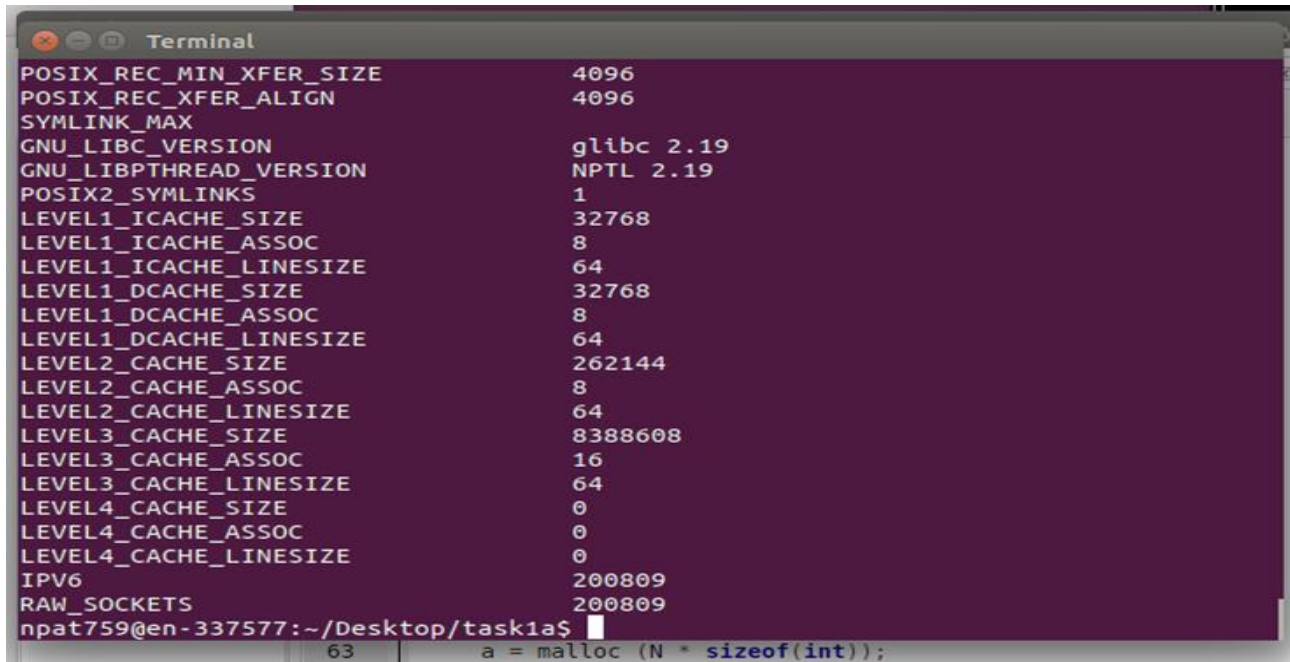


Question 1

Processor: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz



```

Terminal
POSIX_REC_MIN_XFER_SIZE      4096
POSIX_REC_XFER_ALIGN         4096
SYMLINK_MAX
GNU_LIBC_VERSION             glibc 2.19
GNU_LIBPTHREAD_VERSION      NPTL 2.19
POSIX2_SYMLINKS              1
LEVEL1_ICACHE_SIZE           32768
LEVEL1_ICACHE_ASSOC          8
LEVEL1_ICACHE_LINESIZE       64
LEVEL1_DCACHE_SIZE           32768
LEVEL1_DCACHE_ASSOC          8
LEVEL1_DCACHE_LINESIZE       64
LEVEL2_CACHE_SIZE            262144
LEVEL2_CACHE_ASSOC           8
LEVEL2_CACHE_LINESIZE        64
LEVEL3_CACHE_SIZE            8388608
LEVEL3_CACHE_ASSOC           16
LEVEL3_CACHE_LINESIZE        64
LEVEL4_CACHE_SIZE            0
LEVEL4_CACHE_ASSOC           0
LEVEL4_CACHE_LINESIZE        0
IPV6                          200809
RAW_SOCKETS                  200809
npa759@en-337577:~/Desktop/task1a$

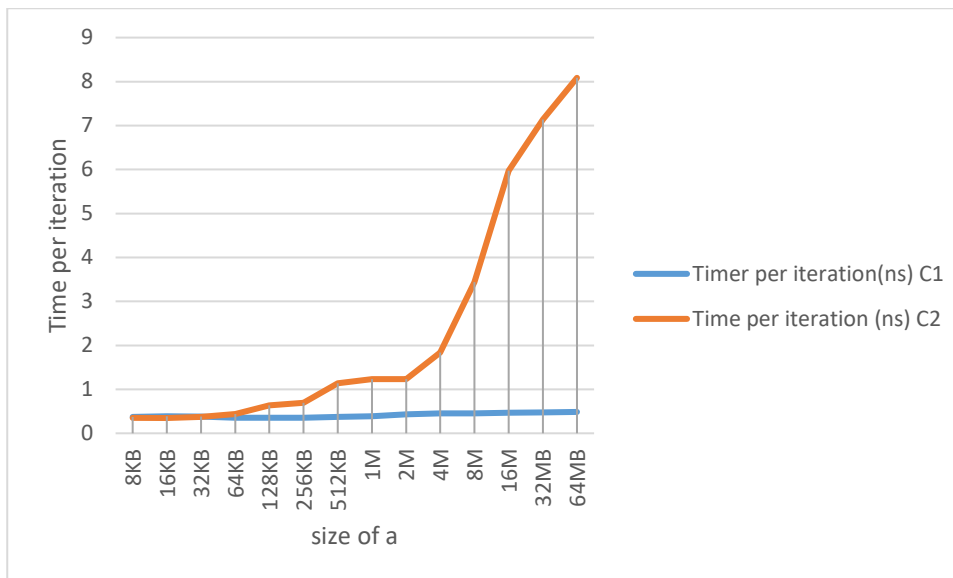
```

