# Part-I

For the first part of the project, i.e. the part that requires us to match the output provided at T-Square 100%, please refer to the attached "`results.txt`" file. I have uploaded it separately and included it in the "`Pritam_Bhattacharyya_Final_code_and_Solutions.tar.gz`" file, under the "`Solution`" folder. For quick reference, I have also written my output at the end of the report in **Appendix A**.

# Part-II

During the experiments phase, I have approached the problem of finding an optimal cache by checking the trends displayed by the cache for various values of [C, B, S, V, K]. The results of the findings are noted in the subsequent sections along with my conclusions.

I have optimized the cache for every trace file, individually. Apart from low AAT, I have also subjected my cache to the limitation of occupying a size of a maximum of 48 KB.

The cache size is calculated from the following expression:

**Cache Size = Size of Main cache + Size of Victim Cache**

Now,

**Main Cache Size** = Size of Tag store + Size of data store + number of valid bits + number of prefetch bits + number of dirty bits

**Victim Cache Size** = Size of Victim Cache store + Size of Victim data store + number of victim valid bits + number of victim prefetch bits + number of victim dirty bits

In terms of cache parameters [C, B, S, V, K], upon simplification the formula for cache size becomes:

$$\boldsymbol{Cache\ Size} = [\mathbf{2}^{\boldsymbol{C-B}} \times (\mathbf{64} - \boldsymbol{C} + \boldsymbol{S})] + [\mathbf{3} \times \mathbf{2}^{\boldsymbol{C-B}}] + [\mathbf{8} \times \mathbf{2}^{\boldsymbol{C}}] + [\mathbf{8} \times \boldsymbol{V} \times \mathbf{2}^{\boldsymbol{B}}] + [\boldsymbol{V} \times (\mathbf{64} - \boldsymbol{B})] + [\mathbf{3} \times \boldsymbol{V}]$$

The cache size calculated by the above formula is in bits, which is then divided by 8 to get the equivalent bytes of data and then, I made sure that the optimized design for each implementation is below 48 KB.

Also, please note that, I have restricted my B from [0, 6], V from [0, 4] and K from [0, 4] as mentioned in the project requirement.

Please refer to the subsequent sections for the detailed experiment report.

For "**mcf.trace**", the trends are as follows:

**Variation of AAT with various values of 'C' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| C = 8 | 53.652196 |
| C = 11 | 21.203112 |
| C = 13 | 6.248611 |
| C = 15 | 6.248611 |
| C = 17 | 6.248611 |
| C = 19 | 6.248611 |

[**Note:** for this simulation, B=5, S=3, V=4, K=2]

As seen from the trend, that as the value of C increases, AAT decreases, however, after a certain point, AAT stagnates. This is mainly due to the compulsory misses, which cannot be avoided no matter how large the cache size is.

**Variation of AAT with various values of 'B' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| B=0 | 12.597242 |
| B=1 | 12.594091 |
| B=2 | 12.423912 |
| B=3 | 11.748316 |
| B=4 | 9.203506 |
| B=5 | 6.248611 |
| B=6 | 4.767422 |

[**Note:** for this simulation, C=15, S=3, V=4, K=2]

As seen from the trend, AAT decreases steadily with increase in values of B. This occurs due to the fact that more data is brought in the cache with every read (or prefetch) operation, which effectively improves AAT.

**Variation of AAT with various values of 'S' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| S=0 | 7.096317 |
| S=1 | 5.904944 |
| S=2 | 6.048611 |
| S=3 | 6.248611 |
| S=4 | 6.448611 |
| S=6 | 6.648611 |
| S=8 | 7.248611 |
| S=10 | 7.648611 |

[**Note:** For this simulation, C=15, B=5, V=4, K=2]

As seen from the trend, AAT first decreases, then starts to increase again with increase in value of S. This is because, initially, AAT reduces due to the increase in storage space in the main cache. However, after a certain point, the increase in time for accessing and comparing the different tag values in a set, overpowers the benefit of hit time reduction due to increase in storage space. Due to this, the hit time first reduces, and then increases.

**Variation of AAT with various values of 'V' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| V=0 | 6.248611 |
| V=1 | 6.248611 |
| V=2 | 6.248611 |
| V=3 | 6.248611 |
| V=4 | 6.248611 |

[**Note:** For this simulation, C=15, B=5, S=3, K=2]

The above trend clearly shows that in the presence of prefetch in a set-associative cache, for "`mcf.trace`", AAT is independent of variation in V. However, it is a possibility that this trend is a special case and AAT might actually vary with a different combination of [C, B, S, K].

