CS6600 2024

Indian Institute of Technology, Madras

Authors

Sai Ashwin R NS24Z344

Date

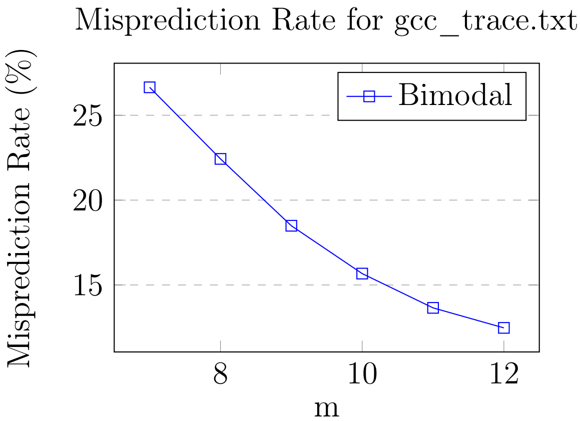
October 20th 2024

Contents

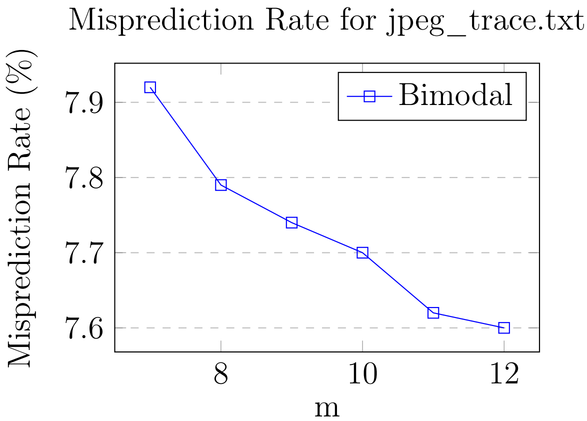
1 Bimodal predictor

1.1 Question 1A

The following plots show the misprediction rate vs "m" for the 2 trace files:



(a) gcc_trace.txt



(b) jpeg_trace.txt

Figure 1: Bimodal Predictor Misprediction Rate for different trace files

1.1.1 Trends in gcc_trace.txt

For the gcc benchmark, the misprediction rate decreases steadily as m increases from 7 to 12. The rate drops from 26.65% at $m = 7$ to 12.47% at $m = 12$. This trend indicates that increasing the number of low-order PC bits used to index the bimodal predictor significantly improves prediction accuracy for the gcc workload, particularly in the early range of m.

1.1.2 Trends in jpeg_trace.txt

For the jpeg benchmark, the misprediction rate shows only a slight improvement as m increases, starting at 7.92% for $m = 7$ and decreasing to 7.6% for $m = 12$. This smaller variation suggests that the jpeg trace is less sensitive to changes in the number of PC bits used for the bimodal predictor.

1.1.3 Similarities

Both benchmarks demonstrate a general decrease in misprediction rate as m increases, showing that using more PC bits for indexing improves the accuracy of the bimodal predictor. Additionally, the improvement in misprediction rate flattens out for higher values of m, suggesting diminishing returns as m approaches 12.

1.1.4 Differences

The gcc benchmark experiences a much larger reduction in misprediction rate (over 14% decrease from $m = 7$ to $m = 12$), while the jpeg benchmark shows only a minor reduction (around 0.3%). This indicates that the bimodal predictor performs much better with increasing m for the gcc workload, while the jpeg workload remains relatively unaffected by changes in m.

In addition, the gcc benchmark on average experiences a **much larger misprediction rate** than the jpeg benchmark with the maximum misprediction rate being 7.92 % for the jpeg trace vs the 26.65 % of the gcc trace. This suggests that the gcc benchmark has more **complicated control flow logic** than the jpeg benchmark. This can be reasoned easily by noting that gcc is a compiler, while the jpeg trace is probably that of a simple jpeg parser. Thus, the gcc benchmark might need a more sophisticated branch predictor.

1.2 Question 1B

We are given a maximum budget of 16 kB for the branch predictor. The amount of storage required when there are "m" bits of the PC used is $2^m * 2 = 2^{m+1}$ bits. Thus,

$$
\begin{aligned}
2^{m+1} &= 16 * 1024 * 8 \\
2^{m+1} &= 2^{17} \\
m &= 16
\end{aligned}
$$

For the **gcc_trace.txt** benchmark, the misprediction rate decreases as m increases from 7 to 12:

- At $m = 7$, the misprediction rate is 26.65%.
- At $m = 10$, it drops to 15.67%.
- At $m = 11$, it drops to 13.65%
- At $m = 12$, the misprediction rate further decreases to 12.47%.

