Report on Second Compression Experiment

Diego Jurado

May 19, 2024

1 Introduction

The first compression experiment was devised to determine the best size in bits to represent an integer for our ORAM counting scheme given a specific cache line size. For the first experiment this was 128B, and the results stated an 8-bit counter was best. However, this meant on average we could perform roughly 15k accesses before the compression buffer grew too large, requiring a counter reset. This is an expensive operation. I hypothesized with more space we could easily get more accesses. So I devised this experiment testing both more memory and a way of exploiting the LZW compression algorithm. There are two motivations here. First, with more memory we can add more counters to the compressible array meaning we are representing more memory in one line, reducing the size of our logical memory tree, and second with a larger data type size we are able to increase the number of accesses before overflow. Both in the average and the worst case.

2 Methods

For this experiement, I modified the simulation from the first one. I would use 5 strategies on two different array compression sizes that corresponded to matching array size i.e., a 252B compression corresponds with 256 counters, and 124B compression corresponds with 128 counters. The 5 strategies are as follows, both a base 8-bit and 16-bit counter similar to the first experiment now being tested on both sizes now. The three new ones are similar. A 9-bit, 10-bit, and 16-bit counter split by the lower 8 bits, where the higher bits are placed together in the first half of the array-to-be-compressed and the lower bits are placed together in the second half of the array-to-be-compressed. The idea here is that the leading bits will remain empty until the lower 8-bit counters are filled, which will be the same as the base array, while having reserve bits to handle overflow if needed.

3 Results

The following results presented in Table 1 represent the average case for the given array of counters, which is random accessing and compressing. On the other hand, we have determined the worst case to be consecutive accesses to the same location translating to consecutive increments to the same counter, which is represented as the following equation: $2^n - 1$ where n is the number of bits used to represent the counter.

4 Discussion

We should not have to make a compromise between the worst case and the average case for accesses, so we had determined that a ratio between the average case and the worst case close to 1 would be best for our purposes. While we can prioritize one over the other, we should not. In the case of 128

	128 Items - 124B					256 Items - 252B				
	Baseline		Split			Baseline		Split		
	8b	16b	9b	10b	16b	8b	16b	9b	10b	16b
Min	95	2	64	57	72	173	8	180	185	183
Med	122	8	86	79	93	211	18	223	225	221
Max	153	14	111	101	119	255	34	267	267	263
Avg-case	15636	990	11015	10183	11944	54208	4830	57272	57693	56764
Worst-case	255	65535	511	1023	65535	255	65535	511	1023	65535
Avg : Worst	61.32	0.02	21.56	9.96	0.18	212.58	0.07	112.08	56.40	0.87

Table 1: Random Incrementation Scheme Results

items we extract around 15k accesses with the base 8-bit strategy, significantly more than any other strategy however, the worst-case for the baseline is incredibly low. We encounter a similar although reduced issue between the split 9-bit strategy and 16-bit strategies in 256 items, where we get more average-case accesses with the 9-bit strategy, though the ratio is significantly better for the split 16-bit strategy. For that reason we have determined that using the split 16-bit strategy on a 256 item array is going to be the best for our implementation of the counting array.

5 Conclusion

Following the results of the second experiment we have decided to move forward with a 256 counter array with a maximum compression size of 252B. To represent each counter we will use a 16-bit counter split by the higher 8 bits and lower 8 bits. This serves as a nice compromise between the average cases and the worst cases. However, an issue introduced by this strategy is that most systems use a 128B cache line, meaning 2 accesses are needed as opposed to a single which can be costly.