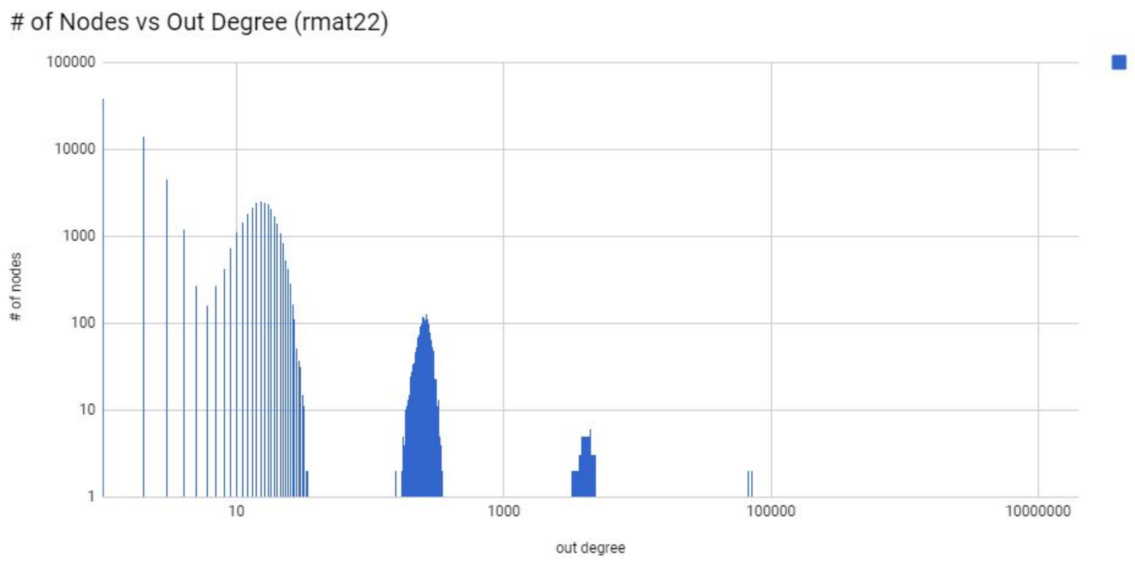
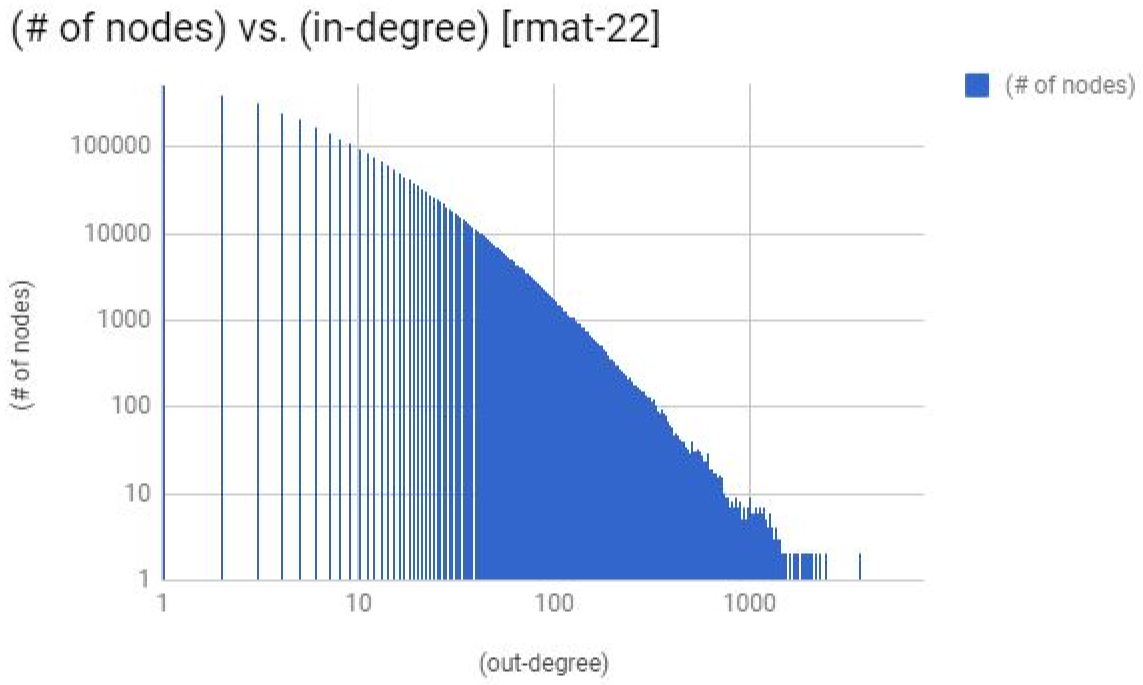
Joseph Bess, Krishna Ramdeep
CS 377P: Programming for Performance