Level 1 Cache: 32 KB

Level 2 Cache: 32 KB

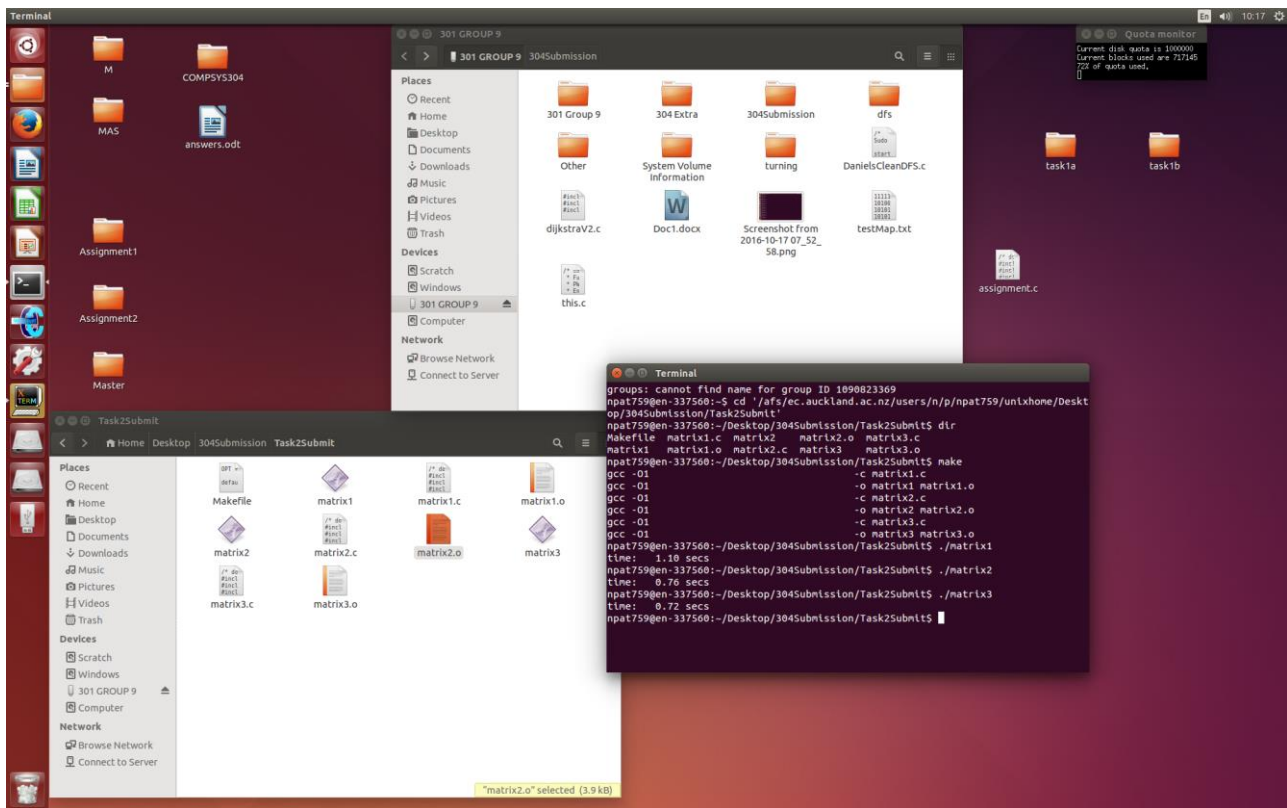
Level 3 Cache: 8 MB

M	100000		
N	Size of a	Timer per iteration(ns)	Time per iteration (ns)
		C1	C2
2048	8KB	0.375	0.349
4096	16KB	0.389	0.348
8192	32KB	0.383	0.376
16384	64KB	0.354	0.439
32768	128KB	0.354	0.635
65536	256KB	0.354	0.697
131072	512KB	0.372	1.135
262144	1M	0.385	1.235
524288	2M	0.433	1.232
1048576	4M	0.456	1.835
2097152	8M	0.456	3.435
4194304	16M	0.469	5.975
8388608	32MB	0.476	7.135
16777216	64MB	0.486	8.085



C1 remains about the same time because you are loading the cache line once and using it, then loading the next and using it. Where as in C2 you are having to load in different ones whenever you get a cache miss because they are random. So as N increases the number of cache misses in a random array is likely to contribute to the increase in average time per iteration.

Question 2



Time taken for execution matrix1: 1.10 seconds

Time taken for execution matrix2: 0.76 seconds

Time taken for execution matrix3: 0.72 seconds

2.1) Data for a 2d array is stored in memory as shown below.

Mem0	Mem1	Mem2	Mem3	Mem4
Mem5	Mem6	Mem7	Mem8	Mem9
Mem10	Mem11	Mem12	Mem13	Mem14
Mem15	Mem16	Mem17	Mem18	Mem19
Mem20	Mem21	Mem22	Mem23	Mem24

Cache Size (Note: This is just an example)

So when you do matrix multiplication you need to access different rows keeping the column the same ie. You would access Mem0 then Mem5 then Mem10 and so on...

This becomes inefficient when the matrix becomes large because every time you access a new row cache loads X number of consecutive memory. And if X is less than the number of columns, you will need to load new data into the cache every single time, because you will always get a cache miss.

