**EE 562 Project Report**

Cache Simulator

by

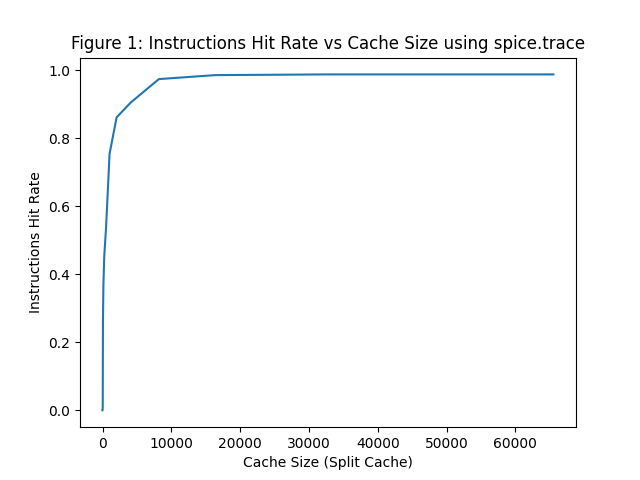
**Name:** Nazmul Haque Turja

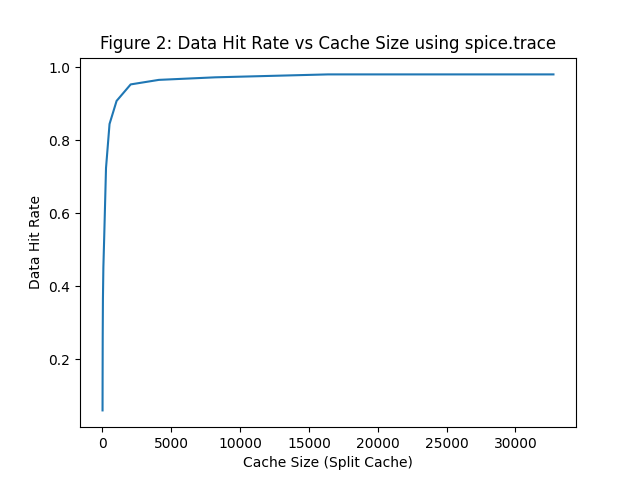
**Aggie ID:** 800779609

**2.1 Working Set Characterization**

2.1.1 Characterization from spice.trace:

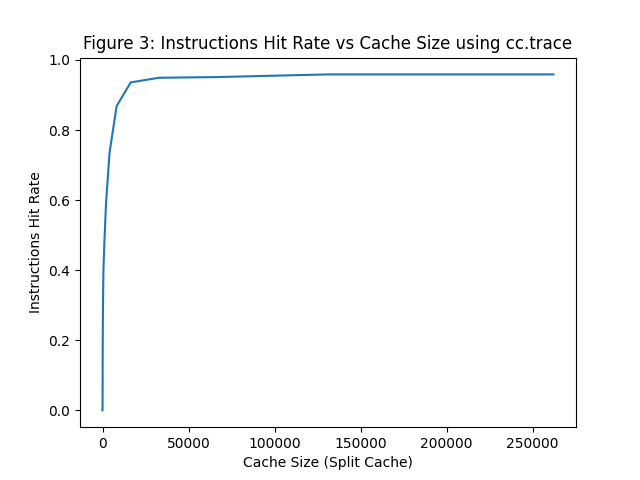
Setting Block Size: 4 B, Write-back cache, Write-allocate, Fully Associative Cache->

