# Flexible Cache – Performance Analysis

## 1.Effects of changing various parameters on miss rates

#### L1 cache size vs miss rate

Keeping constant block size = 16bytes, file – gcc\_trace.txt

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cache Size** | **log(CacheSize)** | **Assoc 1(geometric mean)** | **Assoc 2(geometric mean)** | **Assoc 4(geometric mean)** | **Assoc 8(geometric mean)** | **Assoc 16(geometric mean)** |
| 128 | 7 | 0.440322687 | 0.384016666 | 0.360265425 | 0.320460867 |  |
| 256 | 8 | 0.343506754 | 0.295211742 | 0.258967238 | 0.244707707 | 0.23958585 |
| 512 | 9 | 0.274747027 | 0.214831184 | 0.190169186 | 0.183656543 | 0.181405605 |
| 1024 | 10 | 0.208089131 | 0.155156748 | 0.135965735 | 0.127657052 | 0.124257626 |
| 2048 | 11 | 0.136317895 | 0.103007881 | 0.087739305 | 0.082907104 | 0.0751653 |
| 4096 | 12 | 0.096911328 | 0.073533261 | 0.060526542 | 0.057764125 | 0.056607402 |
| 8192 | 13 | 0.072120575 | 0.052851251 | 0.048394576 | 0.047213724 | 0.046651486 |
| 16384 | 14 | 0.060664405 | 0.044076945 | 0.041586674 | 0.041229021 | 0.041047319 |
| 32768 | 15 | 0.049737599 | 0.041705101 | 0.040546152 | 0.040510721 | 0.040499677 |
| 65536 | 16 | 0.043341306 | 0.038879463 | 0.038642451 | 0.038452323 | 0.038551036 |
| 131072 | 17 | 0.041052528 | 0.038013909 | 0.03786931 | 0.03786931 | 0.03786931 |
| 262144 | 18 | 0.038156576 | 0.037889394 | 0.03786931 | 0.03786931 | 0.03786931 |
| 524288 | 19 | 0.037932183 | 0.03786931 | 0.03786931 | 0.03786931 | 0.03786931 |

#### Associativity vs miss rate

Keeping constant cache block size = 16384bytes and block size = 16.

Therefore, for full associativity associativity will be 1024

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Associativity** | **Miss Rate(gcc\_trace.txt)** | **Miss Rate (perl\_trace.txt)** | **Miss Rate (go\_trace.txt)** | **Miss Rate (vortex\_trace.txt)** | **Miss Rate(average)** |
| 1 | 0.0672 | 0.0536 | 0.1033 | 0.0364 | 0.0606644 |
| 2 | 0.0525 | 0.0297 | 0.0984 | 0.0246 | 0.0440769 |
| 4 | 0.0482 | 0.0273 | 0.0984 | 0.0231 | 0.0415867 |
| 8 | 0.0477 | 0.027 | 0.0984 | 0.0228 | 0.041229 |
| 16 | 0.0476 | 0.0267 | 0.0984 | 0.0227 | 0.0410473 |
| 32 | 0.0476 | 0.0266 | 0.0984 | 0.0225 | 0.0409182 |
| 64 | 0.0475 | 0.0265 | 0.0984 | 0.0225 | 0.0408582 |
| 128 | 0.0475 | 0.0265 | 0.0984 | 0.0225 | 0.0408582 |
| 256 | 0.0475 | 0.0264 | 0.0984 | 0.0225 | 0.0408196 |
| 512 | 0.0475 | 0.0265 | 0.0984 | 0.0225 | 0.0408582 |
| 1024 | 0.0475 | 0.0265 | 0.0984 | 0.0225 | 0.0408582 |

#### L2 cache size vs miss rate

Setting L1 cache size = 16KB

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cache Size** | **log(CacheSize)** | **Assoc 1(geometric mean)** | **Assoc 2(geometric mean)** | **Assoc 4(geometric mean)** | **Assoc 8(geometric mean)** | **Assoc 16(geometric mean)** |
| 128 | 7 | 1 | 0.999825 | 0.99925 | 0.99965 |  |
| 256 | 8 | 1 | 0.99975 | 0.998899 | 0.998975 | 0.997947 |
| 512 | 9 | 1 | 0.999325 | 0.997872 | 0.997822 | 0.997822 |
| 1024 | 10 | 0.999925 | 0.99895 | 0.997547 | 0.997148 | 0.997072 |
| 2048 | 11 | 0.99935 | 0.997874 | 0.996122 | 0.995521 | 0.995396 |
| 4096 | 12 | 0.997346 | 0.995619 | 0.992539 | 0.991233 | 0.990629 |
| 8192 | 13 | 0.970253 | 0.986003 | 0.978532 | 0.976178 | 0.973874 |
| 16384 | 14 | 0.847332 | 0.878599 | 0.898639 | 0.902777 | 0.907955 |
| 32768 | 15 | 0.713702 | 0.701201 | 0.651606 | 0.639295 | 0.64417 |
| 65536 | 16 | 0.655483 | 0.612349 | 0.604314 | 0.6027 | 0.604645 |
| 131072 | 17 | 0.603744 | 0.604968 | 0.595951 | 0.595951 | 0.595951 |
| 262144 | 18 | 0.597432 | 0.59626 | 0.595951 | 0.595951 | 0.595951 |
| 507904 | ~19 | 0.596476 | 0.595951 | 0.595951 | 0.595951 | 0.595951 |

