CSE530 Assignment 011

Name- Niramay Vaidya
PSU ID- 939687597
Date- 09/30/21
Time- 11:30 PM

Table of Contents

1. Test Bench Details-

1. Shell Script

The shell script takes the config file path as an input, and it takes either a constant cache size and multiple values of cache associativity as input, or it takes a constant cache associativity and multiple values of cache size as input.
After performing error handling for the command line parameters taken as inputs, it creates a folder structure if it doesn't already exist (to store output and result files, as well as images of the graph plots).
It then gets the line numbers for the size and associativity keys from the provided cache config file, and then based upon which parameter is varying, does some more validity checks on the inputs provided (for example, there is a check in place for both size and associativity which doesn't allow these values to be non powers of 2). It also performs cleanup of the old generated output and result files.
Based upon the obtained line numbers as stated previously, it always replaces the values for these keys according to the inputs provided in the same cache config file (loops sequentially for all values of the varying parameter) and then runs the cacti tool by providing this cache config file as the input. The output and result files are placed in their appropriate locations in the folder structure created previously.
Finally, the python script is called with a varying size or associativity command line input parameter.

Note- In case the shell script is run in zsh, for the 1st time, the check for an empty folder structure to see if there are any older output and result files to be deleted or not will throw an error for the ls command even though stderr has been redirected to /dev/null because apparently zsh ignores redirection to /dev/null in case of file regex expansions. Though, this does not affect the functionality in any way, and for any other shell, it should not even throw this error for the 1st time. Also, python3 has been mentioned in the shell script while running the python script. If python3 has not been aliased to the python 3.x distribution but python has been, a replacement of python3 with python would be required.

2. Python Script

The python script uses the result files generated via the shell script upon running cacti, and gathers the required metrics from these files.
It then plots the graphs accordingly. 2 images (varying size and associativity) are plotted and saved, where each image contains 3 graphs for the 3 measured metrics. Along with this, individual images for each individual graph are also saved (but not plotted).

Note- In order for the python script to run, the matplotlib module needs to be installed.

3. Execution Instructions

```
Usage: source ./testbench.sh --config_file <config_file_path> --
cache_size <size> --cache_associativity <associativity_1>
<associativity_2> ...
OR
Usage: source ./testbench.sh --config_file <config_file_path> --
cache_associativity <associativity> --cache_size <size_1> <size_2>
...
```

```
Examples-
source ./testbench.sh --config_file cache.cfg --cache_size 524288
--cache_associativity 1 2 4 8 16 32
source ./testbench.sh --config_file cache.cfg --
cache_associativity 16 --cache_size 131072 262144 524288
Note-
Provide the cache size in bytes as input/s
```

The above usage output for the test bench can also been obtained by running-

```
source ./testbench.sh –-help
```

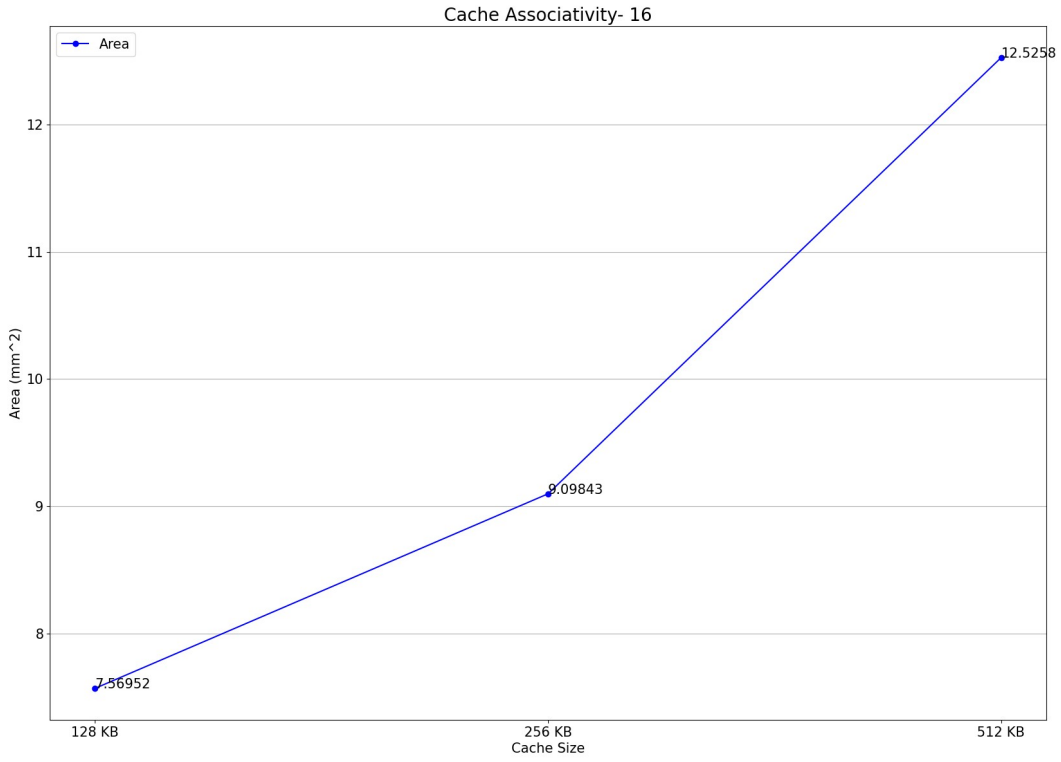
As can be seen above, the note mentions that the test bench expects the input cache size to be provided in bytes. Also, as shown in the examples, the test bench needs to be run twice, once for varying cache size but constant cache associativity, and then the second time for varying cache associativity but constant cache size.

