**COMPSYS 304 Assignment 3**

**Task 1 Cache measurement**

**Name of the processor**: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz

**LEVEL 1 Cache size**:

Instruction cache size: 32768 – 32 KB

Data cache size: 32768 – 32 KB

**LEVEL 2 Cache size**: 262144 – 256 KB

**LEVEL 3 Cache size**: 6291456 – 6 MB

**Time measurement table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **size of a** | **time per iteration /ns** | **time per iteration /ns** |
|  |  | Case 1 | Case 2 |
| 2048 | 8 KB | 0.391 | 0.386 |
| 4096 | 16 KB | 0.369 | 0.378 |
| 8192 | 32 KB | 0.376 | 0.369 |
| 16384 | 64 KB | 0.375 | 0.463 |
| 32768 | 128 KB | 0.385 | 0.540 |
| 65536 | 256 KB | 0.409 | 0.711 |
| 131072 | 512 KB | 0.406 | 0.977 |
| 262144 | 1 MB | 0.404 | 1.171 |
| 524288 | 2 MB | 0.418 | 1.413 |
| 1048576 | 4 MB | 0.485 | 3.262 |
| 2097152 | 8 MB | 0.558 | 5.469 |
| 4194304 | 16 MB | 0.610 | 6.809 |
| 8388608 | 32 MB | 0.629 | 7.510 |
| 16777216 | 64 MB | 0.638 | 8.005 |

**Time measurement chart:**

**Performance analysis:**

For case 1, the access pattern is going through the array linearly which provides a good spatial locality. All the subsequent elements of the array will be loaded into the cache so the next memory access can use the data in the cache block directly. Therefore, the time per iteration has a slow uniform growth after the working set is larger than the cache size.

For case 2, the access pattern is going through the array randomly which will greatly increase the miss rate as the next memory access will most likely not be in the cache when the working set is larger than the cache size. Therefore, the time per iteration has a rapid and uneven growth after the working set is larger than the cache size.

The time per iteration of both case 1 and case 2 are similar when the array size is below 32 KB. Because all the data of the array can be loaded into L1 cache, so both case 1 and case 2 have similar miss rate. However, after the array size is larger than L1 cache size (32KB) but smaller than L3 cache size (6 MB), the time per iteration of case 2 has a relative uniform growth as the average memory access time of L2/3 is higher than L1. Finally, when the working set is greater than L3, the time taken is significantly increased because a large number of main memory accesses are required.

**Task 2 Matrix product**

**Matrix 1 Time taken: 2.03 secs**

In this implementation, the access pattern of array A is row-major which provides a good spatial locality as the subsequent elements will be loaded into the same cache line. However, the access pattern of array B is column major. The size of array is 1000 x 1000 which is 8000 bytes per row. Most of the sequential elements will not load into the same cache line. Therefore, the performance is reduced by not using all words of each cache line.

**Matrix 2 Time taken: 1.29 secs**

**Matrix 3 Time taken: 0.81 secs**

Based on the original approach, instead of doing multiplication on the N x N matrices, we use blocking algorithm that has a submatrix of size **k** x **k** (**k** = 16). This can be done by adding two outer for-loop with an increment of **k**. Therefore, the **k** x **k** submatrix of array **b** and a row of length **k** of array **c** can fit in the cache. Both array **b** and array **c** can reuse **k** times each time the data are brought in.

Text

Description automatically generatedText

Description automatically generated**Appendix:**