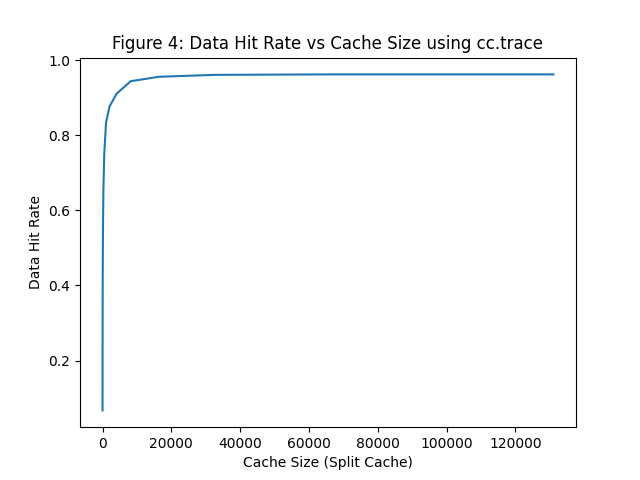




2.1.2 Characterization from cc.trace:

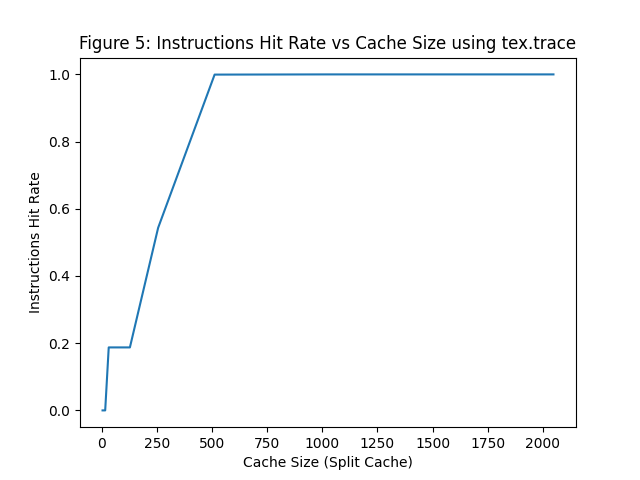
Setting Block Size: 4 B, Write-back cache, Write-allocate, Fully Associative Cache ->

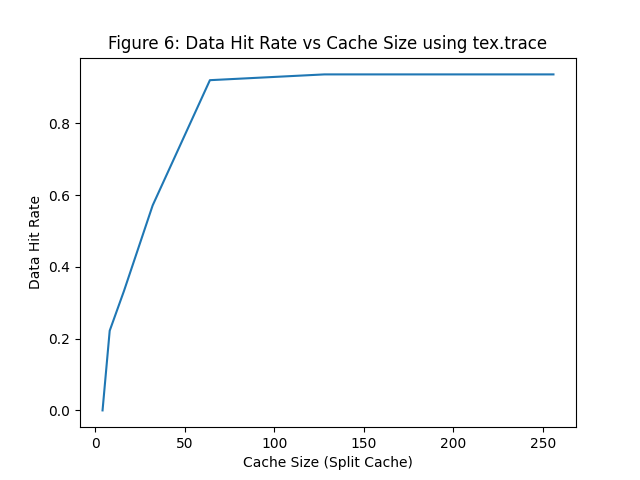




2.1.3 Characterization from tex.trace:

Setting Block Size: 4 B, Write-back cache, Write-allocate, Fully Associative Cache ->





**Ques 1 Ans:** This experiment is used to show the performance of caches of various sizes. In these plots, we can see that, the larger a cache is, the less chance there will be of a conflict. Again this means the miss rate decreases, but the hit time increases.

**Ques 2 Ans:**

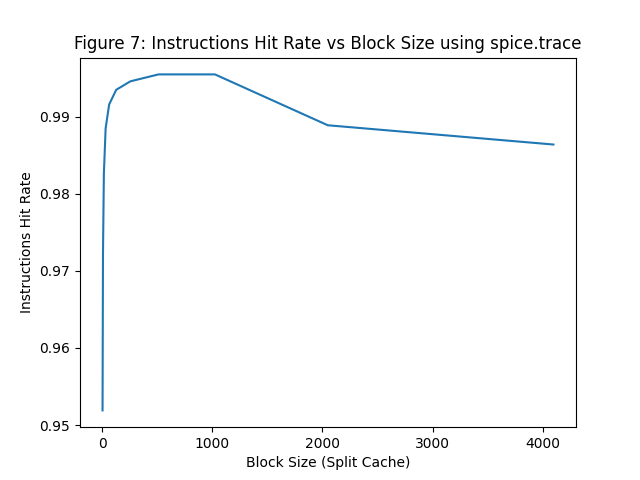
1. For spice.trace, Total Instruction/Data working set size = Cache size (because there is only 1 set for fully associative cache)
2. For cc.trace, Total Instruction/Data working set size = Cache size (because there is only 1 set for fully associative cache)
3. For tex.trace, Total Instruction/Data working set size = Cache size (because there is only 1 set for fully associative cache)

