Measuring Different Configuration Parameters of Memory Hierarchy by Running a Program

Akshat Khare

COL718

Setup

I had two machines at disposal:

1. Personal (lab assigned) computer (ubuntu)
2. Server (lab assigned) baadal virtual machine

For getting started with the software I experimented on Mac. I moved to Linux system (Ubuntu) afterwards.

Description of machines:

1. Personal computer (ubuntu):
    a. L1d cache: 32K
    b. L2 cache: 256K
    c. L3 cache: 8192K
    d. cacheline : 64B
    e. Page size: 4K
    f. Tlb size: 64 entries
2. Baadal:
    a. L1d cache: 32K
    b. L2 cache: 4M
    c. cacheline : 64B
    d. Page size: 4K
    e. Tlb size: 64 entries

Method

The program was changed to work on linux machine by taking the time in clock and then in time_t. Both gave similar graphs and cache parameters extracted came out to be same. I restarted the pc before every run. For server I used screen command to let the server run for long time without continous ssh access. It took almost 2 hours to complete each execution.

Assessments and Measures

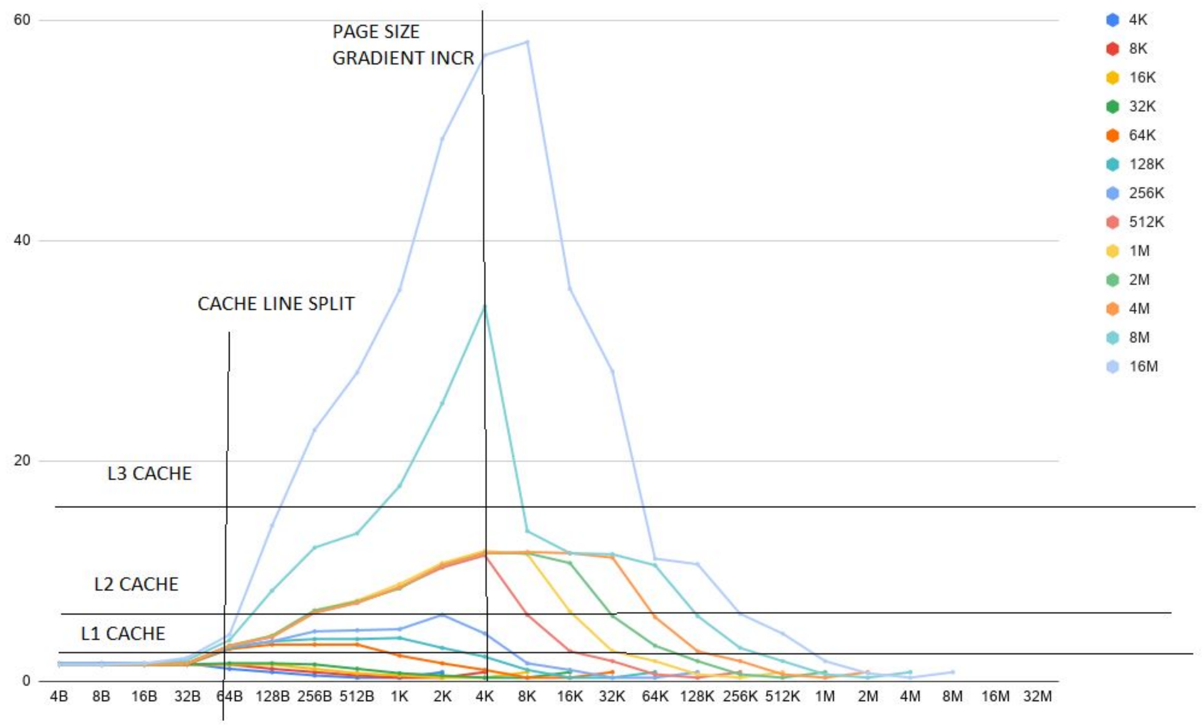
For personal computer:

Each row is size of array and each column is stride length

	4B	8B	16B	32B	64B	128B	256B	512B	1K	2K	4K	8K	16K	32K	64K	128K	256K	512K	1M	2M	4M	8M	16M	32M
4K	1.6	1.6	1.6	1.6	1.1	0.8	0.5	0.3	0.3	0.8														
8K	1.5	1.5	1.5	1.5	1.5	1.1	0.8	0.5	0.3	0.3	0.8													
16K	1.5	1.5	1.5	1.5	1.5	1.5	1.1	0.7	0.5	0.3	0.3	0.8												
32K	1.5	1.5	1.5	1.5	1.6	1.6	1.5	1.1	0.7	0.5	0.3	0.3	0.8											
64K	1.5	1.5	1.5	1.5	2.9	3.3	3.3	3.3	2.3	1.6	1	0.3	0.3	0.8										
128K	1.5	1.5	1.5	1.6	3	3.6	3.8	3.8	3.9	3	2.2	1	0.3	0.3	0.8									
256K	1.5	1.5	1.5	1.6	3.1	3.6	4.5	4.6	4.7	6	4.3	1.6	1	0.3	0.3	0.8								
512K	1.5	1.5	1.5	1.6	3.2	4	6.2	7.1	8.5	10.3	11.4	6	2.7	1.8	0.6	0.3	0.8							
1M	1.5	1.5	1.5	1.6	3.2	4.1	6.4	7.3	8.8	10.7	11.8	11.5	6.3	2.7	1.8	0.6	0.3	0.8						
2M	1.5	1.5	1.5	1.6	3.2	4.1	6.4	7.2	8.4	10.5	11.6	11.6	10.7	5.9	3.2	1.8	0.6	0.3	0.8					
4M	1.5	1.5	1.5	1.6	3.2	4	6.2	7.1	8.5	10.	11.	11.	11.	11.	5.8	2.7	1.8	0.6	0.3	0.8				

										5	6	7	6	2										
8M	1.5	1.5	1.6	1.9	3.7	8.2	12.1	13.4	17.7	25.2	34	13.6	11.6	11.5	10.5	5.9	3	1.8	0.6	0.3	0.8			
16M	1.5	1.5	1.6	2.1	4.2	14.1	22.8	28	35.5	49.2	56.8	58	35.6	28.1	11.1	10.6	6.1	4.3	1.8	0.7	0.3	0.8		
32M	1.5	1.5	1.6	2.1	4.4	16	27	33.7	41.7	67.8	80.5	87.7	59.8	56.4	49.7	32.4	10.2	6.9	4.6	2.7	0.7	0.3	0.8	
64M	1.5	1.6	1.6	2.1	4.4	16.4	27.9	35.4	44.2	77.1	91.9	93.5	84.7	80.6	68.4	71.2	65.7	21.4	7.1	4.9	3.2	0.7	0.3	0.8

