Assignment-5: Cache Model, Address Translation and TLB

Arnab Das, u1014840

April 10, 2018

1 *Virtually Indexed Cache*

The virtual address space is decomposed primarily into two fields, namely the virtual page number and the offset field. The virtual page number is used as the key to lookup the TLB or in case of TLB miss the page table, to find the corresponding page number of the physical address. Generally the virtual page and the physical page are of the same size, in which case thje offset field of the virtual address space directly maps to the "index"+"byte" field of the physical address, hence the offset can be used directly to index inside the cache, while in parallel the virtual address translation proceeds which is called the virtually indexed physically tagged.

However, several complications arise if the page offset does not equals the (index+byte) field of the physical address. This results in the physical page size and the virtual page size being different and hence we cannot directly use the virtual address offset to index into the cache. Two cases may arise due to this. Firstly, the virtual page size is larger than the the physical page(frame) size, or secondly the virtual page size is less than the physical page(frame) size. Both the situations are discussed below.

Case-1: Page offset less than the (index+byte) In this situation more than one virtual addresses(VA) can map to the same physical address. This situation has two side effects, one relating to stale data and the other relating to memory utilization.

- If more than one VA maps to the same PA, then it is possible to have multiple copies of the same physical address in TLB and stale data in the cache. For example, suppose adress PQR and XYZ both map to the same physial address ABC. First one process generates a request for PQR, its a cache miss and data is fetched from main memory, and the cache and TLB are both updated. Next if a request is generated for XYZ, it will again be a miss since it's entry does not exists in the TLB, although it's corresponding physical address ABC is already in the cache. This results in a new copy of ABC being accessed from memory, and TLB and cache being updated with a redundant copy. Now suppose a write happens to PQR, it will update its corresponding copy of the physical memory in the cache, but XYZ will still be pointing to the stale value of ABC. This is referred to as the *synonym* or **aliasing** problem.
- Also, since the page offset is less than the (index+byte) field, only the offset field will be used to access into the physical frames in

the cache. This results in underutilization of cache, since the full block is not being indexed. For example, suppose the page offset is 10 bits, where 8 bits are for indexing and 2 bits are for byte access. If the physical page, that is, the cache set size is larger say 12 bits, with 10 index bits and 2 offset bits, only the lower order 8 bits of index field would be valid corresponding to the 8 index bits of the offset field of the virtual page.

Solution: To solve the *aliasing* problem, we need to ensure primarily that multiple VA's do not map to same PA, or even if they do, there is a coherency protocol to be maintanied to updated the PA when any of the corresponding VA's existing in the tlb gets accessed. Thus we can do the following

- If same data exists in multiple cache lines, do a search in the cache to update all copies of the data, whenever any of the corresponding VA is accessed.
- To avoid the synonym problem, set the cache size to be equal to the page size times the associativity. This ensures that the page size is equal to the cache set size. If not, then multiple copies of the same data can exists in different cache sets.
- Requiring that all VA's that map to same PA, are made to access the same cache set, using cache coloring, where the corresponding physical address bits are artificially made to be equal to be the same as the virtual address bits used for indexing.
- Using page splintering, where the guest uses large page sizes but the hypervisor in the OS can limit those page sizes to match the frame size granularity.

Case-2: Page offset larger than the (index + byte) In this situation the virtual page size is larger than the physical frame size. In case of a single process system this is not an issue, since the virtual page number uniquely idenitfies a physical page, but the indexing into the cache needs to happen using the page offset of the physical address which is equal to the lower index bits of the virtual address. For example if the page offset is 12 bits with 10 bits index and 2 bits byte access while the physical address has 8 bits index and 2 bits offset, the indexing into the cache is required to happen with 8 bits index and 2 bits byte offset. The 2 extra index bits in the offset will be redundant, since the virtual page number uniquely identifies the physical page. However, in case of a multiprocessing/multitasking system **this may not hold**. Suppose , in this example we have a 22 bit address, which means the virtual tag is 10 bits while the physical tag is 12 bits. Thus, virtual address from different processes can have the same virtual tag of 10 bits, mapping to different PA's, hence the tag does not uniquely identifies the cache data. This is called the *homonym* problem leading to incorrect data access.

