**ECE – 563 MICROPROCESSOR ARCHITECTURE**

**PROJECT 1B REPORT**

**Memory Hierarchy Simulator with two level of caches L1 and L2**

SUBMITTED BY,

VIGNESH KUMAR RAMARAJ

200310417

**Section 9.1.1**

1. **Cache Size Vs Miss Rate**
2. **gcc\_trace -**  Block size is fixed at 32 Bytes
3. **perl\_trace –** Block size is fixed at 32 Bytes
4. **go\_trace –** Block size is fixed at 64 Bytes
5. **vortex\_trace –** Block size is fixed at 64 Bytes
6. **Associativity Vs Miss Rate**

Here I have made the cache\_block\_size = 32B, cache\_capacity = 65536B

1. **L2 Cache Size Vs Miss Rate**

Here I have kept the L1 parameters, L1\_cache\_block\_size = 64, L1\_cache\_capacity = 1024, L1\_Associativity = 2 and L2\_Associativity = 4

1. **N vs Miss Rate**

Constant parameters, L1\_block\_size = 32, L1\_cache\_capacity = 8192, L1\_Associativity = 1, L2\_Associativity = 1, P = 8, L2\_capacity = 65536

1. **P Vs Miss Rate**

Constant parameters, L1\_block\_size = 32, L1\_cache\_capacity = 8192, L1\_Associativity = 1, L2\_Associativity = 1, N = 4, L2\_capacity = 65536

**Section 9.1.2**

**Average Access Time Calculation:**

Implement the formulas for AAT given in the project website with L2 cache and without L2 cache

**Best L1 Cache Design**

First start with catching the best L1 design. Below is the set of runs performed with a fixed L2 configuration (L2\_size = 65536, L2\_associativity = 1, P = 8, N = 1). Highlighted (Green) are the configurations for L1 that resulted in minimal miss rate for L1 cache and also considerably lesser Average Access Time. These runs are taken with gcc\_trace.txt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Block Size** | **L1 Capacity** | **L1 Associativity** | **L1 AAT** | **L1 Miss Rate** | **L2 AAT** |
|  |  |  |  |  |  |
| 16 | 1024 | 64 | 4.788 | 0.1417 | 1.1577 |
| 16 | 262144 | 4 | 2.29 | 0.0471 | 1.6 |
| 32 | 1024 | 8 | 3.36 | 0.1363 | 1.6 |
| 32 | 1024 | 16 | 3.56 | 0.1362 | 1.76 |
| 32 | 4096 | 8 | 1.64 | 0.0536 | 1.29 |
| 32 | 8192 | 8 | 1.3686 | 0.0395 | 1.248 |
| 32 | 16384 | 8 | 1.1599 | 0.0277 | 1.244 |
| 32 | 32768 | 8 | 1.2076 | 0.0262 | 1.15 |
| 32 | 32768 | 16 | 1.4072 | 0.0256 | 1.18 |
| 64 | 4096 | 8 | 1.8876 | 0.0599 | 1.6 |
| 64 | 4096 | 16 | 2.0519 | 0.0583 | 1.96 |
| 64 | 8192 | 8 | 1.2845 | 0.0316 | 1.0861 |
| 64 | 8192 | 16 | 1.4174 | 0.0286 | 1.114 |
| 64 | 8192 | 4 | 1.338 | 0.0386 | 1.0147 |
| 64 | 16384 | 2 | 1.0502 | 0.0183 | 0.9338 |
| 64 | 16384 | 4 | 0.9781 | 0.026 | 0.944 |
| 64 | 16384 | 8 | 1.0589 | 0.02 | 1.03 |
| 64 | 32768 | 2 | 0.9591 | 0.0183 | 0.9652 |
| 64 | 32768 | 4 | 0.9491 | 0.0156 | 0.9777 |
| 64 | 32768 | 8 | 1.0363 | 0.015 | 0.9878 |
| 128 | 32768 | 4 | 0.9791 | 0.011 | 0.958 |
| 128 | 32768 | 8 | 1.0355 | 0.0095 | 1.0463 |
| 128 | 32678 | 16 | 1.2293 | 0.0093 | 1.2241 |
| 128 | 65536 | 8 | 1.1689 | 0.0086 | 1.0463 |
| 128 | 16384 | 8 | 1.1904 | 0.0193 | 1.0478 |
| 128 | 16384 | 16 | 103047 | 0.0157 | 1.2281 |