2.2) So in the case when you transpose the matrix your memory is looks like as shown below. This

is better because every time you need to access a new element cache will load that memory location + X number of consecutive elements. This is good because now you can guarantee you will most likely get a cache hit, and you will NOT need to access the main memory as often as before.

Mem0	Mem5	Mem10	Mem15	Mem20
Mem1	Mem6	Mem11	Mem16	Mem21
Mem2	Mem7	Mem12	Mem17	Mem22
Mem3	Mem8	Mem13	Mem18	Mem23
Mem4	Mem9	Mem14	Mem19	Mem24



Cache Size (Note: This is just an example)

2.3) The general idea of blocking is to organize the matrix in the program into large chunks called blocks. The program is structured so that it loads a chunk into the L1 cache, does all the reads and writes that it needs to on that chunk, then discards the chunk, loads in the next chunk, and so on.

$$\begin{array}{|c|c|} \hline C11 & C12 \\ \hline C21 & C22 \\ \hline \end{array} = \begin{array}{|c|c|} \hline B11 & B12 \\ \hline B21 & B22 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline A11 & A12 \\ \hline A21 & A22 \\ \hline \end{array}$$

Suppose we want to compute $C = AB$, where A, B, and C are each 8×8 matrices. Then we can partition each matrix into four 4×4 submatrices.

The key idea is that it loads a block of B into the cache, uses it up, and then discards it. The most beneficial thing about blocking is that the time per iteration remains nearly constant with increasing array size.