2. Results (Graphs / Plots)-

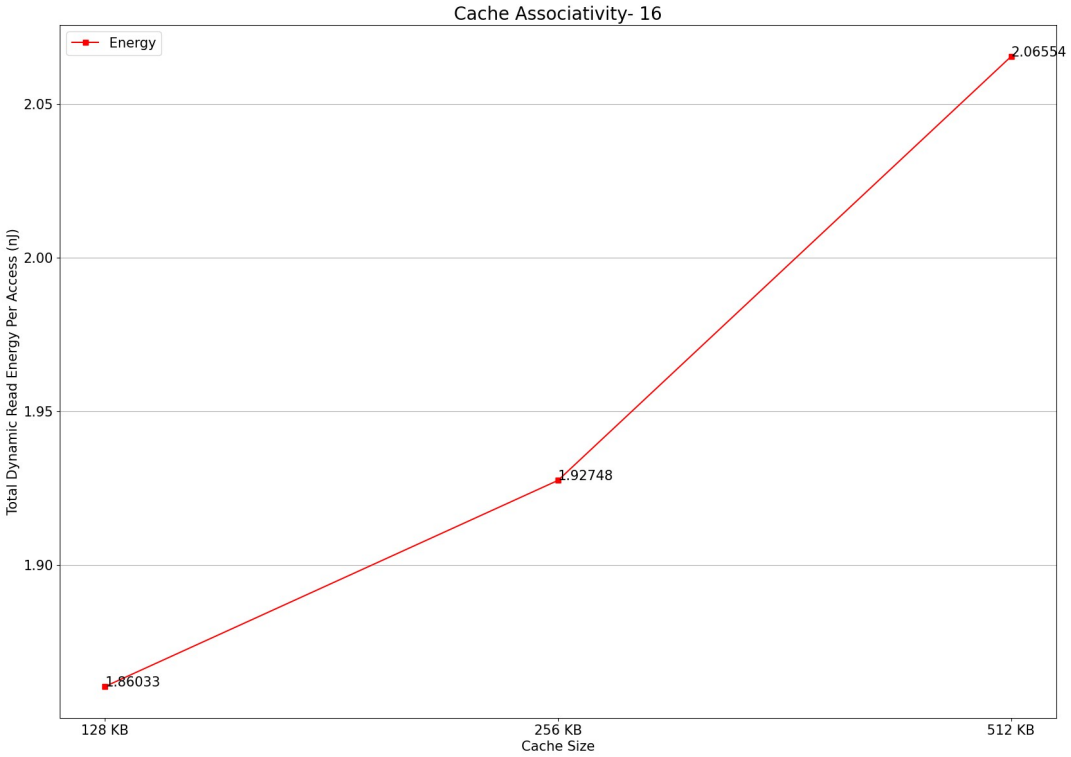
1. Varying Cache Size (Constant Associativity)