#### N vs miss rate

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **CacheSize256 GM miss rate** | **CacheSize65536 GM miss rate** | **CacheSize507904 GM miss rate** |
| 1 | 0.998799572 | 0.602699732 | 0.534637439 |
| 2 | 0.998799572 | 0.684104081 | 0.587437821 |
| 4 | 0.998799572 | 0.772823649 | 0.606757118 |
| 8 | 0.998799572 | 0.810663773 | 0.608395823 |
| 16 | 0.998799572 | 0.863977826 | 0.609130239 |
| 32 | 0.998799572 | 0.915064306 | 0.609130239 |

#### P vs miss rate

|  |  |  |  |
| --- | --- | --- | --- |
| **P** | **CacheSize256 GM miss rate** | **CacheSize65536 GM miss rate** | **CacheSize507904 GM miss rate** |
| 1 | 0.997346 | 0.655483 | 0.534637 |
| 2 | 0.999375 | 0.677565 | 0.534706 |
| 4 | 0.9999 | 0.696386 | 0.534797 |
| 8 | 1 | 0.711787 | 0.535024 |
| 16 | 1 | 0.731186 | 0.535523 |
| 32 | 1 | 0.752715 | 0.535975 |
| 64 | 1 | 0.786717 | 0.536394 |
| 128 | 1 | 0.822882 | 0.539247 |
| 256 | 1 | 0.863981 | 0.542722 |
| 512 |  | 0.928583 | 0.542788 |
| 1024 |  | 0.973329 | 0.547567 |
| 2048 |  | 0.985447 | 0.571491 |
| 4096 |  | 0.999149 | 0.629597 |
| 8192 |  |  | 0.612546 |
| 16384 |  |  | 0.970978 |

## 2. Design space and noteworthy trends

Block size vs AAT while keeping L2 size constant (for gcc\_trace)

|  |  |  |  |
| --- | --- | --- | --- |
| BlockSize | AAT(L2 size - 49152) | AAT(L2Size- 114688) | AAT(L2 Size 507904) |
| 8 | 1.6974 | 1.7224 | 1.8927 |
| 16 | 1.148 | 1.1606 | 1.2591 |
| 32 | 0.8735 | 0.8793 | 0.9428 |
| 64 | 0.7683 | 0.7706 | 0.8194 |
| 128 | 0.8119 | 0.8135 | 0.867 |
| 256 | 1.0286 | 1.0141 | 1.086 |
| 512 | 1.5271 | 1.4605 | 1.5507 |
| 1024 | 2.6581 | 2.4606 | 2.5965 |
| 2048 | 5.9249 | 4.6805 | 4.8777 |

We can understand that there is a point(~64Bytes) after which increasing the block size is detrimental to AAT.

L1 size + Associativity vs AAT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| L1 size | Assoc 1 | Assoc2 | Assoc4 | Assoc8 | Assoc 16 |
| 128 | 2.2147 | 2.1324 | 2.1163 | 2.2025 |  |
| 256 | 1.8952 | 1.8475 | 1.8323 | 1.9157 | 2.102 |
| 512 | 1.7084 | 1.6183 | 1.6139 | 1.7042 | 1.893 |
| 1024 | 1.529 | 1.4423 | 1.4465 | 1.5212 | 1.7168 |
| 2048 | 1.3945 | 1.3037 | 1.3131 | 1.3783 | 1.5681 |
| 4096 | 1.2711 | 1.2088 | 1.2144 | 1.3004 | 1.4957 |
| 8192 | 1.192 | 1.1408 | 1.1702 | 1.2663 | 1.461 |
| 16384 | 1.1918 | 1.1545 | 1.1814 | 1.2774 | 1.4767 |
| 32768 | 1.2989 | 1.3002 | 1.3406 | 1.446 | 1.6536 |
| 65536 | 1.4799 | 1.6152 | 1.6906 | 1.8512 | 2.0721 |
| 131072 | 1.8507 | 1.9694 | 2.1061 | 2.2093 | 2.4093 |
| 262144 | 2.6423 | 2.6709 | 2.7341 | 2.8343 | 3.0343 |
| 524288 | 3.8923 | 3.9343 | 3.9843 | 4.0843 | 4.2843 |