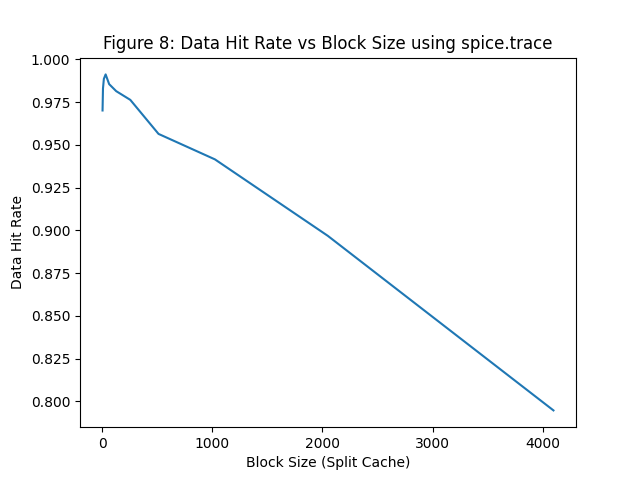
**2.2 Impact of Block Size**

2.2.1 Characterization from spice.trace:

Setting Instruction Cache Size: 8 kB, Data Cache Size: 8kB, Write-back cache,

Write-allocate, Cache Associative 2:

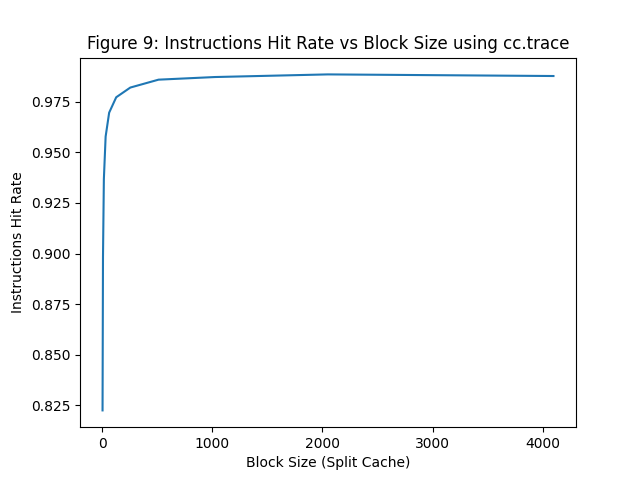


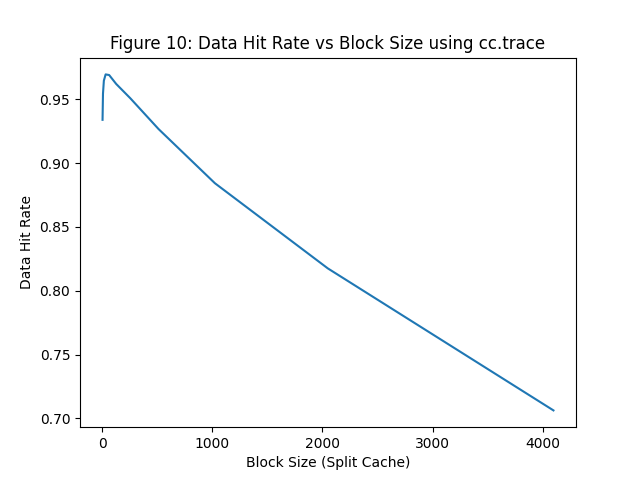


2.2.2 Characterization from cc.trace:

Setting Instruction Cache Size: 8 kB, Data Cache Size: 8kB, Write-back cache,

Write-allocate, Cache Associative 2:

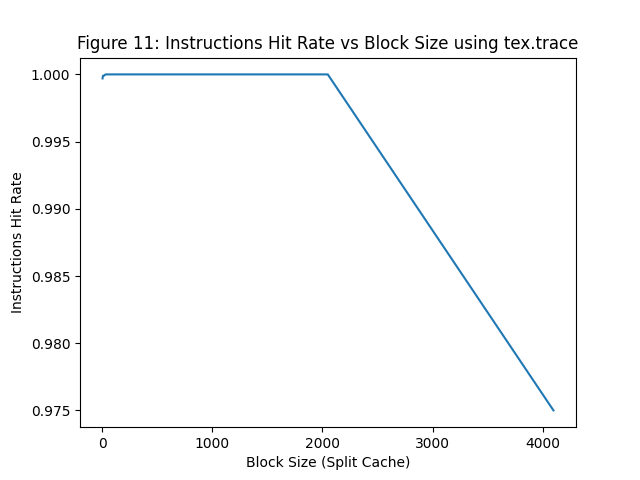


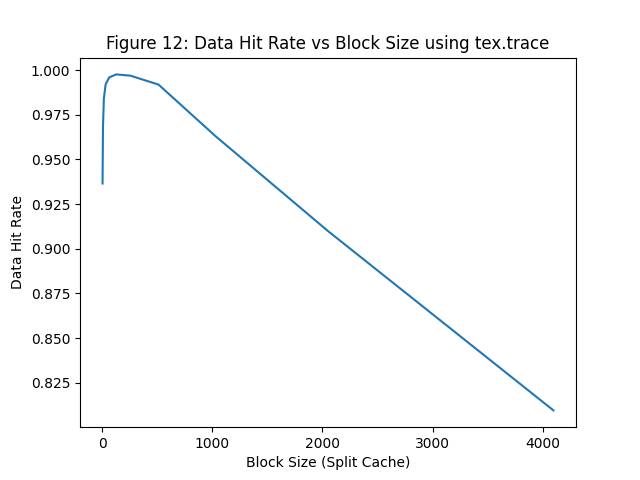


2.2.3 Characterization from tex.trace:

Setting Instruction Cache Size: 8 kB, Data Cache Size: 8kB, Write-back cache,

Write-allocate, Cache Associative 2:





**Ques 1 Ans:** In the plots, we can see that the data hit rate gets maximize for a certain block size, but after that it decreases linearly. This is because, smaller blocks do not take maximum advantage of spatial locality. But if blocks are too large, there will be fewer blocks available, and more potential misses due to conflicts.

**Ques 2 Ans:** Optimal block size for,

1. spice.trace (considering instructions references) = 512B

spice.trace (considering data references) = 32B

1. cc. trace (considering instructions references) = 2048B

cc. trace (considering data references) = 32B

1. tex.trace (considering instructions references) = 32B

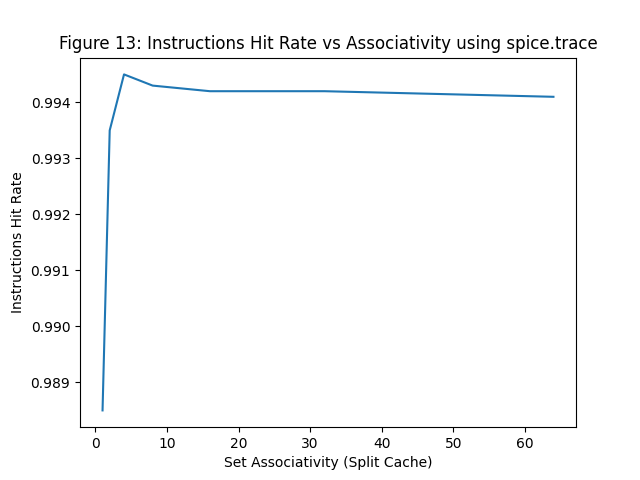
tex.trace (considering data references) = 128B

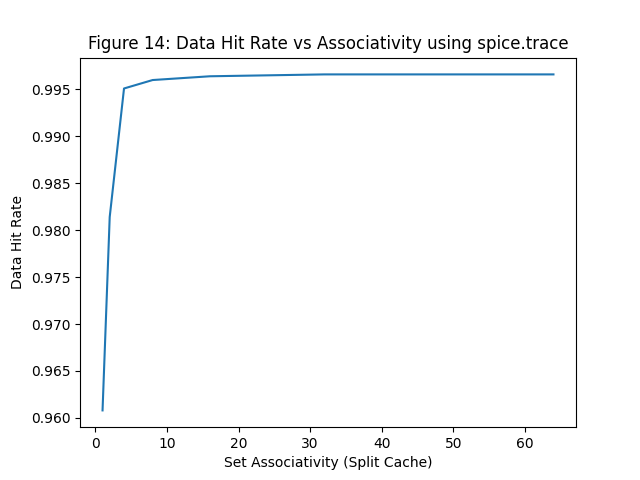
**Ques 3 Ans:** For spice.trace and cc.trace the optimal block size for instructions references are greater than the data references. So, we need larger block size for instruction references than data references in order to achieve less AMAT (average memory access time).