Cache Associativity- 16

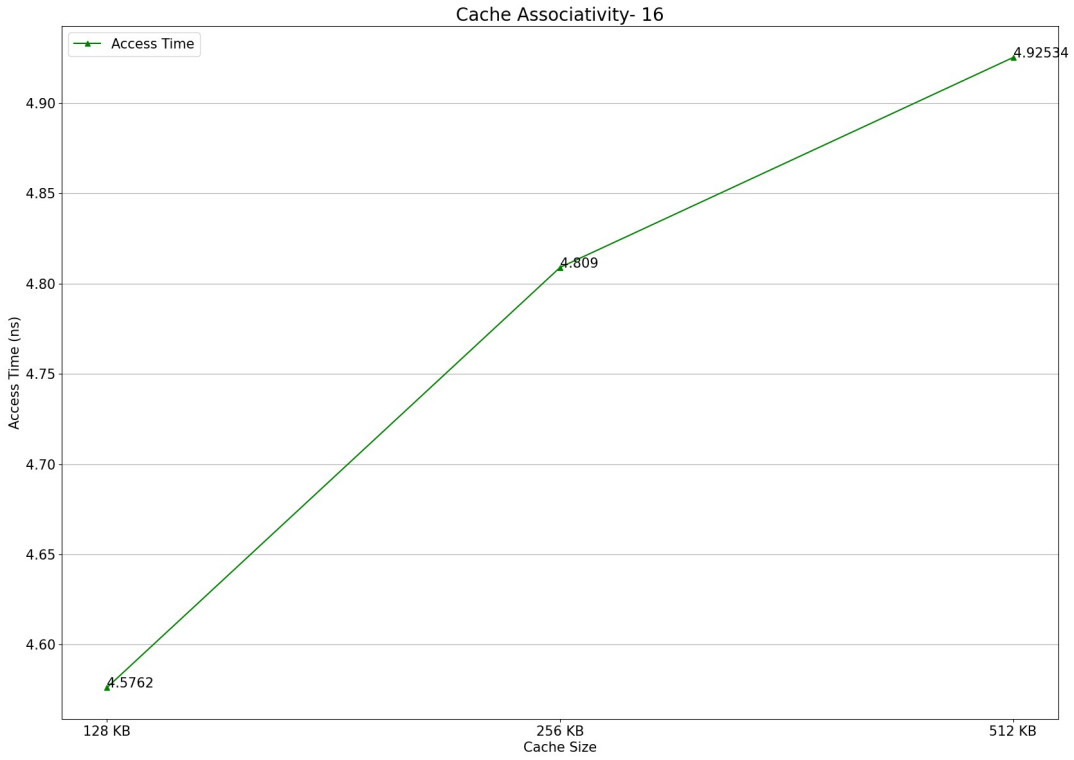
1. Area (mm^2)



2. Total Dynamic Read Energy Per Access (nJ)



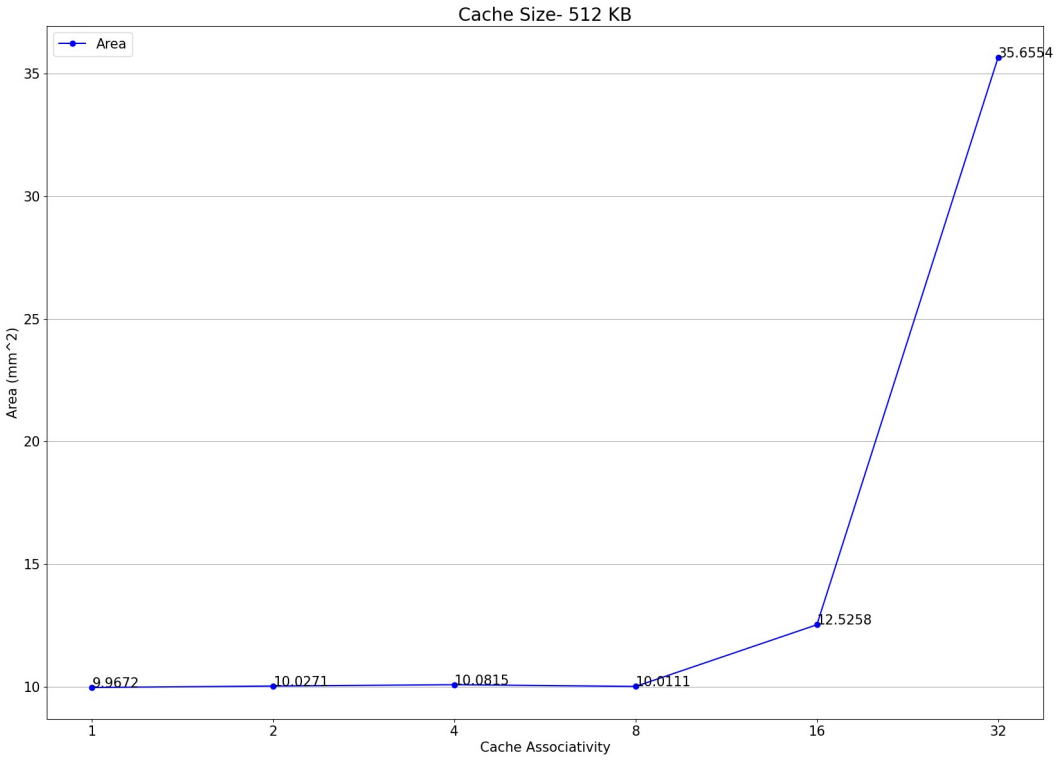
3. Access Time (ns)