CACHE LINE, CACHE SIZE, PAGE SIZE



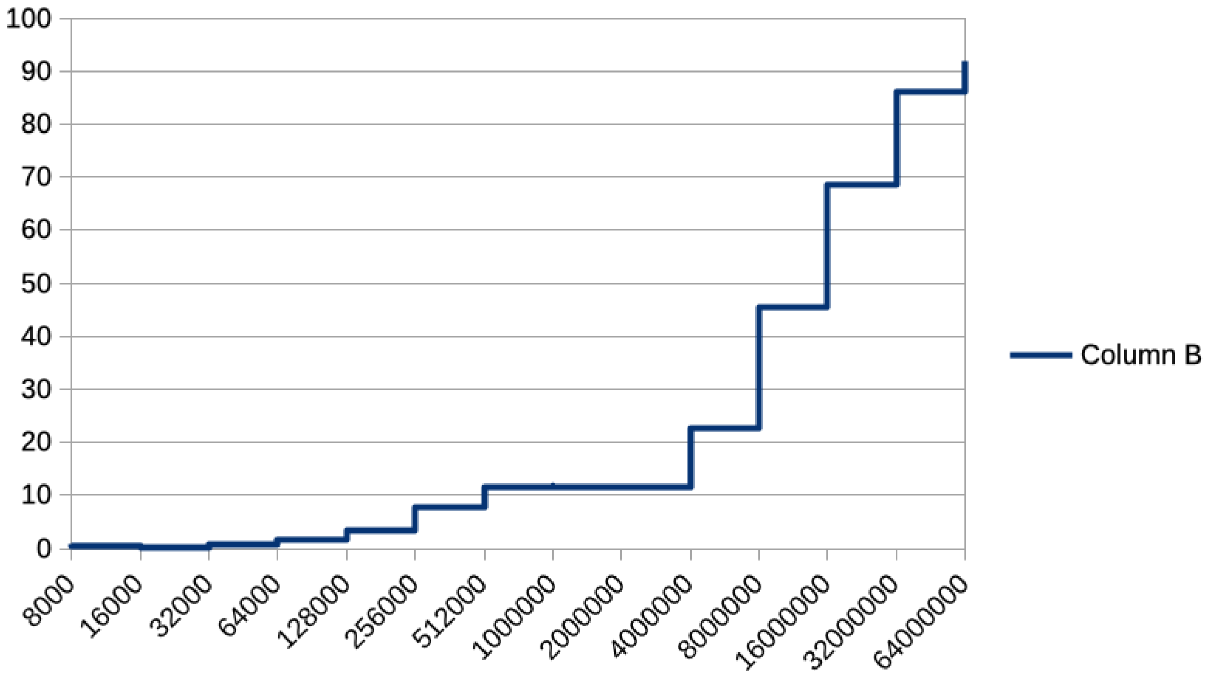
Graph is plotted for read time against stride length for different array sizes.
Stride length is in x axis. Different Array sizes are labeled with different colour lines.

We can see that results diverge from 64B stride hence cache line is 64B. Before 64B strides there would be no cache miss for any access as the array will be prefetched.

We see the first major jump in time from 32K entries, then from 256K entries and then 8M Hence cache sizes are 32K, 256K and then 8M. The jumps are marked with black lines. We were able to arrive at the results as at each jump the level of cache it corresponded to become unable to serve each access and an abrupt increase in reading time was observed

We see that till 4K stride we get huge increases in time for huge datasets, hence page size is 4K. This is visible in the graph above as a slight increase in miss service time for large data sets, and is 4KB for the graph above.

TLB



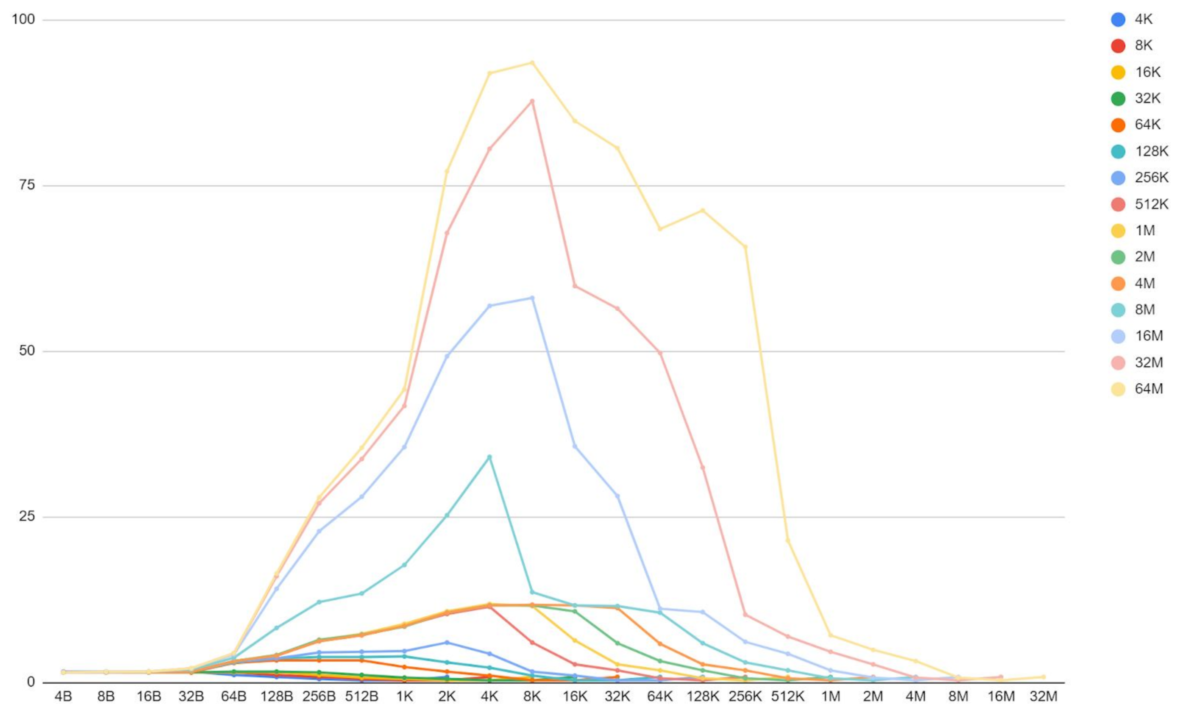
Graph is plotted for read time against array sizes for stride = 4k
Array size is in x axis.