Table 1

The above filtered configurations are simulated with all other benchmarks,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column1** | **L1 Block Size** | **L1 capacity** | **Assiciativity** | **L1 AAT** | **Miss Rate** | **L2 AAT - gcc\_trace** |
| 1 | 64 | 8192 | 4 | 1.338 | 0.0386 | 1.0147 |
| 2 | 64 | 8192 | 8 | 1.2845 | 0.0316 | 1.0861 |
| 3 | 64 | 16384 | 2 | 1.0502 | 0.0183 | 0.9338 |
| 4 | 64 | 16384 | 4 | 0.9781 | 0.026 | 0.944 |
| 5 | 64 | 16384 | 8 | 1.0589 | 0.02 | 1.03 |
| 6 | 64 | 32768 | 2 | 0.9591 | 0.0183 | 0.9652 |
| 7 | 64 | 32768 | 4 | 0.9491 | 0.0156 | 0.9777 |
| 8 | 64 | 32768 | 8 | 1.0363 | 0.015 | 0.9878 |
| 9 | 128 | 32768 | 4 | 0.9791 | 0.011 | 0.958 |
| 10 | 128 | 16384 | 8 | 1.1904 | 0.0193 | 1.0478 |

Table 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Column1** | **L2 AAT - perl\_trace** | **L2 AAT - go\_trace** | **L2 AAT - vortex trace** |
| 1 | 0.7837 | 1.1677 | 0.7991 |
| 2 | 0.8674 | 1.2677 | 0.8832 |
| 3 | 0.744 | 1.157 | 0.7521 |
| 4 | 0.7602 | 1.2067 | 0.7835 |
| 5 | 0.7902 | 1.2206 | 0.7914 |
| 6 | 0.7954 | 1.2255 | 0.8008 |
| 7 | 0.8217 | 1.2847 | 0.8372 |
| 8 | 0.9206 | 1.3847 | 0.9337 |
| 9 | 0.8391 | 1.0779 | 0.8886 |
| 10 | 0.8828 | 1.0998 | 0.9344 |

Table 3

From Table 2 and 3, we get the optimal configurations for L1 cache for each benchmarks

|  |  |  |  |
| --- | --- | --- | --- |
| **Column1** | **BLOCK\_SIZE** | **CAPACITY** | **ASSOCIATIVITY** |
| **GCC\_TRACE** | 64 | 16384 | 2 |
| **PERL\_TRACE** | 64 | 16384 | 2 |
| **GO\_TRACE** | 128 | 32768 | 4 |
| **VORTEX\_TRACE** | 64 | 16384 | 2 |

Table 4

Now that we found the best design of L1 cache for each benchmark, let us jump towards the design aspect of L2 cache.

Simulated multiple configurations for L2 to determine the nearly best capacities that would result in a low miss rate and AAT. I have used the L1 configurations for each trace from the results we obtained above.

GCC\_TRACE, PERL\_TRACE, VORTEX\_TRACE

L1\_capacity = 16384 L1\_block\_size = 64 L1\_associativity = 2

|  |  |  |  |
| --- | --- | --- | --- |
| **L2 Capacity** | **L2 AAT - gcc\_trace** | **L2 AAT - perl\_trace** | **L2 AAT - vortex\_trace** |
| 16384 | 1.0456 | 0.86952 | 0.80456 |
| 32768 | 0.9876 | 0.8079 | 0.770044 |
| 65536 | 0.9212 | 0.7418 | 0.75204 |
| 131072 | 0.9119 | 0.72681 | 0.74177 |
| 262144 | 0.9018 | 0.735096 | 0.739056 |

Table 5

GO\_TRACE

L1\_capacity = 32768 L1\_block\_size = 128 L1\_associativity = 4

|  |  |
| --- | --- |
| **L2 Capacity** | **L2 AAT - go\_trace** |
| 32768 | 1.0907 |
| 65536 | 1.0778 |
| 131072 | 1.0019 |
| 262144 | 1.0108 |

Table 6

By the above experiment, we have obtained L2 capacities that yield lower AAT for each trace. I have taken three sizes 65536B, 131072B, 262144B for consideration with associativity.