**2.3 Impact of Associativity**

2.3.1 Characterization from spice.trace:

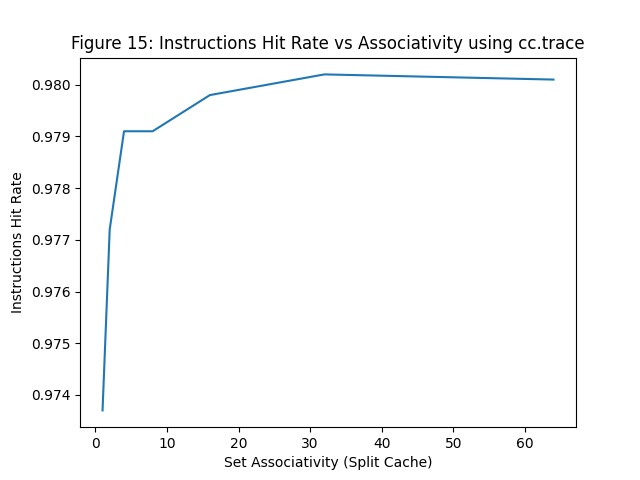
Setting up the Split Cache (Instruction Cache Size: 8 kB, Data Cache Size: 8kB), Write-back, Write-allocate and cache Block Size 128 Bytes ->

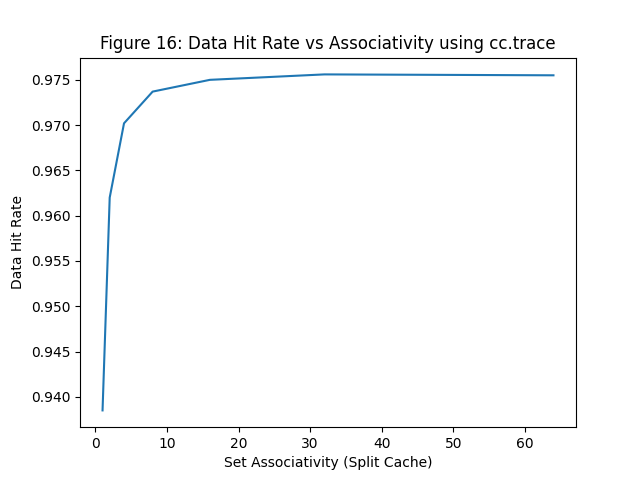




2.3.2 Characterization from cc.trace:

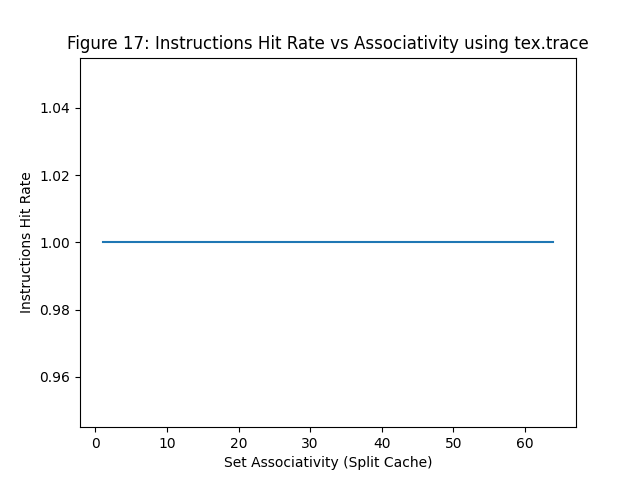
Setting up the Split Cache (Instruction Cache Size: 8 kB, Data Cache Size: 8kB), Write-back, Write-allocate and cache Block Size 128 Bytes ->