**Variation of AAT with various values of 'K' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| K=0 | 6.249005 |
| K=1 | 6.248611 |
| K=2 | 6.248611 |
| K=3 | 6.248217 |
| K=4 | 6.248217 |

[**Note:** For this simulation, C=15, B=5, S=3, V=4]

As seen from the trend, the improvement in AAT is marginal with increase in K for "`mcf.trace`" for the mentioned values of [C, B, S, V]. This is due to the initial prefetching of a few useful addresses, thereby reducing AAT. However, as the degree of prefetch is increased, useful blocks in the cache are evicted out to make place for incoming prefetched blocks, which are not actually accessed during the course of the trace.

Now, I have considered a few boundary conditions for checking the trends for "mcf.trace":

**Variation of AAT with various values of 'V' (for no prefetch structure with C=15, B=5, S=3 and K = 0)**

| | AAT |
|---|---|
| V=0 | 6.249005 |
| V=1 | 6.249005 |
| V=2 | 6.249005 |
| V=3 | 6.249005 |
| V=4 | 6.249005 |

As seen from the trend, even in this configuration, the AAT for "mcf.trace" is independent of the value of 'V'.

**Variation of AAT with various values of 'K' (for no victim cache structure with C=15, B=5, S=3 and V = 0)**

| | AAT |
|---|---|
| K=0 | 6.249005 |
| K=1 | 6.248611 |
| K=2 | 6.248611 |
| K=3 | 6.248217 |
| K=4 | 6.248217 |

Again, the trend shows that AAT of "mcf.trace" does not vary significantly with changes in the value of 'K'. In fact, it is seen that the value actually increases when K>2. This is due to possible eviction of useful blocks of data in order to make room for the incoming prefetched data.

**Variation of AAT with various values of 'V' (for direct mapped cache structure with C=15, B=5, S=0 and K = 0)**

| | AAT |
|---|---|
| V=0 | 7.994485 |
| V=1 | 7.320071 |
| V=2 | 7.253102 |
| V=3 | 7.218042 |
| V=4 | 7.106559 |

Now, the victim cache benefits are apparent as AAT decreases significantly with increase in the value of 'V'. This is because in a direct mapped implementation, the victim cache is heavily taxed and hence with a larger victim cache, AAT reduces.

Based on the trends discussed in this section, it can be safely said that for "mcf.trace", the following operations needs to be performed in order to obtain an optimized cache structure:

Increase C.

Increase B strongly.

Do not make S too big, keep it somewhere around 1-2 so that the optimal AAT is obtained.

Victim cache does not really matter – implement it if cost budget is not an issue and if one can fit it in the total cache size limit.

Prefetch degree 'K' should limited to 1-2, so that we can get the optimal solution.

Keeping the above considerations in mind, and the fact that the cache structure cannot exceed 48 KB, the optimal cache parameters that I have found for "mcf.trace" are:

**[C, B, S, V, K] = [15, 6, 2, 4, 1]**

The resultant output for this structure is as follows:

Cache Statistics

Accesses: 507700
Reads: 280182
Read misses: 797
Read misses combined: 787
Writes: 227518
Write misses: 4716
Write misses combined: 4715
Misses: 5513
Writebacks: 4888
Victim cache misses: 5502
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 664960
Hit Time: 2.400000
Miss Penalty: 200
Miss rate: 0.010859
Average access time (AAT): 4.567422

Size = 36511 Bytes (approx.)

The size of the cache is calculated using the formula given in the beginning and approximated to the next integral value.

For "**perlbench.trace**", the trends are as follows:

**Variation of AAT with various values of 'C' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| C = 8 | 60.28513 |
| C = 11 | 28.491483 |
| C = 13 | 15.230828 |
| C = 15 | 10.502397 |
| C = 17 | 4.060662 |
| C = 19 | 3.796592 |

[**Note:** for this simulation, B=5, S=3, V=4, K=2]

As seen from the trend, an increase the value of C, results in a decrease in AAT.