We see that for 4k strides (page size), we get first major step at 256K.
Hence tlb has 64 entries = 256k/4k.
This is possible as for size greater than 256k tlb will be fully utilized and page addrsess will have to fetched into tlb.

MISS PENALTY

Miss penalty for 32K = 3.3 -1.6 = 1.7 ns
Miss penalty for 256K = 8.4 - 4.7 = 3.7 ns
Miss penalty for 8M = 28 - 13.4 = 16.6 ns
The miss penalty was of say 32K cache level was calculated by subtracting the stable read time of 32K size from 64K size.

PAGE FAULT TIME



Graph is plotted for read time against stride length for different array sizes.
Stride length is in x axis. Different Array sizes are labeled with different colour lines.

Page fault time: 91.9 - 4.3 = 86.6 ns
The page fault is calculated was calculated by very large size array’s read time at 4k strides - 256k array size read time at 4k strides. In this way every consecutive access will read different page. For 256k there will be no page fault as all addresss will be in 64 pages of 4k page size each in tlb. When array size is large, every read will have page fault and hence page fault in every consecutive read. In this way we can calculate page fault time.

SET ASSOCIATIVITY OF CACHE

Set associativity of cache was found out by getconf -a | grep CACHE command
L1d cache: 8
L2 cache: 4
L3 cache: 16