Now let us consider these capacities and simulate L2 caches for them with different associativity.

Note: For different set associativity for L2 cache, I had assumed P = 1 and N = 1.

With L2 capacity = 262144 Bytes and respective best design of L1 cache, each trace yielded the following results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L2\_Associativity** | **L2 AAT gcc\_trace** | **L2 AAT vortex\_trace** | **L2 AAT perl\_trace** | **L2 AAT go\_trace** |
| 1 | 0.90075 | 0.7379 | 0.7351 | 1.0108 |
| 2 | 0.89962 | 0.7312 | 0.7355 | 1.0109 |
| 4 | 0.90026 | 0.7319 | 0.7364 | 1.0116 |
| 8 | 0.90028 | 0.7334 | 0.7382 | 1.0131 |
| 16 | 0.90801 | 0.7362 | 0.7417 | 1.016 |
| 32 | 0.9184 | 0.742 | 0.7489 | 1.0212 |
| 64 | 0.93922 | 0.7534 | 0.7532 | 1.0296 |
| 128 | 0.98088 | 0.7763 | 0.7776 | 1.0348 |

Table 7

With L2 capacity = 131072 Bytes and respective best design of L1 cache, each trace yielded the following results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L2\_Associativity** | **L2 AAT gcc\_trace** | **L2 AAT vortex\_trace** | **L2 AAT perl\_trace** | **L2 AAT go\_trace** |
| 1 | 0.9108 | 0.7338 | 0.7268 | 1.0019 |
| 2 | 0.8851 | 0.7238 | 0.7268 | 1.0126 |
| 4 | 0.884 | 0.723 | 0.7252 | 1.0025 |
| 8 | 0.8866 | 0.7244 | 0.727 | 1.004 |
| 16 | 0.8918 | 0.7273 | 0.7306 | 1.0069 |
| 32 | 0.9022 | 0.733 | 0.7377 | 1.0127 |
| 64 | 0.9134 | 0.7412 | 0.752 | 1.024 |
| 128 | 0.9276 | 0.7671 | 0.7805 | 1.0475 |

Table 8

With L2 capacity = 65536 Bytes and respective best design of L1 cache, each trace yielded the following results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L2\_Associativity** | **L2 AAT gcc\_trace** | **L2 AAT vortex\_trace** | **L2 AAT perl\_trace** | **L2 AAT go\_trace** |
| 1 | 0.9117 | 0.7395 | 0.7392 | 1.0771 |
| 2 | 0.9042 | 0.7255 | 0.7232 | 1.0263 |
| 4 | 0.8974 | 0.7192 | 0.7212 | 1.0393 |
| 8 | 0.8817 | 0.72 | 0.7215 | 1.0547 |
| 16 | 0.8861 | 0.7228 | 0.725 | 1.066 |
| 32 | 0.8961 | 0.7285 | 0.7321 | 1.0814 |
| 64 | 0.9022 | 0.7341 | 0.7464 | 1.0922 |
| 128 | 0.9122 | 0.7447 | 0.7749 | 1.0976 |

Table 9

Now we have analyzed the L1 and L2 cache designs to build a best possible memory hierarchy that hold the constraints for Area Budget (L1 size + L2 size < 512 KB).

Further, simulated some of the aspects for the decoupled sector cache (P and N) and tabulated the results