**Variation of AAT with various values of 'B' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| B=0 | 14.706629 |
| B=1 | 14.152476 |
| B=2 | 13.932155 |
| B=3 | 14.870983 |
| B=4 | 13.979845 |
| B=5 | 10.502397 |
| B=6 | 8.103694 |

[**Note:** for this simulation, C=15, S=3, V=4, K=2]

As seen from the trend, AAT decreases with increase in values of B. This occurs because we are bringing in more data with every read (or prefetch) operation, which effectively improves our hit time.

**Variation of AAT with various values of 'S' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| S=0 | 12.0457 |
| S=1 | 10.511114 |
| S=2 | 10.345751 |
| S=3 | 10.502397 |
| S=4 | 10.70082 |
| S=6 | 11.108703 |
| S=8 | 11.532745 |
| S=10 | 11.937475 |

[**Note:** For this simulation, C=15, B=5, V=4, K=2]

As we can see, the AAT first decreases, then starts to increase again with increase in value of S. This is because, initially, the AAT reduces due to the increase in storage space in the main cache. However, after a certain point, the increase in time for accessing and comparing the different tag values in a set, overpowers the benefit of hit time reduction due to increase in storage space. Due to this, the hit time first reduces, then, increases.

**Variation of AAT with various values of 'V' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| V=0 | 10.517768 |
| V=1 | 10.509885 |
| V=2 | 10.505155 |
| V=3 | 10.503185 |
| V=4 | 10.502397 |

[**Note:** For this simulation, C=15, B=5, S=3, K=2]

The above trend clearly shows us that in the presence of prefetch in a set-associative cache, for "`perlbench.trace`", the AAT decreases, though not significantly.

**Variation of AAT with various values of 'K' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| K=0 | 11.264653 |
| K=1 | 10.789721 |
| K=2 | 10.502397 |
| K=3 | 10.304147 |
| K=4 | 10.158317 |

[**Note:** For this simulation, C=15, B=5, S=3, V=4]

As we can see, the AAT decreases with increase in K for "`perlbench.trace`" for the mentioned values of [C, B, S, V]. This is due to the prefetching of a few useful addresses, thereby reducing the AAT.

Now, I have considered a few boundary conditions for checking the trends for "perlbench.trace":

**Variation of AAT with various values of 'V' (for no prefetch structure with C=15, B=5, S=3 and K = 0)**

| | AAT |
|---|---|
| V=0 | 11.272141 |
| V=1 | 11.267806 |
| V=2 | 11.267412 |
| V=3 | 11.265047 |
| V=4 | 11.264653 |

As we can see, even in this configuration, the AAT for "perltrace.trace" is not heavily depended on the value of 'V'.

**Variation of AAT with various values of 'K' (for no victim cache structure with C=15, B=5, S=3 and V = 0)**

| | AAT |
|---|---|
| K=0 | 11.272141 |
| K=1 | 10.800756 |
| K=2 | 10.517768 |
| K=3 | 10.321095 |
| K=4 | 10.175659 |

Again, we can see that as we increase the degree of prefetch, AAT of the structure decreases.

**Variation of AAT with various values of 'V' (for direct mapped cache structure with C=15, B=5, S=0 and K = 0)**

| | AAT |
|---|---|
| V=0 | 15.781701 |
| V=1 | 13.448819 |
| V=2 | 13.085821 |
| V=3 | 12.899395 |
| V=4 | 12.729129 |

As we can see, now, the victim cache benefits are more apparent as the AAT decreases significantly with increase in the value of 'V'. This is because in a direct mapped implementation, the victim cache is heavily taxed and hence with a larger victim cache, the AAT reduces.

Based on the trends discussed in this section, we can safely say that for "`perlbench.trace`", we can perform the following operations to obtain an optimized cache structure:

Increase C.

Increase B strongly.

Make S not too big, keep it somewhere around 2-3 so that we can get the optimal AAT.

Victim cache implementation is preferable. However, its size will depend on the cost and total cache size budget.

Prefetch degree 'K' can be increased provided we do not bother about the amount of bits transferred between primary memory and cache.