Assignment 4: VTune

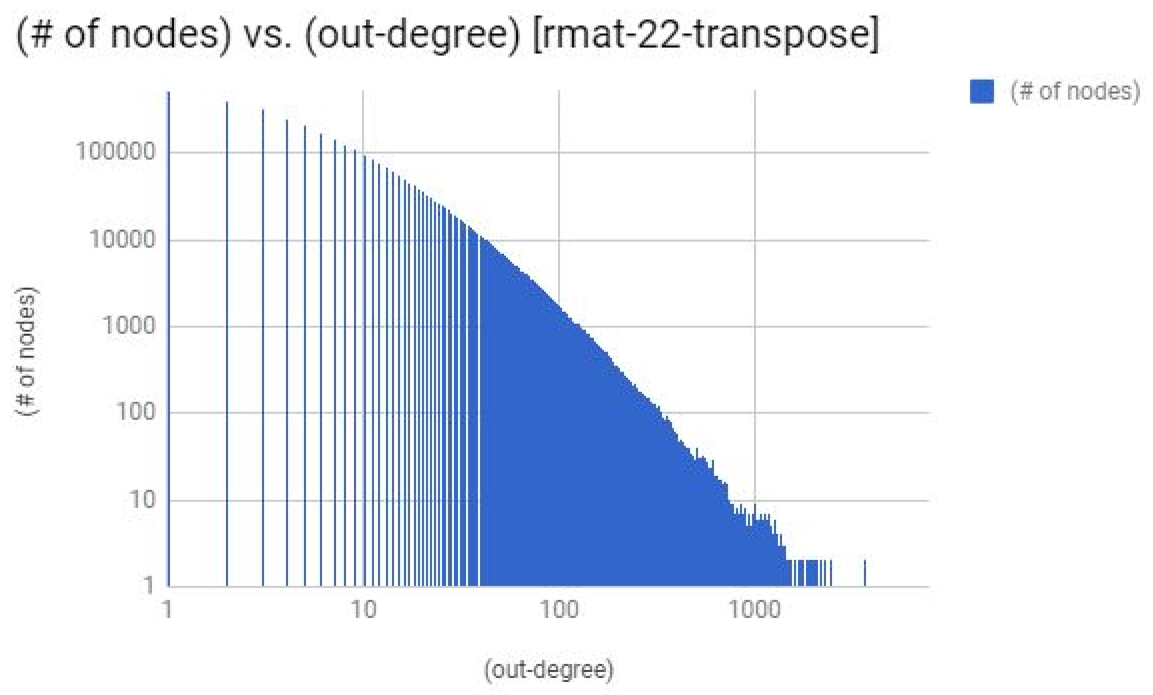
a)

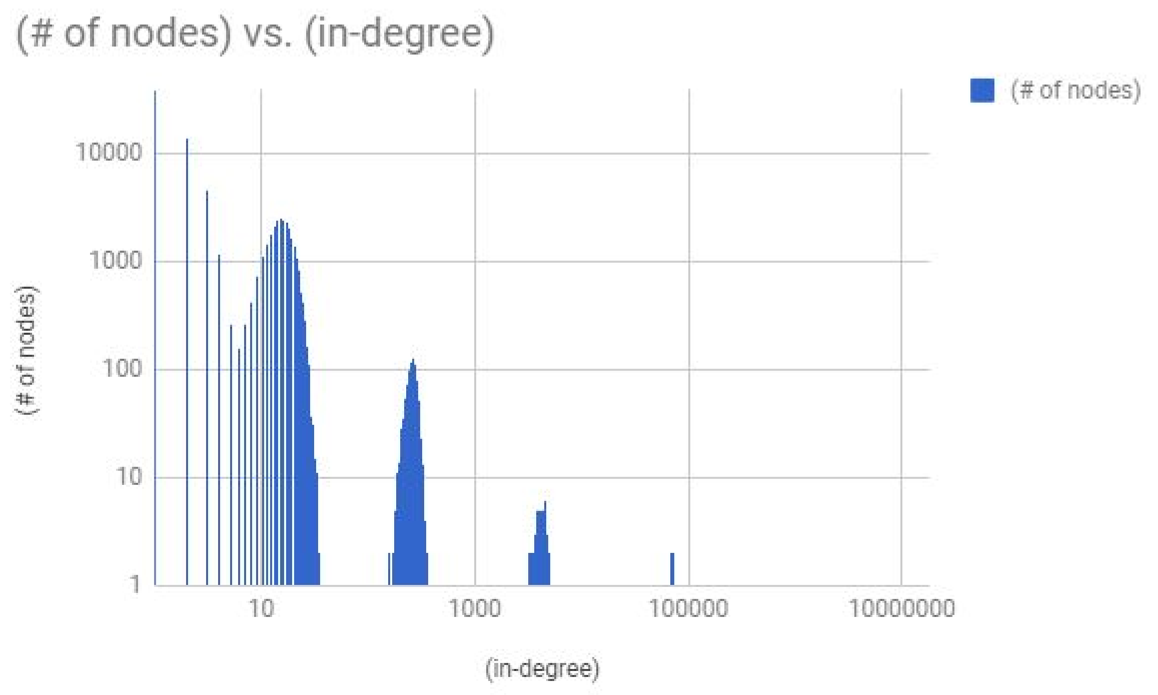
Our histograms for rmat-22:





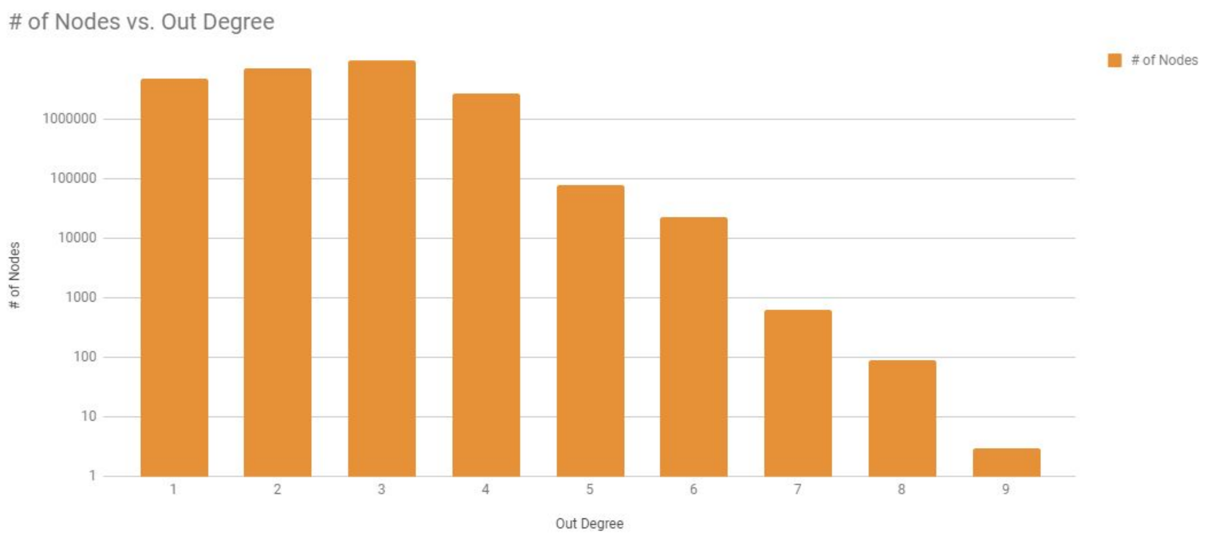
Our histograms for rmat-22 transpose:

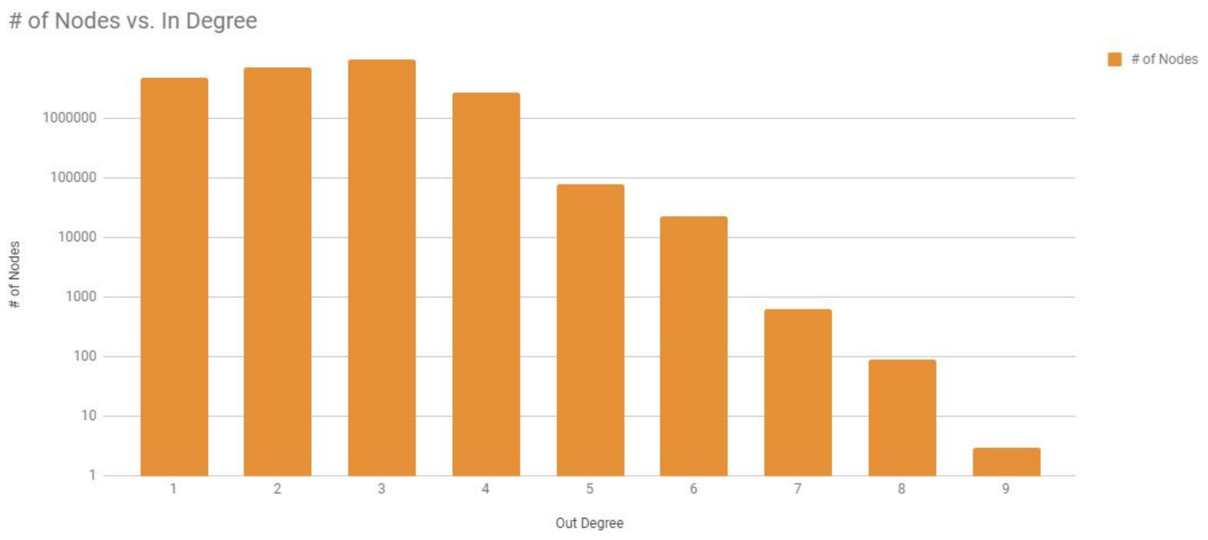




As one would expect, the in- and out-degree graphs are flipped for the graph and its transpose.