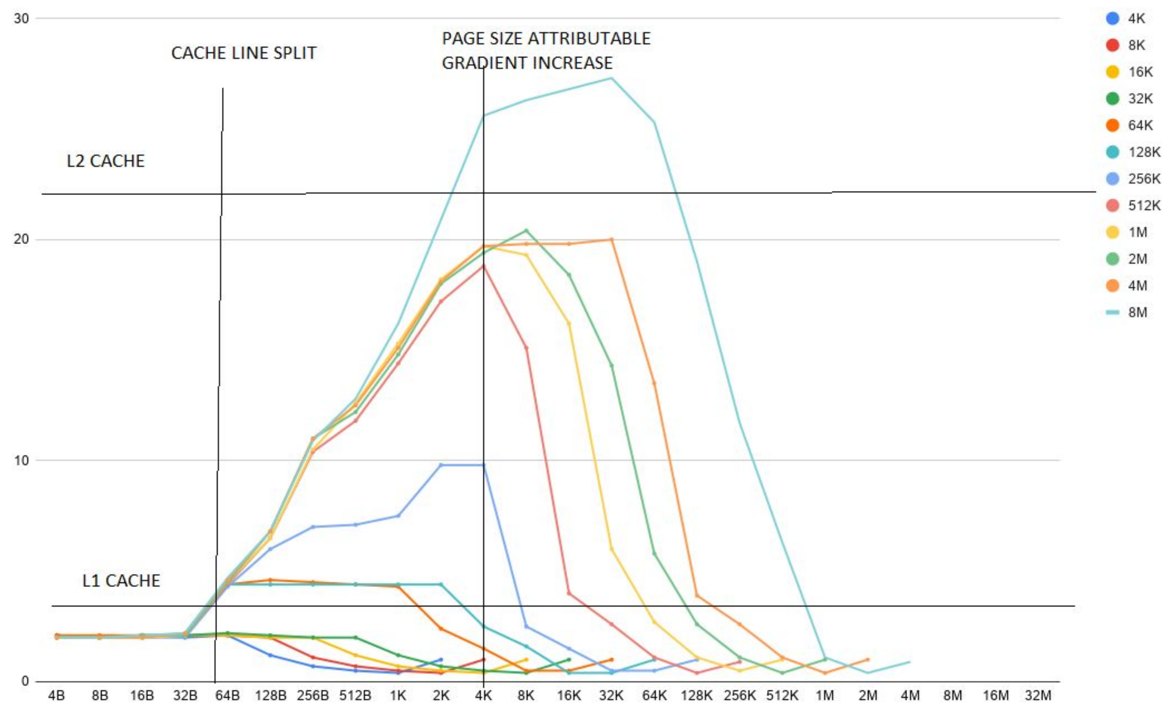
For Baadal Server

Each row is size of array and each column is stride length

	4B	8B	16B	32B	64B	128B	256B	512B	1K	2K	4K	8K	16K	32K	64K	128K	256K	512K	1M	2M	4M	8M	16M	32M	
4K	2.1	2	2.1	2	2.1	1.2	0.7	0.5	0.4	1															
8K	2.1	2	2	2.1	2.1	2	1.1	0.7	0.5	0.4	1														
16K	2	2	2	2.1	2.1	2	2	1.2	0.7	0.5	0.4	1													
32K	2.1	2	2.1	2.1	2.2	2.1	2	2	1.2	0.7	0.5	0.4	1												
64K	2.1	2.1	2.1	2.1	4.4	4.6	4.5	4.4	4.3	2.4	1.5	0.5	0.5	1											
128K	2	2	2	2.1	4.4	4.4	4.4	4.4	4.4	4.4	2.5	1.6	0.4	0.4	1										
256K	2	2	2	2	4.3	6	7	7.1	7.5	9.8	9.8	2.5	1.5	0.5	0.5	1									
512K	2	2	2.1	2.1	4.4	6.5	10.4	11.8	14.4	17.2	18.8	15.1	4	2.6	1.1	0.4	0.9								
1M	2	2	2	2.1	4.5	6.5	10.5	12.6	15.3	18.2	19.7	19.3	16.2	6	2.7	1.1	0.5	1							
2M	2	2	2.1	2.1	4.5	6.8	11	12.2	14.8	18	19.4	20.4	18.4	14.3	5.8	2.6	1.1	0.4	1						

4M	2	2	2	2.2	4.6	6.8	11	12.5	15.1	18.1	19.7	19.8	19.8	20	13.5	3.9	2.6	1.1	0.4	1					
8M	2	2	2.1	2.2	4.7	6.8	10.9	12.8	16.2	20.9	25.6	26.3	26.8	27.3	25.3	19	11.7	6.3	1.1	0.4	0.9				
16M	2	2.1	2.3	2.4	5.1	7.3	12.4	14	17.7	22.1	27.4	27.9	28.8	31.6	30.1	29.9	23.7	14.7	7.5	1.1	0.4	0.9			
32M	2.1	2.2	2.7	4	8.3	25.3	36.3	38.5	44	51.8	55.1	28.7	29.4	31.6	31.1	31	30.6	21.4	14.2	7.4	1.1	0.4	0.9		
64M	2.1	2.3	2.9	4.6	9.8	30	51	63.7	80.7	101.7	113.5	60.3	34.4	37.8	34.8	34	34	30	23.9	15.4	7.5	1.1	0.3	0.9	

CACHE LINE, CACHE SIZE, PAGE SIZE



Graph is plotted for read time against stride length for different array sizes.
Stride length is in x axis. Different Array sizes are labeled with different colour lines.

