000000	1525.878906
19	1000000.000000	5.000000	1525.878906
20	1000000.000000	6.000000	1271.565755

My observations :

- Execution Time :-O2 consistently reduces execution time compared to -O0, especially for larger strides.The improvement is more pronounced for strides where cache locality is poor (e.g.,strides 8,16).
- Memory Bandwidth :-O2 improves memory bandwidth across all strides, demonstrating better utilization of CPU resources.However,the improvement diminishes for very large strides (e.g., stride16), where cache misses dominate performance.

Exercice 2 :

```
PS C:\Users\soufiane\OneDrive\Bureau\TP1_PL> ./mxm
Standard Matrix Multiplication:
Execution Time: 0.001000 seconds
Memory Bandwidth: 16000.00 MB/s
```

```
PS C:\Users\soufiane\OneDrive\Bureau\TP1_PL> ./mxm

Optimized Matrix Multiplication:
Execution Time: 0.001000 seconds
Memory Bandwidth: 16000.00 MB/s
```

- **Execution Time :Both versions of the matrix multiplication algorithm achieved the same execution time of 0.001000 seconds .This suggests that,for the given matrix size and hardware configuration,the reordering of loops did not result in a measurable improvement in execution time.**
- **Memory Bandwidth :Both versions achieved the same memory bandwidth of 16000.00 MB/s This indicates that the memory access patterns in both versions were equally efficient in terms of data transfer rates.**

Possible Reasons for Similar Results :

- **The matrix size used in this experiment might have been small enough to fit entirely in the CPU cache.**
- **My compiler may have applied additional optimizations that mitigated the differences between the two versions.**

Exercice 3 :

```
Block Size, CPU Time (ms), Memory Bandwidth (MB/s)
8, 919.000000, 18.255948
16, 895.000000, 18.745493
32, 913.000000, 18.375921
64, 995.000000, 16.861524
128, 1154.000000, 14.538315
256, 1185.000000, 14.157988
```

- **Optimal Block Size :**From the results:The lowest CPU time is achieved with a block size of 16 , which has a CPU time of 895 ms .The highest memory bandwidth is also achieved with a block size of 16 , at 18.75 MB/s Thus, the optimal block size based on these results is 16 .
- **Reasoning for Optimal Block Size :**A block size of 16 fits well within the CPU's cache hierarchy, allowing efficient reuse of data within blocks. This minimizes cache misses and improves performance.Smaller block sizes (e.g., 8) introduce higher loop overhead due to frequent iterations over smaller chunks of data.

Exercice 4 :

My Analysis :

- **Memory Leak Cause :**The original program did not free the duplicate array (array_copy), leading to a memory leak.The free_memory function was incomplete and did not actually free the memory.
- **Fix :** I updated the free_memory function to properly free memory.and I ensured that both array and array_copy were freed before the program exited.