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | N | L2 AAT - gcc trace |  |  |  | L2 AAT vortex\_trace |  |  | L2 AAT - perl\_trace |  |  | L2 AAT - go\_trace |  |
| Capacity |  | **65536** | **131072** | **262144** | **65536** | **131072** | **262144** | **65536** | **131072** | **262144** | **65536** | **131072** | **262144** |
| 1 | 1 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 1 | 2 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 1 | 4 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 1 | 8 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 1 | 16 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 1 | 32 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 2 | 1 | 0.9212 | 0.9159 | 0.9018 | 0.743 | 0.7369 | 0.7397 | 0.741 | 0.7272 | 0.735 | 1.0781 | 1.0019 | 1.0108 |
| 2 | 2 | 0.9166 | 0.9119 | 0.9007 | 0.7421 | 0.7356 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 2 | 4 | 0.9128 | 0.9109 | 0.9007 | 0.741 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 2 | 8 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 2 | 16 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 2 | 32 | 0.9117 | 0.9109 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0778 | 1.0019 | 1.0108 |
| 4 | 1 | 0.9313 | 0.924 | 0.9031 | 0.7511 | 0.7413 | 0.7423 | 0.7476 | 0.729 | 0.735 | 1.0781 | 1.0019 | 1.0108 |
| 4 | 2 | 0.9249 | 0.9133 | 0.904 | 0.7463 | 0.7379 | 0.7379 | 0.7406 | 0.7268 | 0.735 | 1.0781 | 1.0019 | 1.0108 |
| 4 | 4 | 0.9142 | 0.9113 | 0.9011 | 0.7434 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0781 | 1.0019 | 1.0108 |
| 4 | 8 | 0.9122 | 0.9113 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0781 | 1.0019 | 1.0108 |
| 4 | 16 | 0.9122 | 0.9113 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0781 | 1.0019 | 1.0108 |
| 4 | 32 | 0.9122 | 0.9113 | 0.9007 | 0.7395 | 0.7338 | 0.7379 | 0.7397 | 0.7268 | 0.735 | 1.0781 | 1.0019 | 1.0108 |

Table 10

The results from the above table are in accordance with the graph plotted at section 9.1.1 with varying N and P over constant L2 conditions. As we could see varying the value P, resulted in slight decrease of performance that can be matched with splitting the tag array i.e. increase N.

**Observations:**

Decouple sector caches are efficient for caches with large capacities.

**Section 9.1.3**

Note: I had discussed the (a), (b) and (c) part of this section parallely

From the results obtained from section 9.1.2, we can build a best memory hierarchy for each trace files given.

**a. Best Memory Hierarchy Design:**

**GCC\_TRACE**

L1\_CACHE\_BLOCK\_SIZE = 64 Bytes, L1\_CACHE\_CAPACITY = 16384 Bytes, L1\_CACHE\_ASSOCIATIVITY = 2, L2\_CACHE\_CAPACITY = 65536, L2\_CACHE\_ASSOCIATIVITY = 2

Average Access Time : 0.8817 ns

b. Though L2 cache size = 262144 yielded a better miss rate, due to the advantage of associativity between the sets, we can achieve better AAT with a smaller L2 capacity of 65536 Bytes with respect to this case. Associativity here is 1, which implies most of the misses for this trace are compulsory and capacity misses.

**In case of decouple sectored**, the best result would be,

L2\_CACHE\_CAPACITY = 131072, L2\_Associativity = 1, P = 4, N = 32

Average Access Time: 0.9007 ns

The advantage here is that the number of bits for the tag array is reduced. This can save the number of bit comparisons. But we are paying the penalty of ~200ps in access time.

**a. PERL\_TRACE**

L1\_CACHE\_BLOCK\_SIZE = 64 Bytes, L1\_CACHE\_CAPACITY = 16384 Bytes, L1\_CACHE\_ASSOCIATIVITY = 2, L2\_CACHE\_CAPACITY = 65536, L2\_CACHE\_ASSOCIATIVITY = 4

Average Access Time : 0.7212 ns

b. Though L2 cache size = 0.7212 yielded a better miss rate, due to the advantage of associativity between the sets, we can achieve better AAT with a smaller L2 capacity of 65536 Bytes with respect to this case. We have got almost equal access time with the L2 associativity of 8 in this case.

**In case of decouple sectored**, the best result would be,

L2\_CACHE\_CAPACITY = 131072, L2\_Associativity = 1, P = 4, N = 32

Average Access Time: 0.7268 ns

Here, with the decoupled sector cache, we get better results when the size of L2 cache is increased. This also reduces the miss rate for the cache. The advantage here is that the number of bits for the tag array is reduced. This can save the number of bit comparisons. But we are paying the penalty of ~150 ps in access time.

**a. Best Memory Hierarchy:**

**VORTEX\_TRACE**

L1\_CACHE\_BLOCK\_SIZE = 64 Bytes, L1\_CACHE\_CAPACITY = 16384 Bytes, L1\_CACHE\_ASSOCIATIVITY = 2, L2\_CACHE\_CAPACITY = 65536, L2\_CACHE\_ASSOCIATIVITY = 4

Average Access Time : 0.7192 ns

b. We achieved better AAT with a smaller L2 capacity of 65536 Bytes, that when combined with the L1 size is well below the area budget provided. This case needed higher associativity compared to the perl\_trace, which implies that this case would have had more conflict misses with associativity of 2.