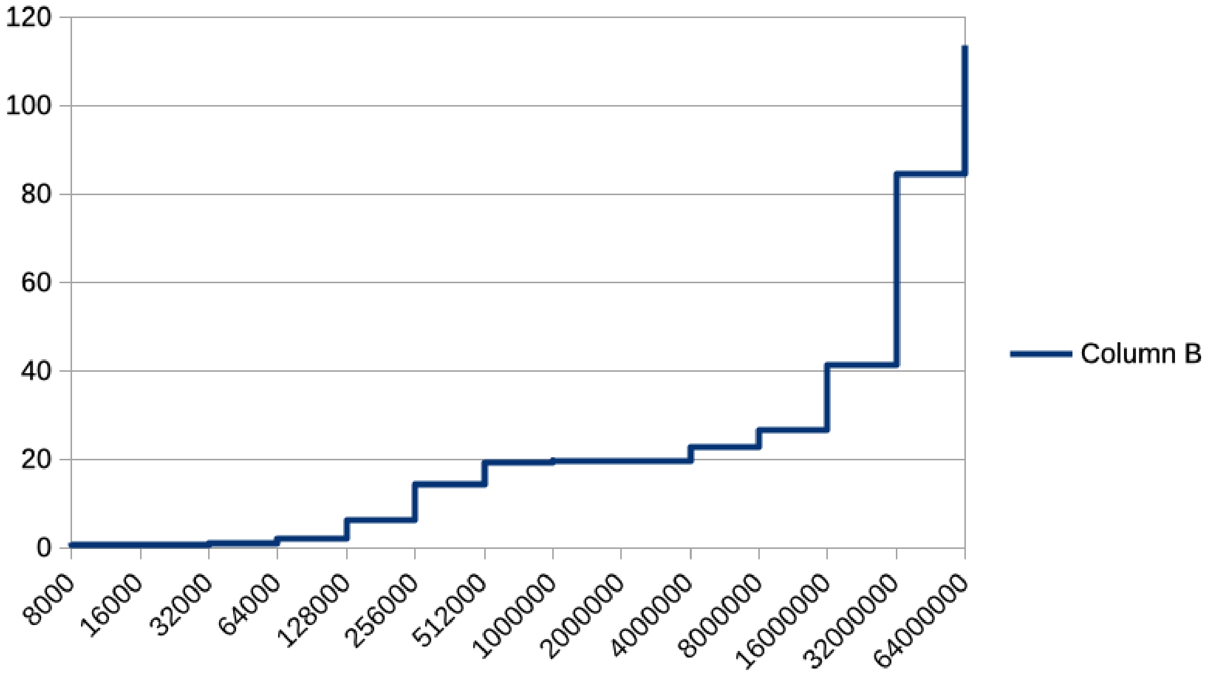
We can see that results diverge from 64B stride hence cache line is 64B.

We see first major jump in time from 32K entries, then from 4M entries
Hence cache sizes are 32K and then 4M

We see that there is a increase after 256k too but that can be attributable to the contention on server. Or lscpu may be be giving the wrong configuration for server. There could also be a level of caching between L1 and L2 we do not know about.

We see that till 4K stride we get huge increases in time for huge datasets, hence page size is 4K