Similar to the observations for block size we see that the both associativity and L1 cachesize have a goldilocks area at around associativity 2 and cache size 8192

L2 size vs AAT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| L2 size | Assoc 1 | Assoc2 | Assoc4 | Assoc8 | Assoc 16 |
| 128 | 1.6529 | 1.6541 | 1.6564 | 1.6612 |  |
| 256 | 1.6529 | 1.6541 | 1.6565 | 1.6612 | 1.6708 |
| 512 | 1.653 | 1.6541 | 1.6565 | 1.6613 | 1.6705 |
| 1024 | 1.6531 | 1.6543 | 1.6565 | 1.6608 | 1.6701 |
| 2048 | 1.6533 | 1.6544 | 1.6557 | 1.6604 | 1.6696 |
| 4096 | 1.6527 | 1.6548 | 1.6559 | 1.6602 | 1.6697 |
| 8192 | 1.6477 | 1.6532 | 1.6564 | 1.6606 | 1.6699 |
| 16384 | 1.5404 | 1.5986 | 1.6215 | 1.6389 | 1.651 |
| 32768 | 1.4129 | 1.4261 | 1.3313 | 1.2975 | 1.2849 |
| 65536 | 1.3926 | 1.3107 | 1.2756 | 1.2774 | 1.2866 |
| 131072 | 1.2954 | 1.322 | 1.2865 | 1.2913 | 1.3009 |
| 262144 | 1.3148 | 1.3146 | 1.3164 | 1.3211 | 1.3307 |
| 524288 | 1.3744 | 1.3736 | 1.376 | 1.3808 | 1.3903 |

Similar to earlier observations we are able to see that we get the best AAT for L2 size ~= 64MB and L2 assoc =4

## 3. Best memory hierarchy configuration

Best memory hierarchy is assumed to be the configuration with the lowest access time. I will not be considering parameters such as size of the tag store as a factor contributing to “best”. ie. The best memory hierarchy regardless of price.

The simulation was run over a number of different configuration while keeping the values of L2 data\_blocks=1 and L2 address\_tags=1. This was done as an increase in miss rate was observed when the values of N and P are increased.

Therefore the best access time for gcc\_trace.txt was observed to be 0.7671ns at block size 64 L1\_size=16KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1

Therefore the best access time for perl\_trace.txt was observed to be 0.6720ns at block size 64 , L1\_size=16KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1(0.6718 was observed for 48KB)

Therefore the best access time for go\_trace.txt was observed to be 0.7354ns at block size 128 ,L1\_size=16KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1(0.7348 was observed for 120KB size)

Therefore the best access time for vortex\_trace.txt was observed to be 0.6534ns at block size 64 , L1\_size=8KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1

So in total this configuration seems to be best - block size 64, L1\_size=16KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1

The reason for this config being the best is that the tradeoff has been dome most effectively between the increase in access times due to a larger and more complex cache and the reduced miss rate which can be obtained from a larger and more complex(in terms of increased associativity) cache. Block size of 64bytes gives us the the right balance between spatial locality and effects of cache pollution.

## 4. Benchmark comparison

The best access time for gcc\_trace.txt was observed to be 0.7671ns at block size 64 L1\_size=16KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1

The best access time for perl\_trace.txt was observed to be 0.6720ns at block size 64 , L1\_size=16KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1(0.6718 was observed for 48KB)

The best access time for go\_trace.txt was observed to be 0.7354ns at block size 128 ,L1\_size=16KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1(0.7348 was observed for 120KB size)

The best access time for vortex\_trace.txt was observed to be 0.6534ns at block size 64 , L1\_size=8KB, L1\_assoc=2, L2\_size=64KB, L2\_assoc=4 , N=1, P=1

We are able to see that the go\_trace gives better AAT for a bigger block size which might be due to it using bigger variable sizes by default or the higher spatial locality in the dataset.

## 5. Advantages of decoupled sectored cache

* The size of the tag store can be reduced if we use a sectored cache
* If we decouple the sectored cache we can decrease the miss rate to close to the levels shown by a set associative cache.
* To obtain a tag size comparable to that of a decoupled sectored cache in the scenario of a classical cache the block size would have to be increased. This would result in greater hit latency and seen in this perspective we can claim that the decoupled sectored cache helps reduce hit latency.

We implemented decoupled sectored cache in this project to explore the tradeoffs we would have to make to leverage a lower tag store size. We were able to understand that miss rate increases when we use a decoupled sectored cache.

## Appendix

Data used for compiling this report can be found here as I was not able to include it in the report due to space constraints

<https://github.com/GeoBK/cache_simulator/blob/sectored-decoupled-cache/Microarch-Proj%201B.xlsx>