Solutions To prevent the homonym problem,

- Additionally tag the virtual addresses with the corresponding address space identifier to identify that they belong to different physical addresses even though the virtual tags match.
- Or, one can use the physical tags to correctly index into the cache after translation from the tlb. This introduces extra cycles in the critical path since the translation first happens and then based on the translated address, the cache is indexed.
- Since this issue is mainly pertinent to multiprocess systems, whenever there is a context switch, flush the cache to ensure avoiding incorrect data access.
- Single address space operating system(SASOS) : In SASOS design, there is a single large address space use by all the processes, and maps this large space onto physical memory. A single address space ensures virtual addresses from different preocesses do not have the same virtual tag, thus avoiding the conflict.

2 *Cache and Memory Model using CACTI)*

CACTI-6.5 was used to analyze the impact of the given L1 and L2 parameters. For a 4GB adress space, number of address bits is 32. The tag bits were computed from the given cache size as follows

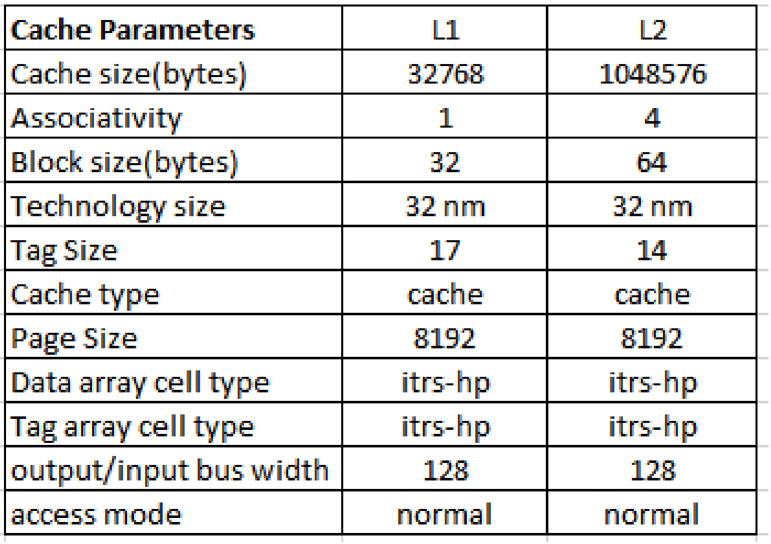
L1-cache configuration

$$\begin{aligned}
\text{Cache Size} &= 32KB = 2^{15}B \\
\text{Block Size} &= 32B = 2^{5}B \\
\text{\# of blocks} &= \frac{Cache\ Size}{Block\ Size} = 2^{10} \\
\text{\# index bits} &= 10 \\
\text{\# byte offset bits} &= 5 \\
\text{\# Tag bits} &= 32 - 15 = \mathbf{17}
\end{aligned}$$

L2-cache configuration

$$\begin{aligned}
\text{Cache Size} &= 1MB = 2^{20}B \\
\text{Cache line} &= 64B = 2^{6}B \\
\text{Associativity} &= 4 \\
\text{\# of cache sets} &= \frac{\textit{Cache Size}}{\textit{Cache set size}} = 2^{20-(6+2)} = 2^{12} \\
\text{\# index bits} &= 12 \\
\text{\# byte offset bits} &= 6 \\
\text{\# Tag bits} &= 32-(12+6) = \mathbf{14}
\end{aligned}$$

Cacti was run with the above tag bits and the configuration specified in figure: 1.



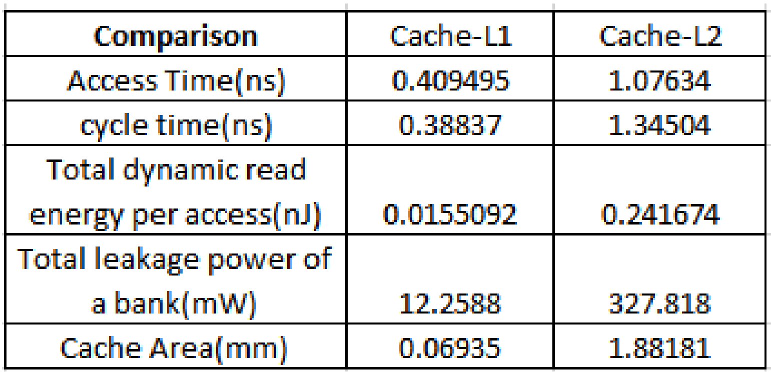
Cache Parameters	L1	L2
Cache size(bytes)	32768	1048576
Associativity	1	4
Block size(bytes)	32	64
Technology size	32 nm	32 nm
Tag Size	17	14
Cache type	cache	cache
Page Size	8192	8192
Data array cell type	itrs-hp	itrs-hp
Tag array cell type	itrs-hp	itrs-hp
output/input bus width	128	128
access mode	normal	normal