Keeping the above considerations in mind, and the fact that the cache structure cannot exceed 48 KB, the optimal cache parameters that I have found for "`perlbench.trace`" are:

[C, B, S, V, K] = [15, 6, 3, 4, 4]

The resultant output for this structure is as follows:

Cache Statistics

Accesses: 507441
Reads: 302052
Read misses: 10386
Read misses combined: 10358
Writes: 205389
Write misses: 3177
Write misses combined: 3153
Misses: 13563
Writebacks: 6150
Victim cache misses: 13511
Prefetched blocks: 2184
Useful prefetches: 1404
Bytes transferred to/from memory: 1398080
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.026728
Average access time (AAT): 7.925151

Size = 36575 Bytes (approx.)

For "**bzip2.trace**", the trends are as follows:

**Variation of AAT with various values of 'C' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| C = 8 | 17.794834 |
| C = 11 | 3.834495 |
| C = 13 | 3.809886 |
| C = 15 | 3.022395 |
| C = 17 | 3.022395 |
| C = 19 | 3.022395 |

[**Note:** for this simulation, B=5, S=3, V=4, K=2]

As we can see from the trend, that as we increase the value of C, the AAT decreases, however, after a certain point, the value of AAT stagnates. This is mainly due to the compulsory misses, which cannot be avoided no matter how large the cache size is.

**Variation of AAT with various values of 'B' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| B=0 | 10.808421 |
| B=1 | 7.148643 |
| B=2 | 5.315082 |
| B=3 | 4.268644 |
| B=4 | 3.435975 |
| B=5 | 3.022395 |
| B=6 | 2.815972 |

[**Note:** for this simulation, C=15, S=3, V=4, K=2]

As we can see, the AAT decreases with increase in values of B. This occurs because we are bringing in more data with every read (or prefetch) operation, which effectively improves our hit time.

**Variation of AAT with various values of 'S' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| S=0 | 2.957918 |
| S=1 | 2.739196 |
| S=2 | 2.830108 |
| S=3 | 3.022395 |
| S=4 | 3.222395 |
| S=6 | 3.622395 |
| S=8 | 4.022395 |
| S=10 | 4.422395 |

[**Note:** For this simulation, C=15, B=5, V=4, K=2]

As we can see, AAT first decreases, then starts to increase again with increase in value of S. This is because, initially, AAT reduces due to the increase in storage space in the main cache. However, after a certain point, the increase in time for accessing and comparing the different tag values in a set, overpowers the benefit of hit time reduction due to increase in storage space. Due to this, the hit time first reduces, then, increases.

**Variation of AAT with various values of 'V' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| V=0 | 3.022395 |
| V=1 | 3.022395 |
| V=2 | 3.022395 |
| V=3 | 3.022395 |
| V=4 | 3.022395 |

[**Note:** For this simulation, C=15, B=5, S=3, K=2]

The above trend clearly shows us that in the presence of prefetch in a set-associative cache, for "`bzip2.trace`", AAT is independent of variation in V. However, it is a possibility that this trend is a special case and AAT might actually vary with a different combination of [C, B, S, K].

**Variation of AAT with various values of 'K' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| K=0 | 3.212656 |
| K=1 | 3.093651 |
| K=2 | 3.022395 |
| K=3 | 2.972442 |
| K=4 | 2.940854 |

[**Note:** For this simulation, C=15, B=5, S=3, V=4]

As we can see, AAT decreases with increase in K for "`bzip2.trace`" for the mentioned values of [C, B, S, V]. This is due to the prefetching of a few useful addresses, thereby reducing AAT.

Now, I have considered a few boundary conditions for checking the trends for "bzip2.trace":

**Variation of AAT with various values of 'V' (for no prefetch structure with C=15, B=5, S=3 and K = 0)**

| | AAT |
|---|---|
| V=0 | 3.212656 |
| V=1 | 3.212656 |
| V=2 | 3.212656 |
| V=3 | 3.212656 |
| V=4 | 3.212656 |

As we can see, even in this configuration, the AAT for "bzip2.trace" is independent of the value of 'V'.

**Variation of AAT with various values of 'K' (for no victim cache structure with C=15, B=5, S=3 and V = 0)**

| | AAT |
|---|---|
| K=0 | 3.212656 |
| K=1 | 3.093651 |
| K=2 | 3.022395 |
| K=3 | 2.972442 |
| K=4 | 2.940854 |

Again, we can see that as we increase the degree of prefetch, AAT of the structure decreases.