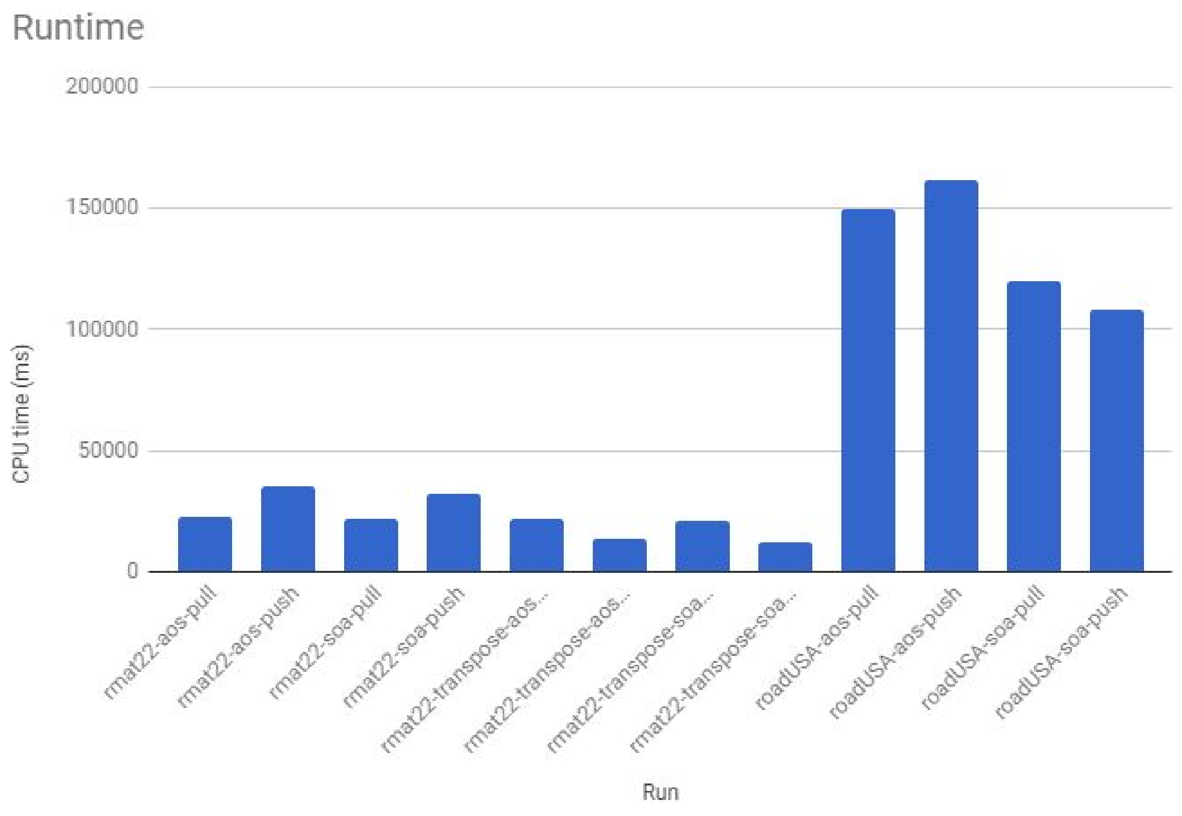
For roadUSA:



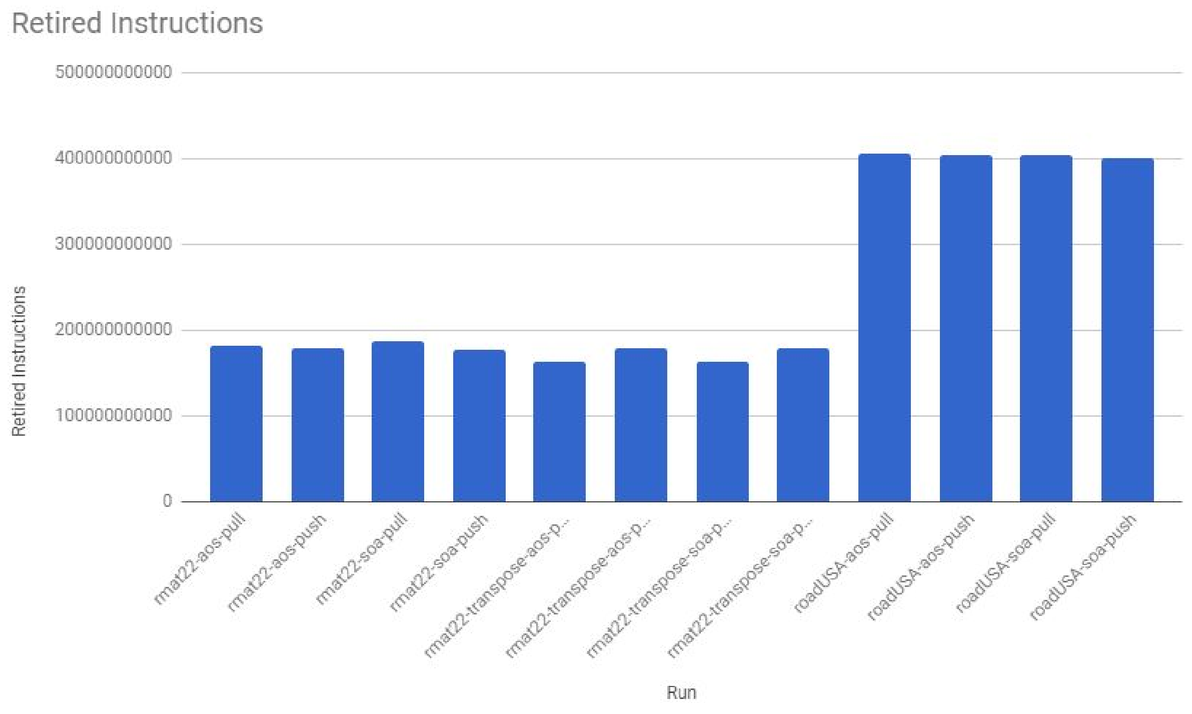


It appears that the roadUSA graph has the same in-degree and out-degree histogram, suggesting that every edge going into a node also comes out of that node.

b)
Runtime graph:

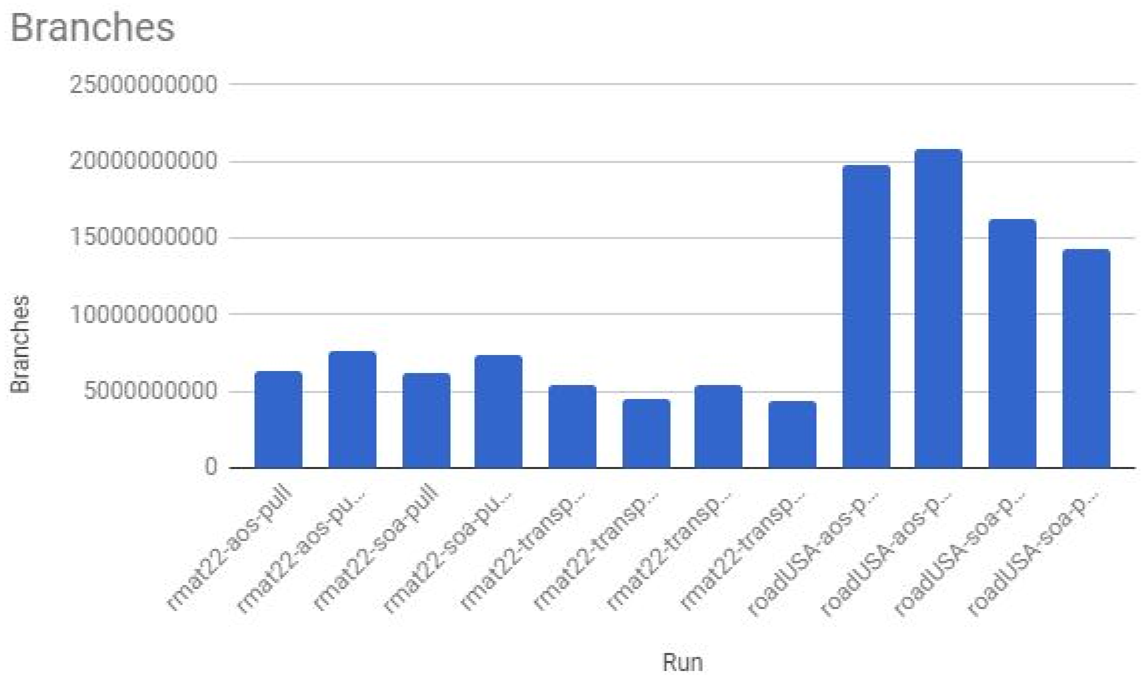


Retired instructions plot:

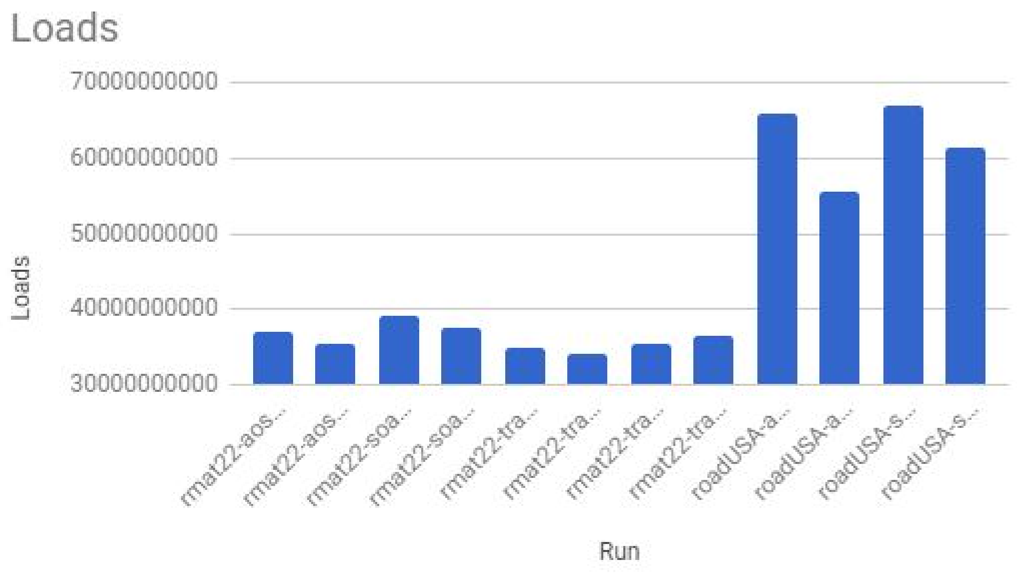


The graphs *do* resemble each other roughly, but there are some small differences. For example, the # of retired instructions for roadUSA (all runs) remains almost constant, but the runtimes decrease (especially for the run with node data representation "Struct of Arrays" and push-style algorithm).