Figure 1: L1 and L2 Configuration

- The input-output bus-width was chosen to be 128 bits such that an entire block of L1-cache(16 B) can be moved in a burst.
- The Data and tag array cell type is chosen for high power as per the ITRS(International Technology Roadmap for Semiconductors) standard.
- For comparison between L1 and L2 cache , the access mode is set to normal. Later we compare between normal and fast access mode for L2 cache

Figure: 2 shows the table for comparing the access time, energy, leakage power and area of L1 and L2 caches.

Below are the observations made from the comparative study



Comparison	Cache-L1	Cache-L2
Access Time(ns)	0.409495	1.07634
cycle time(ns)	0.38837	1.34504
Total dynamic read energy per access(nJ)	0.0155092	0.241674
Total leakage power of a bank(mW)	12.2588	327.818
Cache Area(mm)	0.06935	1.88181

Figure 2: L1 and L2 Comparison

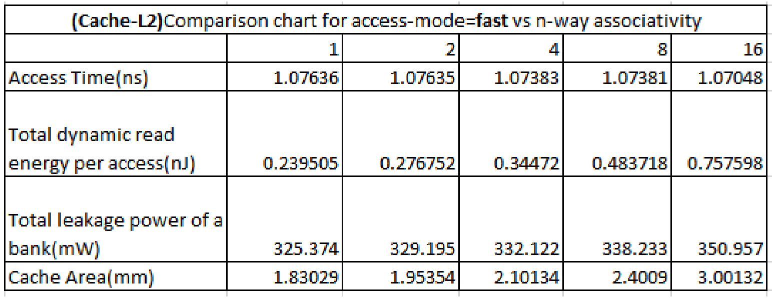
- We expect the access time of L1-cache to be much faster than the L2-cache. This is because the L1 cache is much smaller in size and hence the data access takes shorter time for L1 than L2.
- Since the L2-cache is a much larger structure with 4 cache lines, the data path to access an L-2 cache element requires more work since it is required to travel a larger circuitry. This is reflected by the dynamic read energy per access.
- Also, the leakage power consumption from L2 will be higher since it is built of sram that has high power consumption to hold the charge level in the component transistors. Although L1 is built of sram as well, its smaller structure makes its leakage power consumption much lower than L2.
- The cache area of the larger cache with multiple cache lines is expected to be higher, hence L2 has larger cache area than L1.

L2-cache associativity

Here we perform a comparative study for L2-caches with different values of n for its n-way associativity. The associativity is varied between $1 - 16$ in powers of 2, and the tag size is adjusted accordingly as shown

$$\begin{aligned}
\text{1-way} &= 12\ \textit{Tag bits}\\
\text{2-way} &= 13\ \textit{Tag bits}\\
\text{4-way} &= 14\ \textit{Tag bits}\\
\text{8-way} &= 15\ \textit{Tag bits}\\
\text{16-way} &= 16\ \textit{Tag bits}
\end{aligned}$$

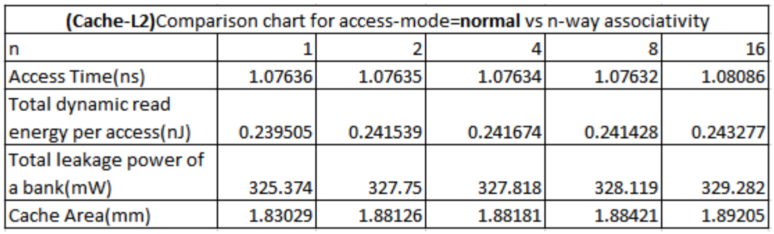
We have run cacti for L2 in both Fast and normal access modes. Figure: 3 and Figure: 4 shows the comparative data for different values of n for fast and normal access modes respectively.