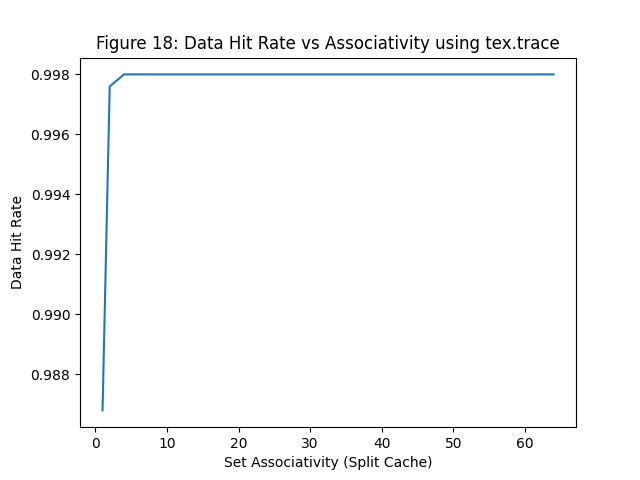




2.3.2 Characterization from tex.trace:

Setting up the Split Cache (Instruction Cache Size: 8 kB, Data Cache Size: 8kB), Write-back, Write-allocate and cache Block Size 128 Bytes ->





**Ques 1 Ans:** The shapes from the plots tell us that higher associativity exhibits lower miss rate. Because it implies lower conflicts between caches. So, that is why hit rate generally increases with set associativity.

**Ques 2 Ans:** Both the instruction and data references hit rate increases with the Block size. But the cache hit rate for data references are larger than their corresponding instruction references (from spice.trace and cc.trace).

**2.4 Memory Bandwidth**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Write-back | | Write-through | |
| Demand Fetch | Copies Back | Demand Fetch | Copies Back |
| i. CS: 8192, BS: 64, Assoc: 2 | 151104 | 13624 | 151104 | 72322 |
| ii. CS: 16384, BS: 64B, Assoc: 4 | 38064 | 7681 | 38064 | 71067 |
| iii. CS: 16384, BS: 128B, Assoc: 4 | 64320 | 9668 | 64320 | 71086 |
| iv. CS: 8192, BS: 128B, Assoc: 2 | 280768 | 32287 | 280768 | 72777 |
| v. CS: 16384, BS: 128B, Assoc: 2 | 93632 | 9880 | 93632 | 71490 |

**Ques 1 Ans:** Write-back has the smaller memory traffic. Because write-back is a storage method in which data is written into the cache every time a change occurs, but is written into the corresponding location in main memory only at specified intervals or under certain conditions

**Ques 2 Ans:** The scenario might change for write-allocate caches. Because, Write-back and write-allocate caches go well together, similarly write-through and write-no-allocate caches are also well-implemented.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Write-allocate | | | Write-no-allocate | |  |
| Demand Fetch | Copies Back | Total Memory Bandwidth | Demand Fetch | Copies Back | Total Memory Bandwidth |
| i. CS: 8192, BS: 64, Assoc: 2 | 156000 | 18880 | 174880 | 151104 | 13624 | 164728 |
| ii. CS: 16384, BS: 64B, Assoc: 4 | 39728 | 5280 | 45008 | 38064 | 7681 | 45745 |
| iii. CS: 16384, BS: 128B, Assoc: 4 | 66304 | 7200 | 73504 | 64320 | 9668 | 73988 |
| iv. CS: 8192, BS: 128B, Assoc: 2 | 291584 | 36256 | 327840 | 280768 | 32287 | 313055 |
| v. CS: 16384, BS: 128B, Assoc: 2 | 99008 | 9088 | 108096 | 93632 | 9880 | 103512 |

**Ques 1 Ans:** In the 5 different cases shown above, write-no-allocate has the smaller memory traffic. Because in write-no-allocate policy, data at the missed-write location is not loaded to cache, and is written directly to the backing store. In this approach, data is loaded into the cache on read misses only.

**Ques 2 Ans:** The scenario might change for write through policy. Because, Write-back and write-allocate caches go well together, similarly write-through and write-no-allocate caches are also well-implemented.