A Time-to-Recache Case Study

Eric Storm Josef Spjut

Department of Engineering, Harvey Mudd College
March 28, 2014

Abstract

We present a case study of using the time to recache (TTR) metric to choose which cache configuration is optimal for a given benchmark and replacement policy. This work builds on prior work describing and implementing the TTR metric, and concludes that, while TTR can provide visualizations of cache behavior to build designer intuition into the behavior of a particular cache for a particular benchmark, the traditional method of simulating a wide variety of cache configurations to decide which is optimal is still required.

1. Introduction

Time to Recache(TTR) is an interesting metric to measure cache performance, but thus far few case studies exist to show its usefulness [5, 2]. This paper represents one such case study to show how TTR can be used to assist a cache designer in finding the optimal cache configuration. While in this particular example, it would have been just as practical to use another cache metric, like hit rate, or misses per 1000 instructions, TTR was useful in this case.

The idea behind this is that if you were to run a program with a given cache configuration and replacement strategy, you would want to minimize the miss rate. However if you only have the miss rate, you have to blindly guess what cache configuration would improve the miss rate. This means that you essentially have to test all possibilities in the solution space, which can take a significant time to simulate. By using TTR as a guide, you can gain insight into what will improve your miss rate, which allows you to selectively examine cache configurations to get to the optimal solution. This could potentially save an engineer considerable time waiting for simulations to run in situations where computing resources are limited or a quick decision is required.

2. Methodology and Results

We used the simulation framework [3] used previously for TTR studies [2] to generate TTR results. In addition, we implemented a web-based visualization tool that was used to generate all of the figures in this paper. For each figure, note that the legend identifies each simulation as follows:

`benchmark.policy.setbits-linebits-associativity`

where `benchmark` is the application trace being used, `policy` describes how the cache chooses how to replace lines when there is a conflict, and `setbits`, `linebits`, and `associativity` all represent the cache configuration for a particular simulation.

The procedure for interpreting TTR curves is as follows. First, we choose to hold the cache size constant. With this constraint, we have three parameters that can be varied, the number of sets, the line length, and the number of ways (associativity). As discussed above, a cache configuration is shown as (set bits)_(line bits)_(associativity ways). For example the configuration `5_6_4` has 2^5 sets, 2^6 bytes per line and 4 associativity ways. Because the cache size is held constant, an increase in any one parameter must be accompanied by a decrease in another parameter. Thus, if we limit ourselves to varying as little as possible at a time, we have six possible ways to modify a cache configuration:

1. Increase associativity, decrease sets
2. Increase associativity, decrease line length
3. Increase sets, decrease line length
4. Increase sets, decrease associativity
5. Increase line length, decrease sets
6. Increase line length, decrease associativity

From this, it is clear that without any insight into the direction that will lead to improvement, enumerating all possibilities would take exponential time! No need to fear, because TTR will allow us to apply heuristics to quickly converge on the optimal cache configuration. To demonstrate this in action, here's an example:

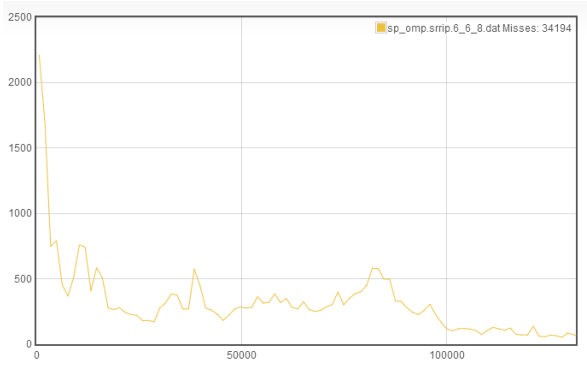

Figure 1: `sp_omp.srrip.6_6_8`

For this case, we use `sp_omp` from the NAS Parallel Benchmarks [1]. Among the traces that we have access to, it is a relatively cache intensive program, which makes it a good candidate to optimize. Additionally, we only consider a single

cache replacement policy at a time. So for this example, we search for the optimal cache configuration using the Static Re-reference Interval Prediction (SRRIP) replacement scheme from Jaleel et. al. [4]. To begin, we arbitrarily select the `6_6_8` cache configuration. The TTR curve for this is in Figure 1.