(Cache-L2) Comparison chart for access-mode=**fast** vs n-way associativity

	1	2	4	8	16
Access Time(ns)	1.07636	1.07635	1.07383	1.07381	1.07048
Total dynamic read energy per access(nJ)	0.239505	0.276752	0.34472	0.483718	0.757598
Total leakage power of a bank(mW)	325.374	329.195	332.122	338.233	350.957
Cache Area(mm)	1.83029	1.95354	2.10134	2.4009	3.00132

Figure 3: n-way comparison for fast access



(Cache-L2) Comparison chart for access-mode=**normal** vs n-way associativity

n	1	2	4	8	16
Access Time(ns)	1.07636	1.07635	1.07634	1.07632	1.08086
Total dynamic read energy per access(nJ)	0.239505	0.241539	0.241674	0.241428	0.243277
Total leakage power of a bank(mW)	325.374	327.75	327.818	328.119	329.282
Cache Area(mm)	1.83029	1.88126	1.88181	1.88421	1.89205

Figure 4: n-way comparison for normal access

Figure: 5, Figure: 6, Figure: 7 and Figure: 8 show the comparative graphs for access time, energy, leakage power and cache area respectively in nomral and fast mode access for varying associativity.

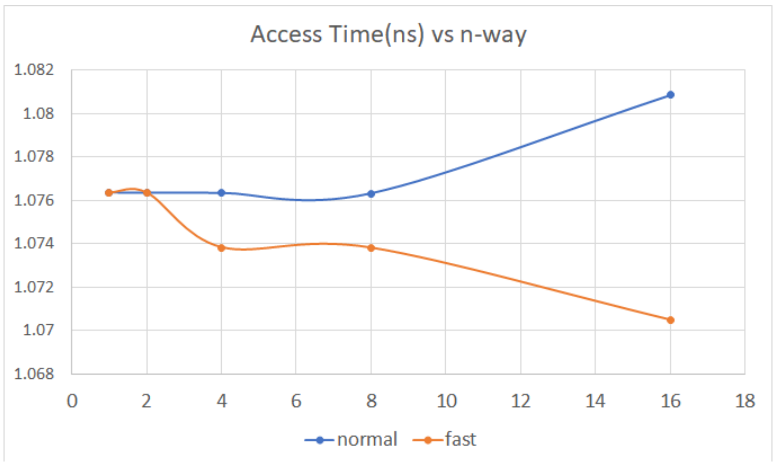


Figure 5: n-way L2 access time

The observations are listed below

- The access time tends to improve with increasing the associativity both in normal and fast mode, however in fast mode the drop in access time is much faster. Also, increasing associativity beyond 8 increases the access time in normal mode. In fast mode we see that drop in access time continues at n=16, but the drop rate reduces indicating it is reaching a saturation point beyond which