**Variation of AAT with various values of 'V' (for direct mapped cache structure with C=15, B=5, S=0 and K = 0)**

| | AAT |
|---|---|
| V=0 | 4.928116 |
| V=1 | 4.027863 |
| V=2 | 3.609876 |
| V=3 | 3.290692 |
| V=4 | 3.03799 |

As we can see, now, the victim cache benefits are more apparent as AAT decreases significantly with increase in the value of 'V'. This is because in a direct mapped implementation, the victim cache is heavily taxed and hence with a larger victim cache, AAT reduces.

Based on the trends discussed in this section, we can safely say that for "bzip2.trace", we can perform the following operations to obtain an optimized cache structure:

Increase C.

Increase B strongly.

Make S not too big, keep it somewhere around 1-2 so that we can get the optimal AAT.

Victim cache does not really matter – implement it if cost budget is not an issue and if one can fit it in the total cache size limit.

Increase prefetch degree 'K'.

Keeping the above considerations in mind, and the fact that the cache structure cannot exceed 48 KB, the optimal cache parameters that I have found for "bzip2.trace" are:

**[C, B, S, V, K] = [15, 6, 1, 4, 4]**

The resultant output for this structure is as follows:

Cache Statistics

Accesses: 544514
Reads: 369344
Read misses: 431
Read misses combined: 340
Writes: 175170
Write misses: 336
Write misses combined: 292
Misses: 767
Writebacks: 261
Victim cache misses: 632
Prefetched blocks: 376
Useful prefetches: 364
Bytes transferred to/from memory: 81216
Hit Time: 2.200000
Miss Penalty: 200
Miss rate: 0.001409
Average access time (AAT): 2.432134

Size = 36447 Bytes (approx.)

For "**astar.trace**", the trends are as follows:

**Variation of AAT with various values of 'C' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| C = 8 | 47.071831 |
| C = 11 | 21.772908 |
| C = 13 | 17.499854 |
| C = 15 | 16.853352 |
| C = 17 | 16.205654 |
| C = 19 | 14.056763 |

[**Note:** for this simulation, B=5, S=3, V=4, K=2]

As we can see from the trend, that as we increase the value of C, the AAT decreases.

**Variation of AAT with various values of 'B' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| B=0 | 90.829757 |
| B=1 | 73.386969 |
| B=2 | 50.61423 |
| B=3 | 37.939842 |
| B=4 | 29.406097 |
| B=5 | 16.853352 |
| B=6 | 10.482058 |

[**Note:** for this simulation, C=15, S=3, V=4, K=2]

As we can see, the AAT decreases with increase in values of B. This occurs because we are bringing in more data with every read (or prefetch) operation, which effectively improves our hit time.

**Variation of AAT with various values of 'S' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| S=0 | 17.027479 |
| S=1 | 16.524344 |
| S=2 | 16.666115 |
| S=3 | 16.853352 |
| S=4 | 17.043381 |
| S=6 | 17.439792 |
| S=8 | 17.777575 |
| S=10 | 18.168402 |

[**Note:** For this simulation, C=15, B=5, V=4, K=2]

As we can see, AAT first decreases, then starts to increase again with increase in value of S. This is because, initially, AAT reduces due to the increase in storage space in the main cache. However, after a certain point, the increase in time for accessing and comparing the different tag values in a set, overpowers the benefit of hit time reduction due to increase in storage space. Due to this, the hit time first reduces, then, increases.

**Variation of AAT with various values of 'V' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| V=0 | 16.854947 |
| V=1 | 16.854947 |
| V=2 | 16.854549 |
| V=3 | 16.85415 |
| V=4 | 16.853352 |

[**Note:** For this simulation, C=15, B=5, S=3, K=2]

The above trend clearly shows us that in the presence of prefetch in a set-associative cache, for "`astar.trace`", AAT is does not change significantly with variation in V. However, it is a possibility that this trend is a special case and AAT might actually vary with a different combination of [C, B, S, K].

**Variation of AAT with various values of 'K' (keeping the other factors constant at their default values)**

| | AAT |
|---|---|
| K=0 | 22.526296 |
| K=1 | 19.011416 |
| K=2 | 16.853352 |
| K=3 | 15.387655 |
| K=4 | 14.398559 |

[**Note:** For this simulation, C=15, B=5, S=3, V=4]

As we can see, AAT decreases with increase in K for "`astar.trace`" for the mentioned values of [C, B, S, V]. This is due to the prefetching of a few useful addresses, thereby reducing AAT.