We begin by explaining the plot in Figure 1 for the uninitiated, since none of the plots in this paper have any axis labels, but they all use the same axes. The x axis is memory accesses to recache. Essentially, if a line is evicted from the cache and brought back in after a small number of memory access, it will end up close to the y axis. The y axis is the number of recache occurences of a particular duration that occur in the course of sampling. Note that the data is also bucketed and was sampled with a warmup period to remove cumpolsory misses. Notice in the legend that the series name is listed and includes the total misses, which we can use as a proxy for miss rate since the same number of cycles was run for each TTR curve we are comparing.

So, how do you interpret this graph? One thing to notice is what we call the rapid recaching (RR), which might also be referred to as thrashing. In this case, we see that among the roughly 34000 misses, only about 2200 were due to rapid recaching. This means that we can afford to decrease associativity and increase either the number of sets or the line length. Without any additional information, we have no good way to determine whether line length or set number should be increased. we found that by choosing either, it will eventually converge on the same answer. So for the purposes of brevity, we first increase the number of sets. This makes the new cache configuration `7_6_4` which is added in Figure 2.

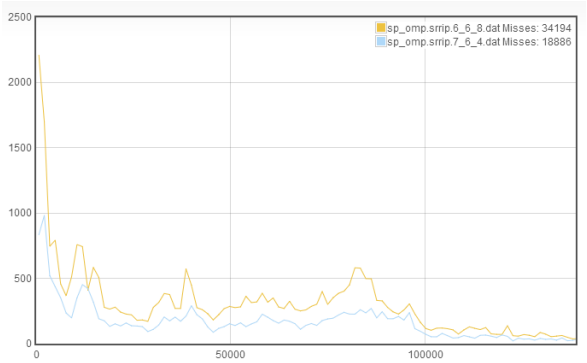

Figure 2: `sp_omp.srrip.7_6_4`

From this, we see that there is a signficant decrease in misses and the decrease comes roughly equally from all parts of the TTR curve. Additionally, despite decreasing associativity, the rapid recaching is still quite low. This suggests that decreasing the associativity to increase the number of sets, this results in a cache configuration of `8_6_2`, which is added in Figure 3.

We see that again we see an improvement in the misses. However, now the rapid recaching has skyrocketed! This

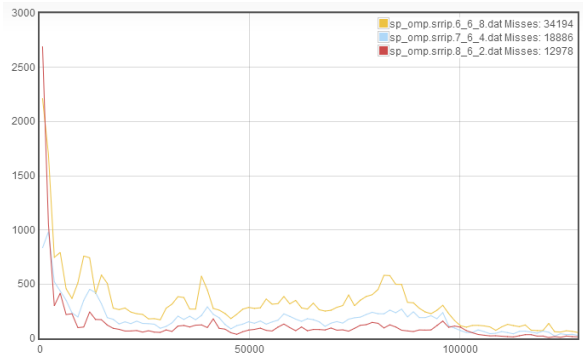

Figure 3: `sp_omp.srrip.8_6_2`

suggests that the increase in number of sets was beneficial, but the decrease in associativity was detrimental. Therefore, we try to increase associativity at the expense of line length, which gives the new cache configuration `8_5_4` found in Figure 4.

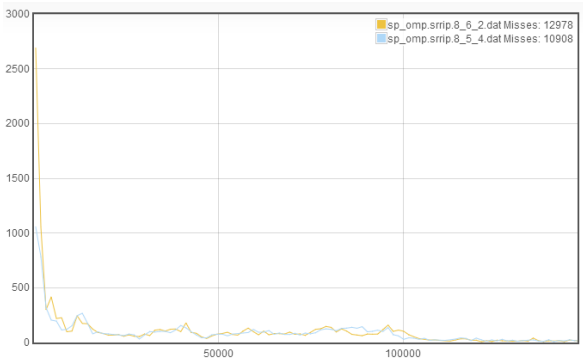

Figure 4: `sp_omp.srrip.8_5_4`

We again see improvement in the number of misses, however in this case, the improvement is due almost entirely to a decrease in the rapid recaching. This suggests that we may have room to further decrease the line length. Since the rapid recaching is not significantly larger than the slower recaching, this suggests that we use the decreased line length to increase the number of sets, to get a new cache configuration of `9_4_4` in Figure 5.

We see very minor improvement in the misses, because the slower recache rates have gone down but the rapid recaching has skyrocketed. This suggests that we increase the associativity. Lacking any intuition for whether line length or number of sets should be decreased, we abritrarily choose to decrease the line length so that we now have `9_3_8` as in Figure 6.