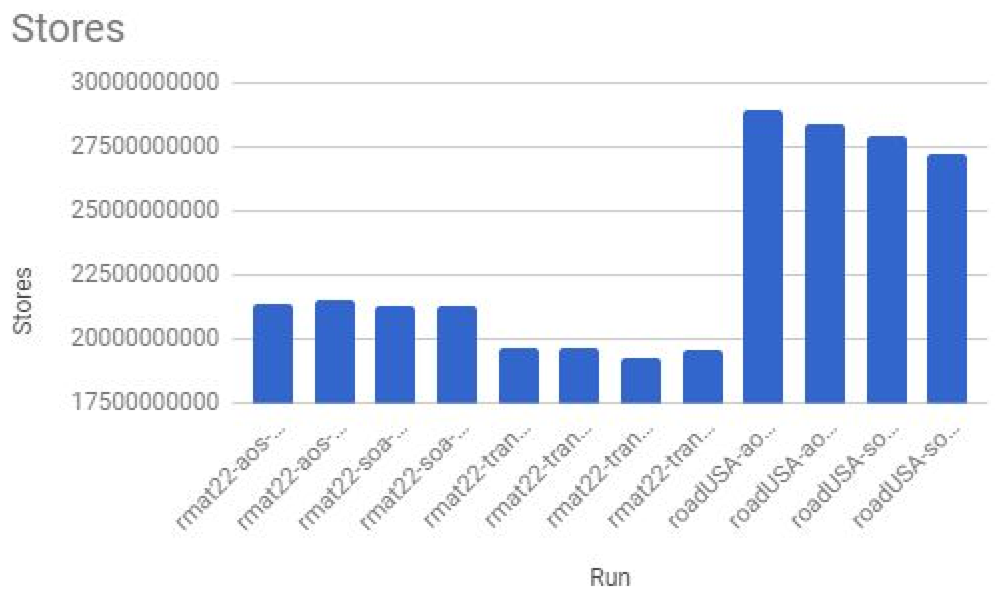
c)



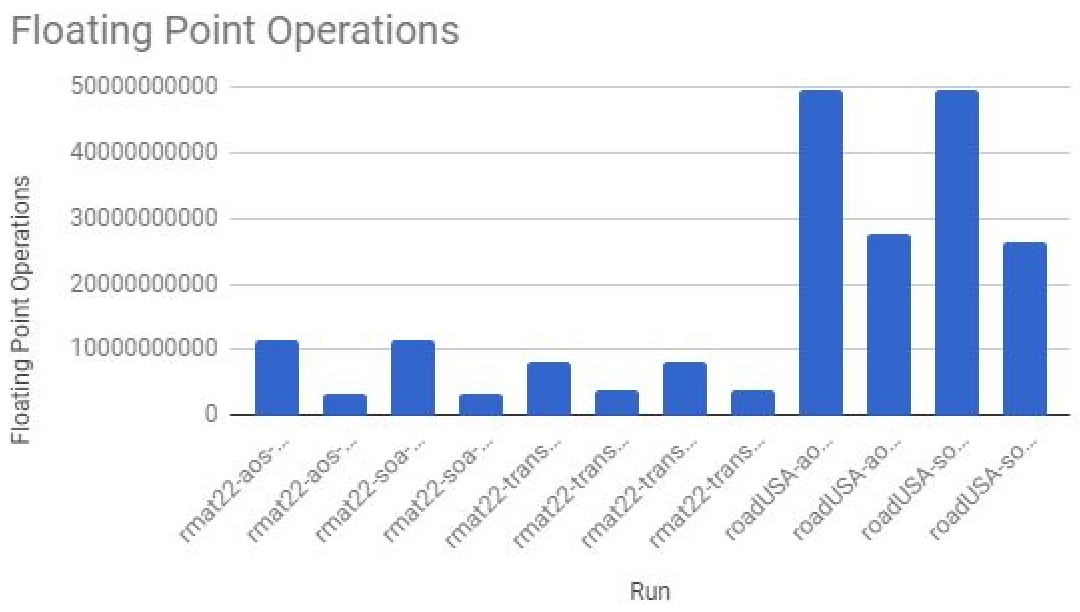
rmat-22 had more branches than rmat-22 transpose, and roadUSA had significantly more than both of them. This make sense given the amount of nodes present in each graph. Additionally, for rmat-22, it appears that a pull style algorithm leads to less branches, but the opposite is true for rmat-22-transpose. For roadUSA, it is hard to say if the algorithm had an effect on number of branches, but the way the data was stored (aos vs. soa) appears to have had an effect, with soa causing less branches.



Number of loads is more consistent. rmat-22 and rmat22-transpose have roughly the same number of loads. roadUSA has almost double that amount. This makes sense given that it took roadUSA more iterations to converge.