Now, I have considered a few boundary conditions for checking the trends for "bzip2.trace":

**Variation of AAT with various values of 'V' (for no prefetch structure with C=15, B=5, S=3 and K = 0)**

| | AAT |
|---|---|
| V=0 | 22.527892 |
| V=1 | 22.527892 |
| V=2 | 22.527493 |
| V=3 | 22.527094 |
| V=4 | 22.526296 |

As we can see, even in this configuration, the AAT for "astar.trace" is not heavily dependent on the value of 'V'.

**Variation of AAT with various values of 'K' (for no victim cache structure with C=15, B=5, S=3 and V = 0)**

| | AAT |
|---|---|
| K=0 | 22.527892 |
| K=1 | 19.013011 |
| K=2 | 16.854947 |
| K=3 | 15.392043 |
| K=4 | 14.404143 |

Again, we can see that as we increase the degree of prefetch, AAT of the structure decreases.

**Variation of AAT with various values of 'V' (for direct mapped cache structure with C=15, B=5, S=0 and K = 0)**

| | AAT |
|---|---|
| V=0 | 23.387606 |
| V=1 | 22.971228 |
| V=2 | 22.840413 |
| V=3 | 22.761444 |
| V=4 | 22.700424 |

As we can see, even in the case of a direct mapped structure, the victim cache benefits, though better than other cases, are not very significant as AAT does not decreases significantly with increase in the value of 'V'. The marginal improvement in the sensitivity to victim cache is because in a direct mapped implementation, the victim cache is heavily taxed and hence with a larger victim cache, AAT reduces.

Based on the trends discussed in this section, we can safely say that for "astar.trace", we can perform the following operations to obtain an optimized cache structure:

Increase C.

Increase B strongly.

Make S not too big, keep it somewhere around 1-2 so that we can get the optimal AAT.

Victim cache does not really matter – implement it if cost budget is not an issue and if one can fit it in the total cache size limit.

Increase prefetch degree 'K'.

Keeping the above considerations in mind, and the fact that the cache structure cannot exceed 48 KB, the optimal cache parameters that I have found for "astar.trace" are:

**[C, B, S, V, K] = [15, 6, 1, 4, 4]**

The resultant output for this structure is as follows:

Cache Statistics

Accesses: 501468
Reads: 289766
Read misses: 6594
Read misses combined: 6577
Writes: 211702
Write misses: 10105
Write misses combined: 10103
Misses: 16699
Writebacks: 19688
Victim cache misses: 16680
Prefetched blocks: 10460
Useful prefetches: 10292
Bytes transferred to/from memory: 2996992
Hit Time: 2.400000
Miss Penalty: 200
Miss rate: 0.033300
Average access time (AAT): 9.052468

Size = 36511 Bytes (approx.)

# Part-III

As far as the question goes regarding the choice between “Victim cache” (Option A) and “Prefetcher” (Option B), I would have to say that it depends on the overall structure of the cache. As seen in the trends, in almost all the traces, then the benefit of having a victim cache (provided the initial assumptions) is practically negligible (if the cache is not direct mapped).

Therefore, it is more advantageous to implement a “Prefetcher”, given the restriction of implementing either one between the two options and not both (Option B).

However, in case of a direct mapped cache, the victim cache option is more lucrative, because, in almost all the traces, the AAT improves considerably upon the implementation of a victim cache, if the main cache has a direct mapped structure.

# Appendix – A

The output of the "`./run_script.sh`" is as follows:

```
**************************
***Default Configuration ***
**************************

--- astar.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 4
K: 2

Cache Statistics
Accesses: 501468
Reads: 289766
Read misses: 10313
Read misses combined: 10309
Writes: 211702
Write misses: 25429
Write misses combined: 25429
Misses: 35742
Writebacks: 38341
Victim cache misses: 35738
Prefetched blocks: 14538
Useful prefetches: 14229
Bytes transferred to/from memory: 2835744
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.071275
Average access time (AAT): 16.853352

--- bzip2.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 4
```

K: 2

Cache Statistics
Accesses: 544514
Reads: 369344
Read misses: 591
Read misses combined: 591
Writes: 175170
Write misses: 559
Write misses combined: 559
Misses: 1150
Writebacks: 298
Victim cache misses: 1150
Prefetched blocks: 520
Useful prefetches: 518
Bytes transferred to/from memory: 62976
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.002112
Average access time (AAT): 3.022395