We see that the miss rate increased. This is due to an increase in the rapid recaching without significant improvement in the slower recaches. This leaves the only reasonable place for improvement to be to go back to the `9_4_4` configuration and decrease the number of sets instead of the line length, to get `8_4_8` found in Figure 7.

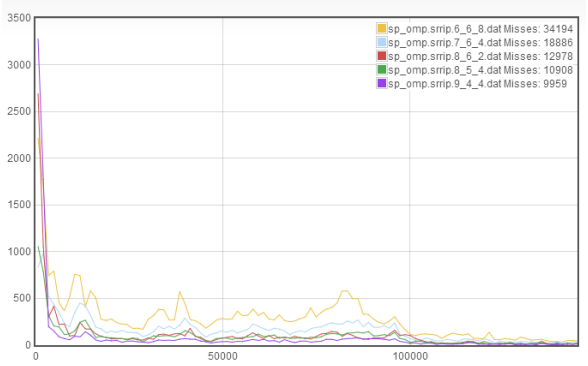

Figure 5: `sp_omp.srrip.9_4_4`

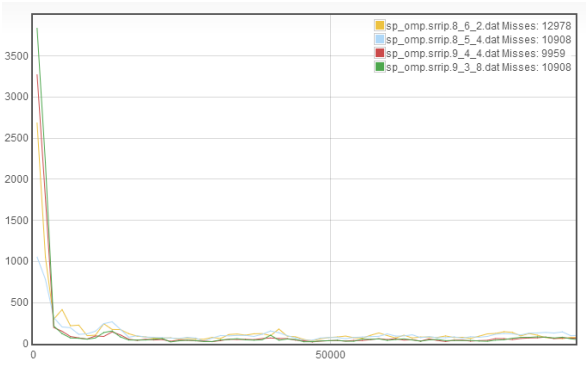

Figure 6: `sp_omp.srrip.9_3_8`

We see that again there is not an improvement in misses relative to `9_4_4`. We find that the rapid recache went down considerably, however there is no way to increase the associativity that we have not already tested! So we conclude that the optimal cache configuration is `9_4_4`. Having run nearly all of the reasonable cache configurations, this does in fact appear to be the optimal cache configuration.

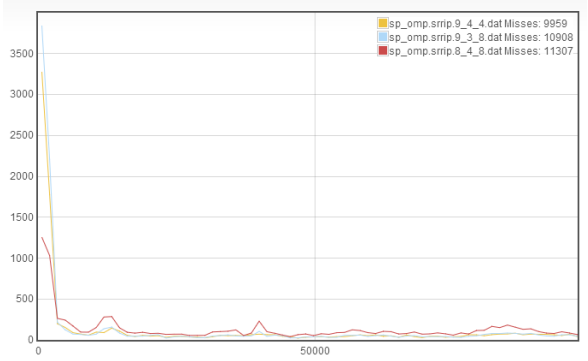

Figure 7: `sp_omp.srrip.8_4_8`

3. Conclusions

It takes some time getting familiarized with TTR curves to figure out both how the added information from a TTR curve to makes it more useful that miss rate. We tried for a while to compare the curves with the ideal cache replacement strategy for a given cache configuration in hopes that the TTR would show where exactly a sub-optimal algorithm was being less effective. We did not have a lot of luck with this approach.

Eventually we switched to using TTR to more quickly find an optimal cache configuration as presented here, but that process ended up exhausting nearly all relative cache configurations, so there was not much savings in terms of computation time. It may be interesting to run more cache configurations to determine whether this process consistently finds the optimal cache configuration. While TTR is an interesting way to build designer intuition about cache performance, it does not significantly change traditional design methodologies or simulating a wide variety of configurations to determine which one is optimal.

References

[1] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, and V. V. andd S. Weeretunga, "The NAS Parallel Benchmarks," *International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 63–73, Fall 1994.

[2] A. Carter, M. Korbel, P. Ning, and J. Spjut, "Qualitative Cache Performance Analysis," in *Technical Report*, 2013.

[3] ——, "TTR Github Repository," 2013, [Online; accessed 16-February-2017]. [Online]. Available: https://github.com/Clay-Wolkin-Fellowship/spock

[4] A. Jaleel, K. Theobald, S. Steely, and J. Emer, "High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)," in *Proceedings of ISCA*, 2010.

[5] J. Spjut and S. Pugsley, "Time to Recache: Measuring Memory Miss Behavior," *Technical Report*, Sep. 2011.