It is clear that the reduction slows down as m increases beyond 10.

Therefore, for **gcc_trace.txt**, we can choose $m = 11$ as the optimal value, as the reduction in misprediction rate beyond this point is close to only 1%, whereas the storage required grows exponentially.

For the **jpeg_trace.txt** benchmark, the misprediction rate also decreases as m increases, but the reduction is less significant:

- At $m = 7$, the misprediction rate is 7.92%.
- At $m = 9$, it reduces to 7.74%.

- At $m = 10$, it reduces to 7.7%.
- By $m = 11$, it only decreases slightly to 7.6%.

Since the misprediction rate improvement is very small after $m = 9$, the best choice is $m = 9$.

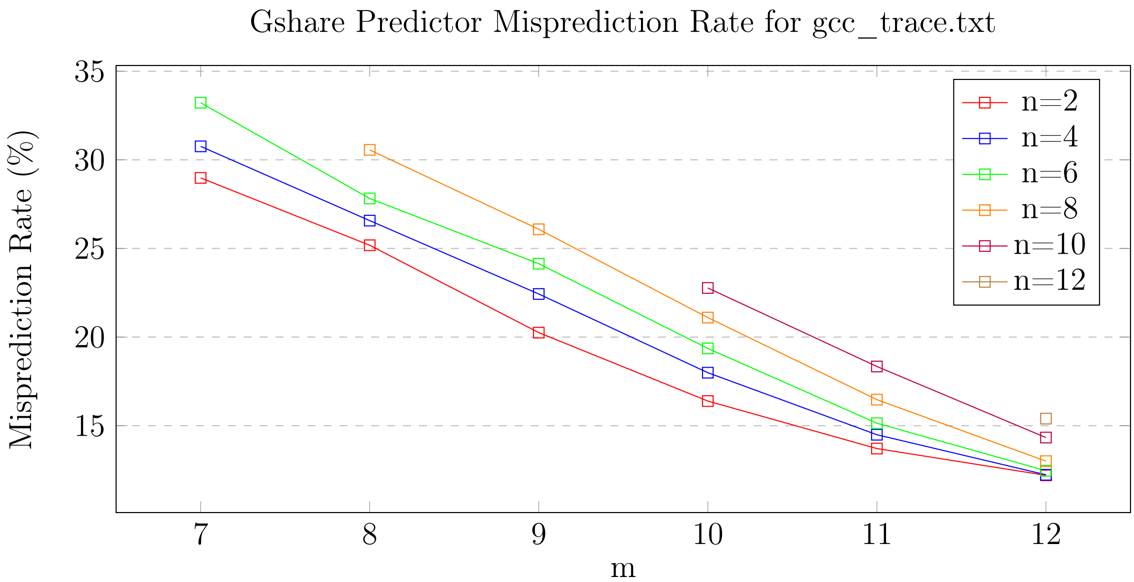
For both configurations, the optimal value chosen by us is less than the limit of m = 16.

In conclusion, the optimal design for **gcc_trace.txt is m = 11**, while the optimal design for **jpeg_trace.txt is m = 9**. Both designs minimize misprediction rates while staying within the 16 kB storage limit.