TLB



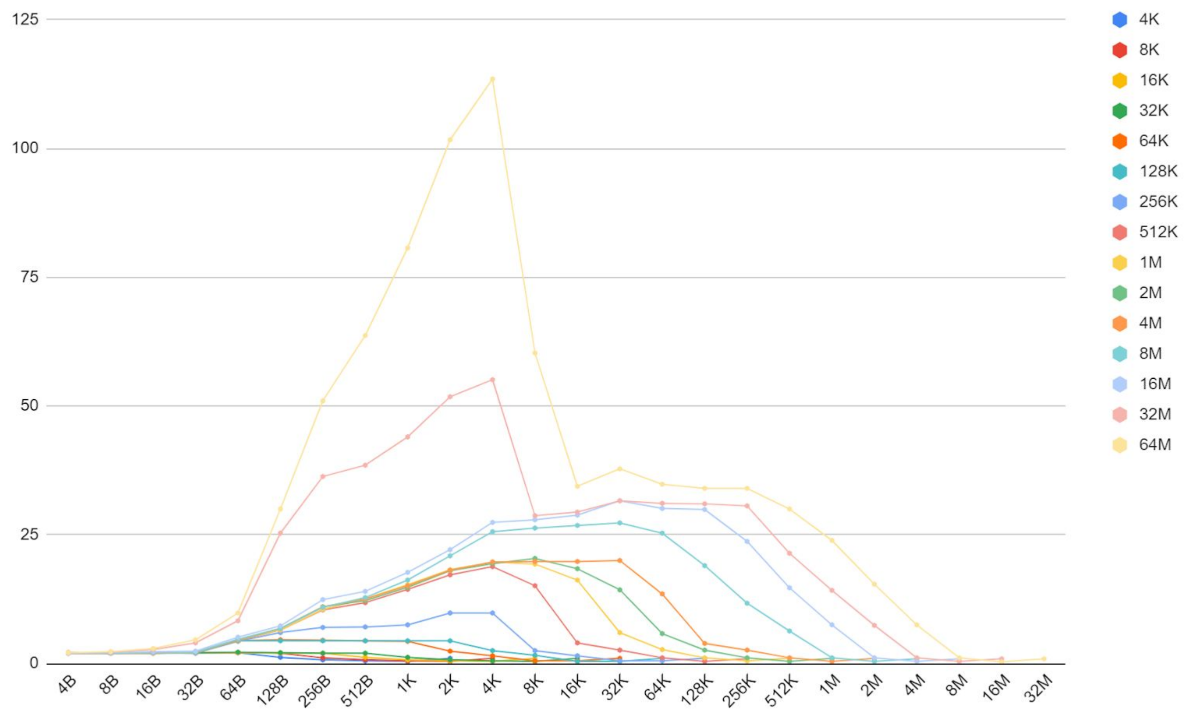
Graph is plotted for read time against array sizes for stride = 4k
Array size is in x axis.

We see that for 4k strides (page size), we get first major step at 256K.
Hence tlb has 64 entries = 256k/4k.

MISS PENALTY

Miss penalty for 32K = 4.4-2 = 2.4 ns
Miss penalty for 4M= 25.6- 19.7 = 7.9 ns
The miss penalty was of say 32K cache level was calculated by subtracting the stable read time of 32K size from 64K size.

PAGE FAULT TIME



Graph is plotted for read time against stride length for different array sizes.
Stride length is in x axis. Different Array sizes are labeled with different colour lines.

Page fault time: 113.5 - 9.8= 103.7ns
The page fault is calculated was calculated by very large size array's read time at 4k strides - 256k array size read time at 4k strides. In this way every consecutive access will read different page. For 256k there will be no page fault as all addresss will be in 64 pages of 4k page size each in tlb. When array size is large, every read will have page fault and hence page fault in every consecutive read. In this way we can calculate page fault time.

SET ASSOCIATIVITY OF CACHE

Set associativity of cache was found out by getconf -a | grep CACHE command
L1d cache: 8
L2 cache: 8

Results

We have properly justified and found out each parameter asked in lab programmatically for personal computer and a server computer. Server computer showed level of cache between L1 and L2 which can not be explained.

Other estimatable parameters

- ***Memory Size***
  - If the size of array is increased to that of physical memory we might be able to get the size of physical memory.
- ***Instruction cache size:***
  - We can try to exhaust instruction cache of 32K size by utilizing more and more instructions
- ***Number of processors***
  - For a constant value of number of process * array size we can verify similar read time. But if we increase number of process above a limit we will observe a suddent increase in read time. This can be used to estimate number of processors

References

Program from Computer Architecture: A Quantitative Approach 5th edition.