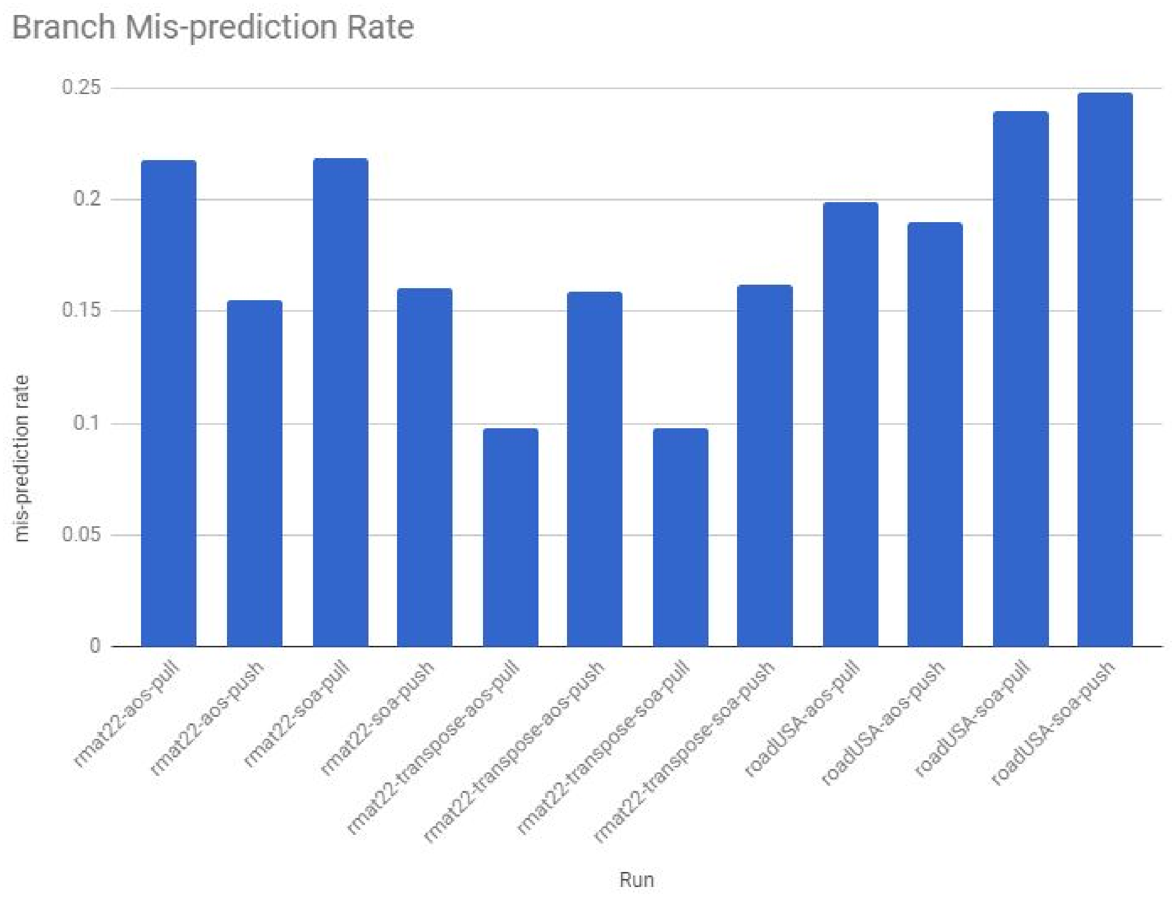


Number of stores is also consistent. rmat-22 and rmat22-transpose have roughly the same number of stores. roadUSA has more.
This makes sense given that it took roadUSA more iterations to converge, hence more intermediate pagerank values are being stored. The node data representation did not seem to have an effect. Neither did the algorithm.



For FP operations, the algorithm seems to be the main factor. Push-style consistently performed better (in terms of minimizing the number of operations) for all graphs. Node data representation did not seem to have an effect.
Overall, # of branches seems to be a decent predictor of runtime, as its plot most closely matches the runtime plot and the same patterns (pull-style better for rmat-22, while push-style better for rmat-22 transpose) emerge.

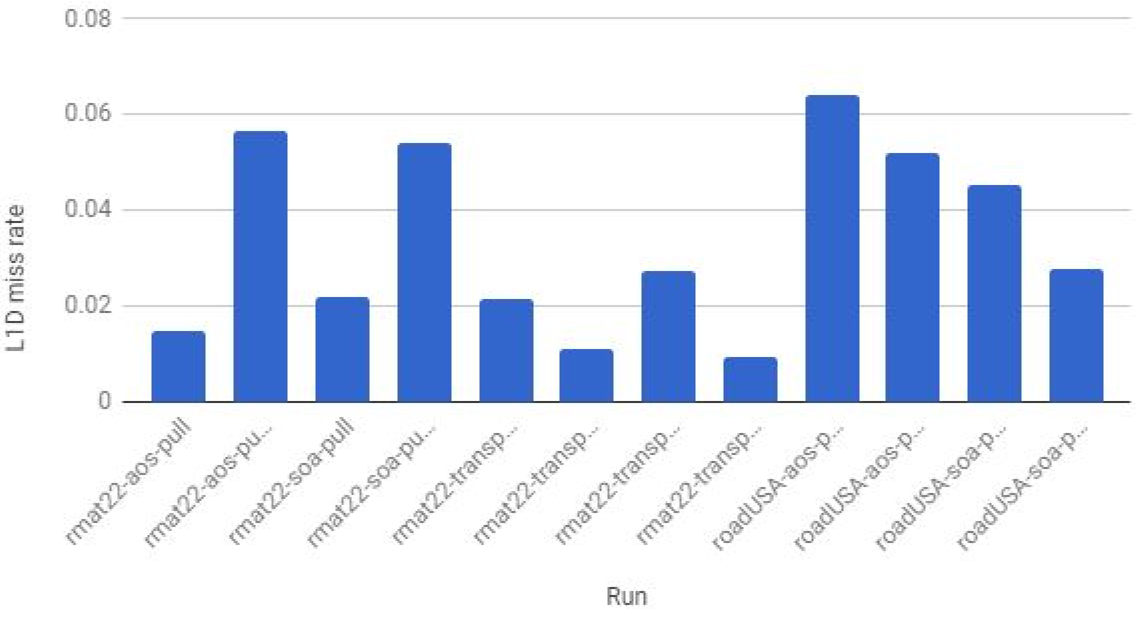
d)