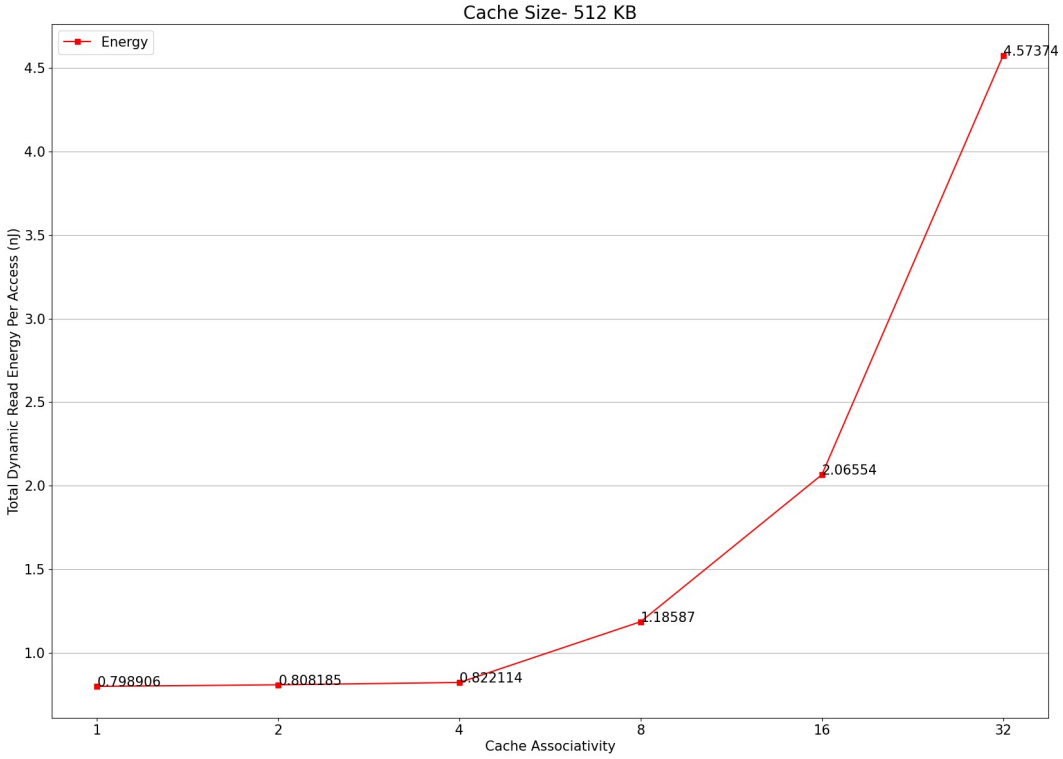
2. Varying Cache Associativity (Constant Size)