--- mcf.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 4
K: 2

Cache Statistics
Accesses: 507700
Reads: 280182
Read misses: 1542
Read misses combined: 1542
Writes: 227518
Write misses: 7720
Write misses combined: 7720
Misses: 9262
Writebacks: 8040
Victim cache misses: 9262
Prefetched blocks: 2

Useful prefetches: 1
Bytes transferred to/from memory: 553728
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.018243
Average access time (AAT): 6.248611

--- perlbench.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 4
K: 2

Cache Statistics
Accesses: 507441
Reads: 302052
Read misses: 13751
Read misses combined: 13733
Writes: 205389
Write misses: 6338
Write misses combined: 6317
Misses: 20089
Writebacks: 10207
Victim cache misses: 20050
Prefetched blocks: 2386
Useful prefetches: 1951
Bytes transferred to/from memory: 1044576
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.039589
Average access time (AAT): 10.502397

**********
***  C = 10, S = 0  ***
**********

--- astar.trace ---
Cache Settings
C: 10

B: 5
S: 0
V: 4
K: 2

Cache Statistics
Accesses: 501468
Reads: 289766
Read misses: 53428
Read misses combined: 45374
Writes: 211702
Write misses: 38639
Write misses combined: 35202
Misses: 92067
Writebacks: 53032
Victim cache misses: 80576
Prefetched blocks: 16446
Useful prefetches: 14833
Bytes transferred to/from memory: 4801728
Hit Time: 2.000000
Miss Penalty: 200
Miss rate: 0.183595
Average access time (AAT): 34.136049

--- bzip2.trace ---
Cache Settings
C: 10
B: 5
S: 0
V: 4
K: 2

Cache Statistics
Accesses: 544514
Reads: 369344
Read misses: 26546
Read misses combined: 10502
Writes: 175170
Write misses: 13567
Write misses combined: 4902
Misses: 40113

Writebacks: 5106
Victim cache misses: 15404
Prefetched blocks: 40
Useful prefetches: 6
Bytes transferred to/from memory: 657600
Hit Time: 2.000000
Miss Penalty: 200
Miss rate: 0.073668
Average access time (AAT): 7.657889

--- mcf.trace ---
Cache Settings
C: 10
B: 5
S: 0
V: 4
K: 2

Cache Statistics
Accesses: 507700
Reads: 280182
Read misses: 52977
Read misses combined: 43722
Writes: 227518
Write misses: 41897
Write misses combined: 39816
Misses: 94874
Writebacks: 50144
Victim cache misses: 83538
Prefetched blocks: 3946
Useful prefetches: 905
Bytes transferred to/from memory: 4404096
Hit Time: 2.000000
Miss Penalty: 200
Miss rate: 0.186870
Average access time (AAT): 34.908410

--- perlbench.trace ---
Cache Settings
C: 10
B: 5

S: 0
V: 4
K: 2

Cache Statistics
Accesses: 507441
Reads: 302052
Read misses: 96154
Read misses combined: 76518
Writes: 205389
Write misses: 39073
Write misses combined: 27071
Misses: 135227
Writebacks: 51013
Victim cache misses: 103589
Prefetched blocks: 4688
Useful prefetches: 3051
Bytes transferred to/from memory: 5097280
Hit Time: 2.000000
Miss Penalty: 200
Miss rate: 0.266488
Average access time (AAT): 42.827998

**********
***  V = 0, K = 0  ***
**********

--- astar.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 0
K: 0

Cache Statistics
Accesses: 501468
Reads: 289766
Read misses: 11145
Read misses combined: 11145
Writes: 211702

Write misses: 38821
Write misses combined: 38821
Misses: 49966
Writebacks: 38344
Victim cache misses: 49966
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 2825920
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.099639
Average access time (AAT): 22.527892

--- bzip2.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 0
K: 0

Cache Statistics
Accesses: 544514
Reads: 369344
Read misses: 851
Read misses combined: 851
Writes: 175170
Write misses: 817
Write misses combined: 817
Misses: 1668
Writebacks: 298
Victim cache misses: 1668
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 62912
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.003063
Average access time (AAT): 3.212656