2 Gshare Predictor Results

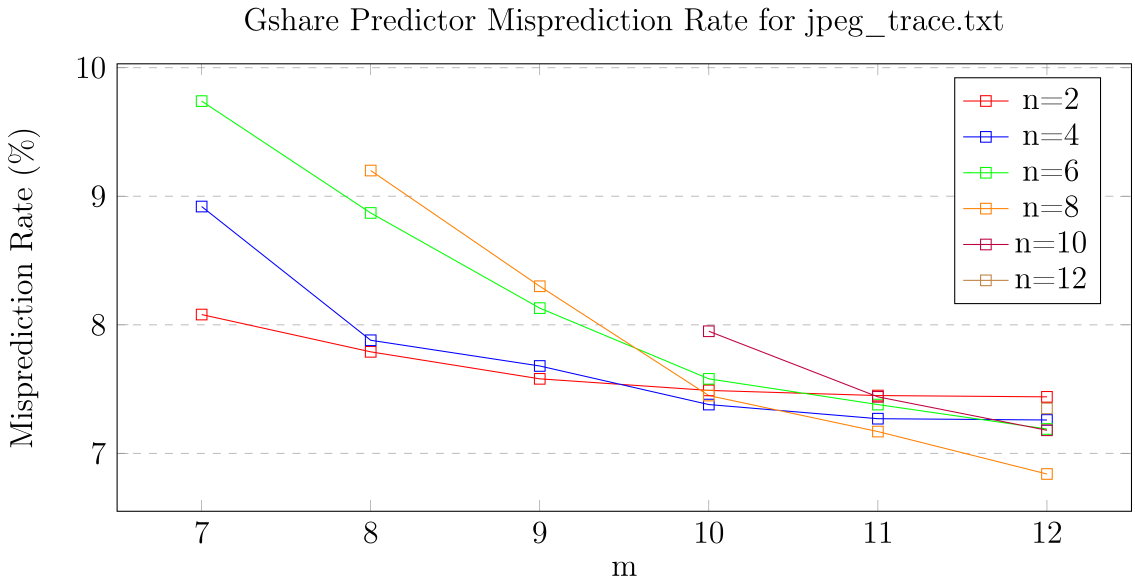
2.1 Question 2A

The folowing plots show the misprediction rate vs "m" for different values of "n" for the 2 trace files.



2.1.1 Trends in gcc_trace.txt

For the gcc benchmark, the misprediction rate decreases as the value of m increases from 7 to 12, across all values of n. This shows that increasing the number of PC bits used for indexing (m) improves the prediction accuracy. The impact of n (global history register size) is more pronounced for smaller m values. However, as m increases, the differences in misprediction rates between different values of n diminish, converging around $m = 12$.



2.1.2 Trends in jpeg_trace.txt

For the jpeg benchmark, the misprediction rate also decreases as m increases, but the decrease is more gradual compared to the gcc benchmark. The impact of n on the misprediction rate is less significant in this case, with a very small difference in rates across different values of n. This indicates that the jpeg workload is less sensitive to the global history register size and relies more on increasing m.

2.1.3 Similarities

Both benchmarks show a consistent decrease in misprediction rate as m increases. In addition, the misprediction rates tend to converge as m approaches 12, implying that at higher values of m, the impact of n becomes less important in both benchmarks.

2.1.4 Differences

The gcc benchmark exhibits higher misprediction rates overall (ranging from 12% to 35%) compared to the jpeg benchmark (6.8% to 10%). Furthermore, gcc is more sensitive to changes in n, particularly for smaller values of m. In contrast, the jpeg benchmark shows minimal sensitivity to n, indicating that increasing m is more impactful for improving prediction accuracy in the jpeg benchmark.

2.2 Question 2B

We are given a storage budget of 16 kB. We can write the number of bits required in terms of m and n:

$$\begin{aligned} \text{Storage} &= n + 2^m * 2 \\ &= n + 2^{m+1} \end{aligned}$$

For the **gcc_trace.txt** benchmark, the misprediction rate decreases significantly as m increases. For instance, for $n = 2$, the misprediction rate decreases from 28.98% at $m = 7$ to 12.2% at $m = 12$. However, as m increases beyond 10, the reduction in misprediction rate becomes less pronounced. Additionally, increasing n actually increases the misprediction rate, rather counter intuitively. This could be attributed to the **inertia** introduced by increasing the number of bits in the BHR. Therefore, for **gcc_trace.txt**, the optimal design is $\mathbf{m = 11}$ and $\mathbf{n = 2}$.

For the **jpeg_trace.txt** benchmark, the misprediction rate is less sensitive to increases in m and n. Most of the configurations seem to saturate at just above 7 % after $m = 10$. We see diminishing returns after $m = 10$, and the best performing configuration at $m = 10$ has $n = 4$. Increasing n has a negligible effect on storage, while storage increases exponentially with m.Therefore, for **jpeg_trace.txt**, the optimal design is $\mathbf{m = 10}$ and $\mathbf{n = 4}$.

In conclusion, the optimal design for **gcc_trace.txt** is $\mathbf{m = 11}$ and $\mathbf{n = 2}$, while the optimal design for **jpeg_trace.txt** is $\mathbf{m = 10}$ and $\mathbf{n = 4}$. Both configurations minimize misprediction rates while respecting the storage constraint of 16 kB.