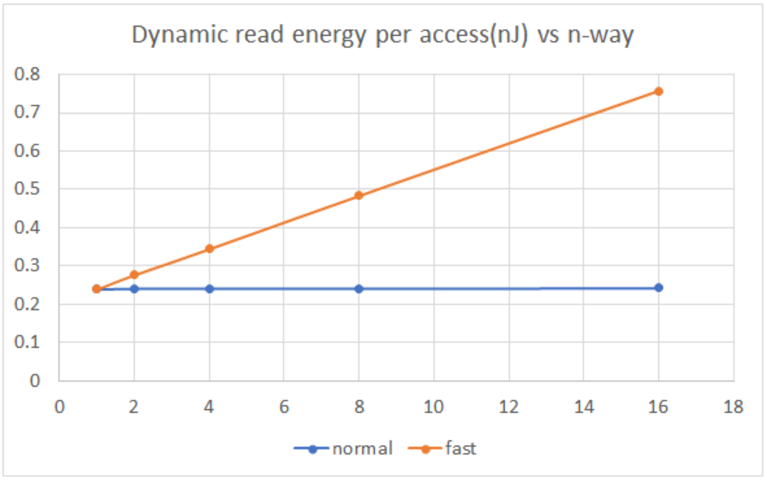


Figure 6: n-way L2 Dynamic energy per read access

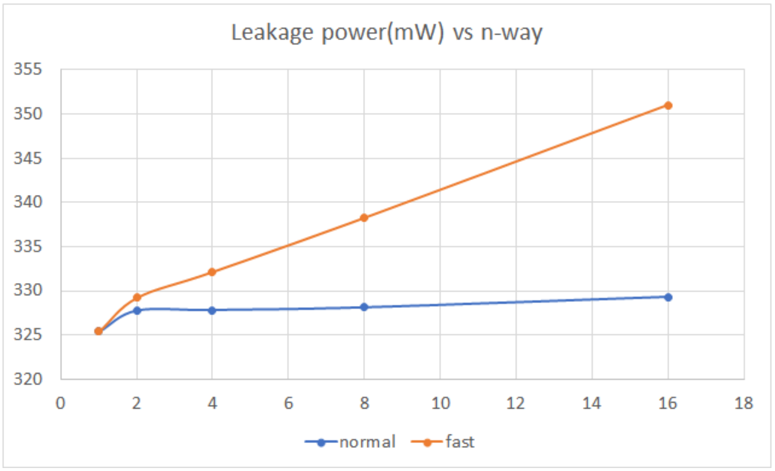


Figure 7: n-way L2 leakage power

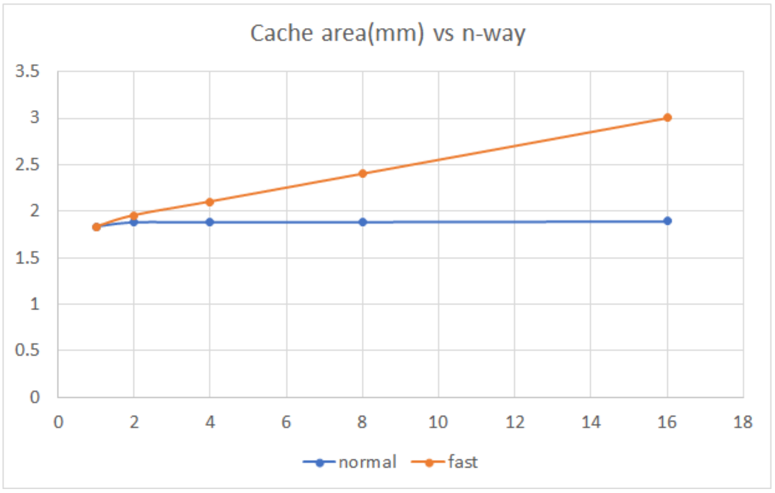


Figure 8: n-way L2 cache area

increasing asociativity results in more overhead than improvement in access time.

- The fast access shows much larger swing in dynamic read energy with increasing cache lines, since with more parallelism, more hardware is active for a read in a given cycle .
- The leakage power also increses with increasing associativity, mainly due to more area of the hardware being active with multiple cache lines. The leakage power is also much higher when in fast access since introducing parallelism increases the volume of active circuitary.
- Increasing the number of cache lines incurs higher overhead in area to include the additional circuitry for the cache line showing the incresing trend in area as we increase the associativity. In fast access mode, even more circuitary is deployed for enhacing parallel access to data hence significantly increasing the area overhead.

3 *References*

- http://www.cse.unsw.edu.au/~cs9242/02/lectures/03-cache/node8.html
- https://www.cs.rutgers.edu/~abhib/binhpham-tr15.pdf
- http://www.cs.utah.edu/~naveen/cacti_report.pdf
- www.semiconductors.org/clientuploads/Research_Technology/ITRS/2005/1_Executive%20Summary.pdf
- http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka16862.html
- https://view.officeapps.live.com/op/view.aspx?src=http://www.cs.utah.edu/~rajeev/cacti6/micro07.ppt
- https://www.freebsd.org/doc/en_US.ISO8859-1/articles/vm-design/page-coloring-optimizations.html
- http://www.cs.rochester.edu/~sandhya/csc256/seminars/lingxiang_page_coloring.pdf