Cache Size- 512 KB

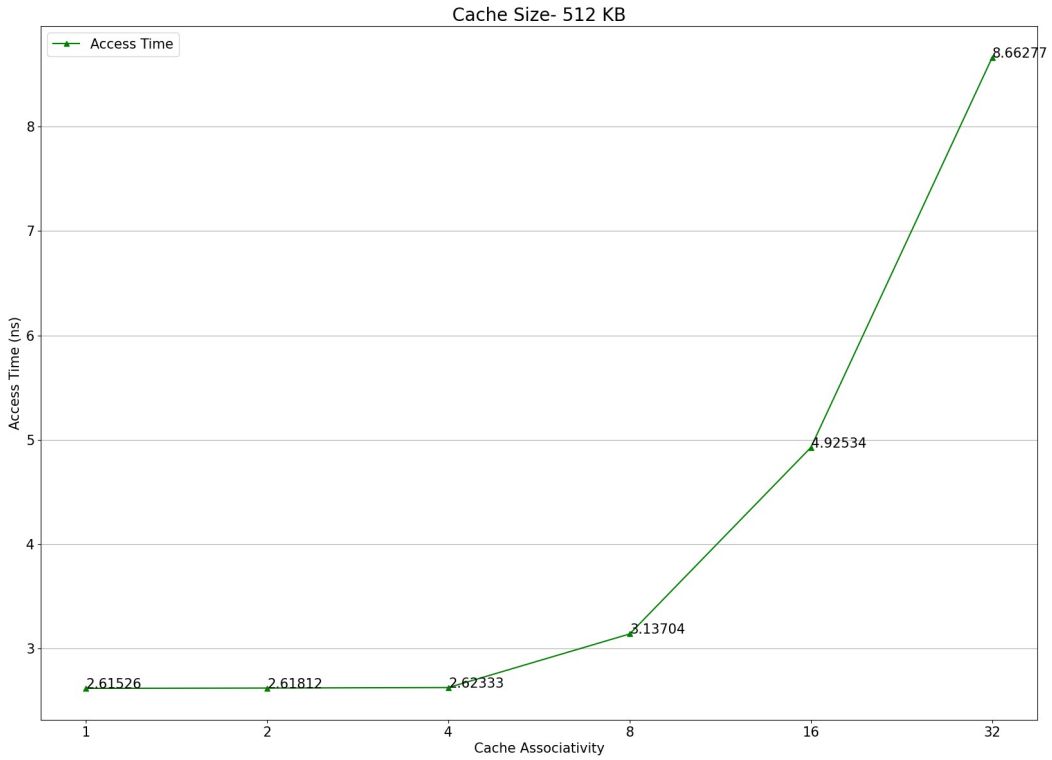
1. Area (mm^2)



2. Total Dynamic Read Energy Per Access (nJ)



3. Access Time (ns)



3. Summary-

Varying Cache Size 128 – 512 KB, Constant Associativity 16			
	Area (mm^2)	Total Dynamic Read Energy Per Access (nJ)	Access Time (ns)
128 KB	7.57	1.86	4.58
256 KB	9.10	1.93	4.81
512 KB	12.53	2.07	4.93
Varying Cache Associativity 1 – 32, Constant Size 512 KB			
1	9.97	0.80	2.62
2	10.03	0.81	2.62
4	10.09	0.82	2.62
8	10.01	1.19	3.14
16	12.53	2.07	4.93
32	35.66	4.57	8.66

4. Explanation-

All simulations have been done for a cache having a single bank.

Area vs Cache Size
The area occupied by a cache increases at a rate in between a linear and an exponential rate with respect to increasing size of the cache, since the cache will now contain more SRAM cells which will occupy more space on-chip.

Access Time vs Cache Size
The access time is related to the physical size of the storage. With a 2-dimensional layout, it is expected that the physical access latency will be roughly proportional to the square root of the capacity i.e. the cache size. As a result, when the cache size increases, the access time is bound to increase and asymptotically approaches a line parallel to the x-axis for very high cache sizes, as difference in the cache sizes then will only reflect a small difference in the corresponding access times (similar to a $y = x^{0.5}$ parabolic graph).

Energy vs Cache Size
Dynamic energy consumption, which comprises of charging and discharging of the load capacitances driven by the switching gates in a CMOS circuit (which makes up an SRAM cell), contributes to most of the total energy dissipation in L1 caches. Cache energy is dissipated during both read and write accesses. The energy per access for both reads and writes is also cache size dependent, showing increase with increasing size. The main reason for this is that different memory wire lengths have different capacitances to be charged. Hence, the energy increases proportionately with an increase in the cache size (with the results showing a rate higher than a linear rate), in case of dynamic read energy per access.

Area vs Cache Associativity
Area on chip increases exponentially with an exponential rise in the associativity because apart from the index address, there is now a tag address for each entry which also needs to be stored, thereby increasing the area needed to store the overall data.

Access Time vs Cache Associativity
The access time exponentially increases with an exponential increase in the associativity (moving from direct mapped to highly set associative till a point when it actually becomes equivalent to being fully associative) since the number of ways in a set increase which results in more time required to search for a block in a particular set once the index has been obtained.

Energy vs Cache Associativity
A majority of the power dissipated by the set associative configurations is caused by two factors, the bit lines and the sense amplifiers. Since the number of sense amplifiers grow with an increase in the associativity at an exponential rate, the energy dissipated also increases exponentially.