**In case of decouple sectored**, the best result would be,

L2\_CACHE\_CAPACITY = 131072, L2\_Associativity = 1, P = 4, N = 32

Average Access Time: 0.7338 ns

The advantage here is that the number of bits for the tag array is reduced. This can save the number of bit comparisons. But we are paying the penalty of ~150 ps in access time.

**a. Best Memory Hierarchy:**

**GO\_TRACE**

L1\_CACHE\_BLOCK\_SIZE = 128 Bytes, L1\_CACHE\_CAPACITY = 32768 Bytes, L1\_CACHE\_ASSOCIATIVITY = 4, L2\_CACHE\_CAPACITY = 131072, L2\_CACHE\_ASSOCIATIVITY = 4

Average Access Time : 1.0025 ns

b. Though L2 cache with associativity yielded a better AAT, due to the advantage of associativity between the sets, we can achieve better miss rate with associativity of 4.This is the advantage of associativity, yields a lower miss rate along with the cache size. Here associativity of 8 yields the same result, this is because of the large cache size, where associativity saturates.

**In case of decouple sectored**, the best result would be,

L2\_CACHE\_CAPACITY = 131072, L2\_Associativity = 1, P = 4, N = 32

Average Access Time: 1.0019 ns

As we see from the table above, we see that for this trace, the decoupling of the cache into sectors does not affect the performance much. This can save the number of bit comparisons in the tag matching checks.

**Section 9.1.4**

Comparison of Different Benchmarks

With the simulation results from section 9.1.2, we can see that the benchmarks, gcc\_trace, perl\_trace and vortex\_trace behave similar with each other for different cache configurations. Best memory hierarchies for these cases are similar. This make us conclude, because of the concept of locality of reference we can design a cache that can work mostly efficiently for different applications and programs. In my observations, go trace needed a different L1 configuration to perform better. But from table 3, we can accept that it did not behave badly with the other configuration (64,16384,2) though.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column1** | **BLOCK\_SIZE** | **L1\_CAPACITY** | **ASSOCIATIVITY** | **L2\_CAPACITY** | **L2\_ASSOCIATIVITY** |
| **GCC\_TRACE** | 64 | 16384 | 2 | 65536 | 2 |
| **PERL\_TRACE** | 64 | 16384 | 2 | 65536 | 4 |
| **GO\_TRACE** | 128 | 32768 | 4 | 131072 | 1 |
| **VORTEX\_TRACE** | 64 | 16384 | 2 | 65536 | 4 |

**Section 9.1.5**

**1. Advantages of Decoupled sectored cache**

* In a sectored cache, a single address tag is associated with a sector consisting on several cache lines, while valid, dirty are associated with each of the inner cache lines. Thus we can combine low tag array size with small block sizes
* But from our observations, the miss ratio is significantly higher than the miss ratio on a non-sectored cache.
* In the decoupled sectored cache cases, the address tag location associated with a cache line location is chosen among several possible locations. The tag volume on a decoupled sectored cache is in the same range as the tag volume in a traditional sectored cache; but the hit ratio on a decoupled sectored cache is very close to the hit ratio on a non-sectored cache.
* A decoupled sectored cache will allow the same level of performance as a non-sectored cache, but at a significantly lower hardware cost due to the reduced memory traffic is moving huge chunks of tag values.

**2. Observations from the Simulator**

* From the table 10, we observe that the implementation of decoupled sector cache model at L2 cache level with high capacity has given us similar results compared to the normal cache at L2. The results were tabulated with associativity = 1. We can still achieve better results with increased associativity with the decoupling of sectors. We can achieve performance (miss rate and AAT) comparable to that of normal L2 cache in cases where the size of the L2 cache is larger. But for smaller caches, it is not efficient to use decoupled sector provided the complexity.

**3. Reason for implementing Decoupled caches**

* As described above, we have the advantages of decoupled sector caches when the cache capacity is higher. So correlating table 10 and table 1, 2 we can say that L1 cache performance degrades with large sizes. Since L1 has to be accessed by every memory operation, degradation in L1 performance would be destructive to the whole memory hierarchy. This make us opt for decoupled sector cache in the lower hierarchy (L2) not in L1.