--- mcf.trace ---

Cache Settings
C: 15
B: 5
S: 3
V: 0
K: 0

Cache Statistics
Accesses: 507700
Reads: 280182
Read misses: 1542
Read misses combined: 1542
Writes: 227518
Write misses: 7721
Write misses combined: 7721
Misses: 9263
Writebacks: 8044
Victim cache misses: 9263
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 553824
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.018245
Average access time (AAT): 6.249005

--- perlbench.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 0
K: 0

Cache Statistics
Accesses: 507441
Reads: 302052
Read misses: 14024
Read misses combined: 14024
Writes: 205389
Write misses: 7979

Write misses combined: 7979
Misses: 22003
Writebacks: 10213
Victim cache misses: 22003
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 1030912
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.043361
Average access time (AAT): 11.272141

**********
***  V = 0  ***
**********

--- astar.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 0
K: 2

Cache Statistics
Accesses: 501468
Reads: 289766
Read misses: 10313
Read misses combined: 10313
Writes: 211702
Write misses: 25429
Write misses combined: 25429
Misses: 35742
Writebacks: 38344
Victim cache misses: 35742
Prefetched blocks: 14538
Useful prefetches: 14229
Bytes transferred to/from memory: 2835968
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.071275

Average access time (AAT): 16.854947

--- bzip2.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 0
K: 2

Cache Statistics
Accesses: 544514
Reads: 369344
Read misses: 591
Read misses combined: 591
Writes: 175170
Write misses: 559
Write misses combined: 559
Misses: 1150
Writebacks: 298
Victim cache misses: 1150
Prefetched blocks: 520
Useful prefetches: 518
Bytes transferred to/from memory: 62976
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.002112
Average access time (AAT): 3.022395

--- mcf.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 0
K: 2

Cache Statistics
Accesses: 507700
Reads: 280182
Read misses: 1542

Read misses combined: 1542
Writes: 227518
Write misses: 7720
Write misses combined: 7720
Misses: 9262
Writebacks: 8044
Victim cache misses: 9262
Prefetched blocks: 2
Useful prefetches: 1
Bytes transferred to/from memory: 553856
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.018243
Average access time (AAT): 6.248611

--- perlbench.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 0
K: 2

Cache Statistics
Accesses: 507441
Reads: 302052
Read misses: 13751
Read misses combined: 13751
Writes: 205389
Write misses: 6338
Write misses combined: 6338
Misses: 20089
Writebacks: 10221
Victim cache misses: 20089
Prefetched blocks: 2386
Useful prefetches: 1930
Bytes transferred to/from memory: 1046272
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.039589
Average access time (AAT): 10.517768

**********
***  K = 0  ***
**********

--- astar.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 4
K: 0

Cache Statistics
Accesses: 501468
Reads: 289766
Read misses: 11145
Read misses combined: 11141
Writes: 211702
Write misses: 38821
Write misses combined: 38821
Misses: 49966
Writebacks: 38341
Victim cache misses: 49962
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 2825696
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.099639
Average access time (AAT): 22.526296

--- bzip2.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 4
K: 0

Cache Statistics

Accesses: 544514
Reads: 369344
Read misses: 851
Read misses combined: 851
Writes: 175170
Write misses: 817
Write misses combined: 817
Misses: 1668
Writebacks: 298
Victim cache misses: 1668
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 62912
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.003063
Average access time (AAT): 3.212656

--- mcf.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 4
K: 0

Cache Statistics
Accesses: 507700
Reads: 280182
Read misses: 1542
Read misses combined: 1542
Writes: 227518
Write misses: 7721
Write misses combined: 7721
Misses: 9263
Writebacks: 8040
Victim cache misses: 9263
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 553696
Hit Time: 2.600000

Miss Penalty: 200
Miss rate: 0.018245
Average access time (AAT): 6.249005

--- perlbench.trace ---
Cache Settings
C: 15
B: 5
S: 3
V: 4
K: 0

Cache Statistics
Accesses: 507441
Reads: 302052
Read misses: 14024
Read misses combined: 14016
Writes: 205389
Write misses: 7979
Write misses combined: 7968
Misses: 22003
Writebacks: 10198
Victim cache misses: 21984
Prefetched blocks: 0
Useful prefetches: 0
Bytes transferred to/from memory: 1029824
Hit Time: 2.600000
Miss Penalty: 200
Miss rate: 0.043361
Average access time (AAT): 11.264653