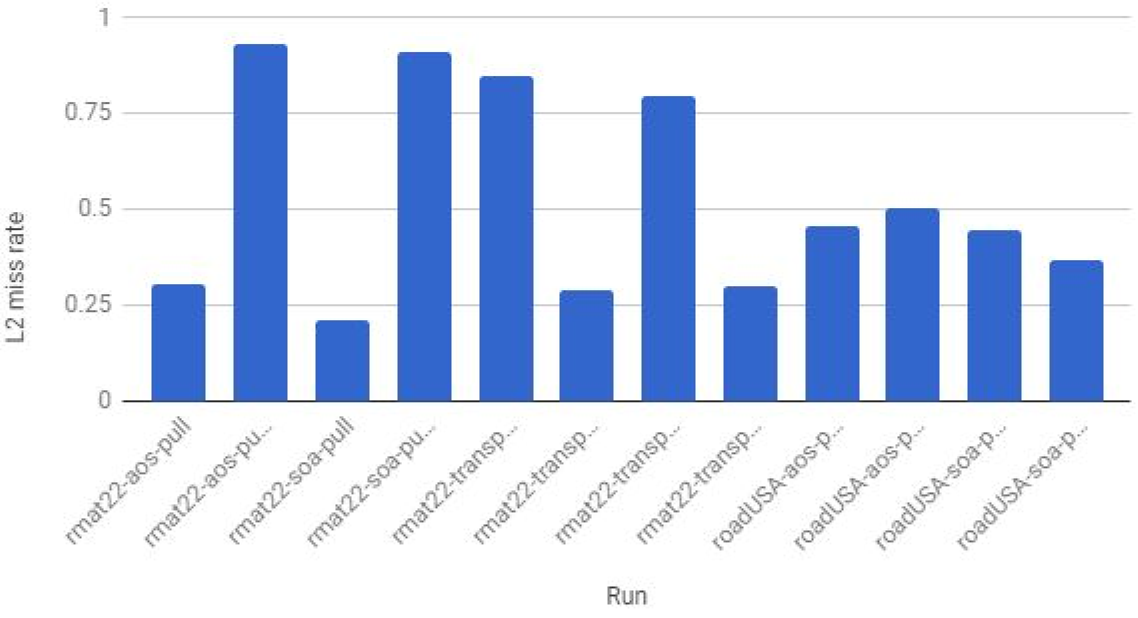
For rmat22, push-style is better at predicting branches, and node data representation seems to have no effect. For rmat22-transpose, pull-style works better, and again we see no real effect of the node data representation. However, for roadUSA, algorithm type seems to not matter, and the node data representation has the main effect, with aos having a better showing than soa.
We can't really predict runtime using the mis-prediction rate, as it seems to be counterintuitive. The higher mis-prediction rates appear to have lower runtimes. This is true for all graphs, all algorithms, and all node data representations.

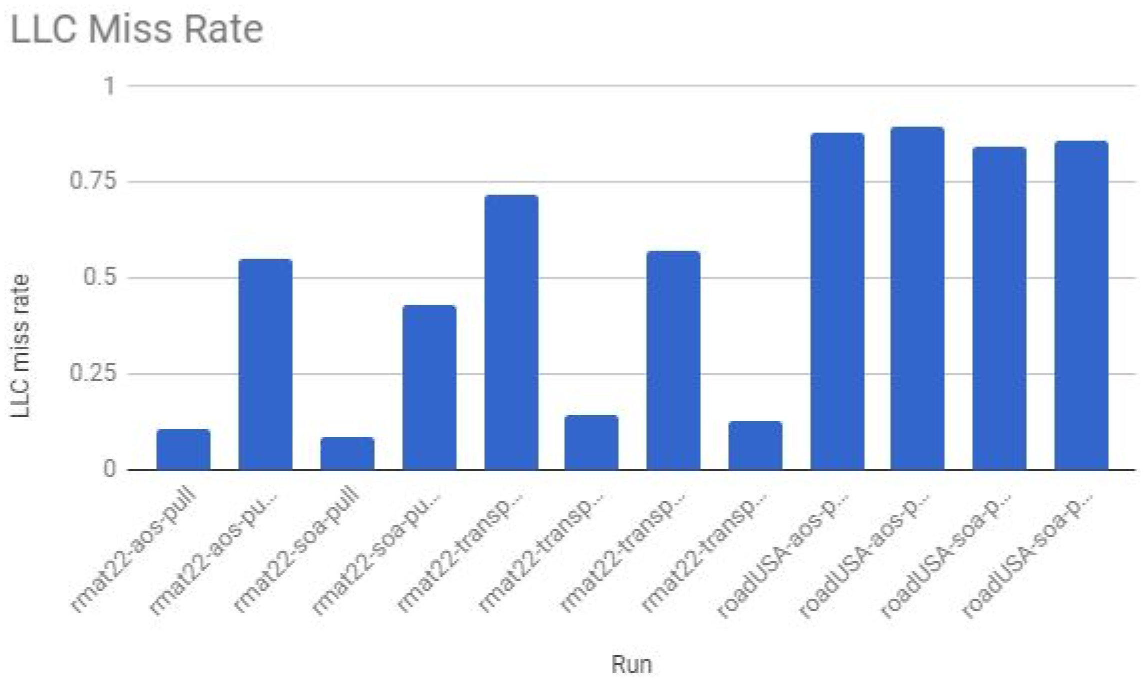
e)

L1D Miss Rate



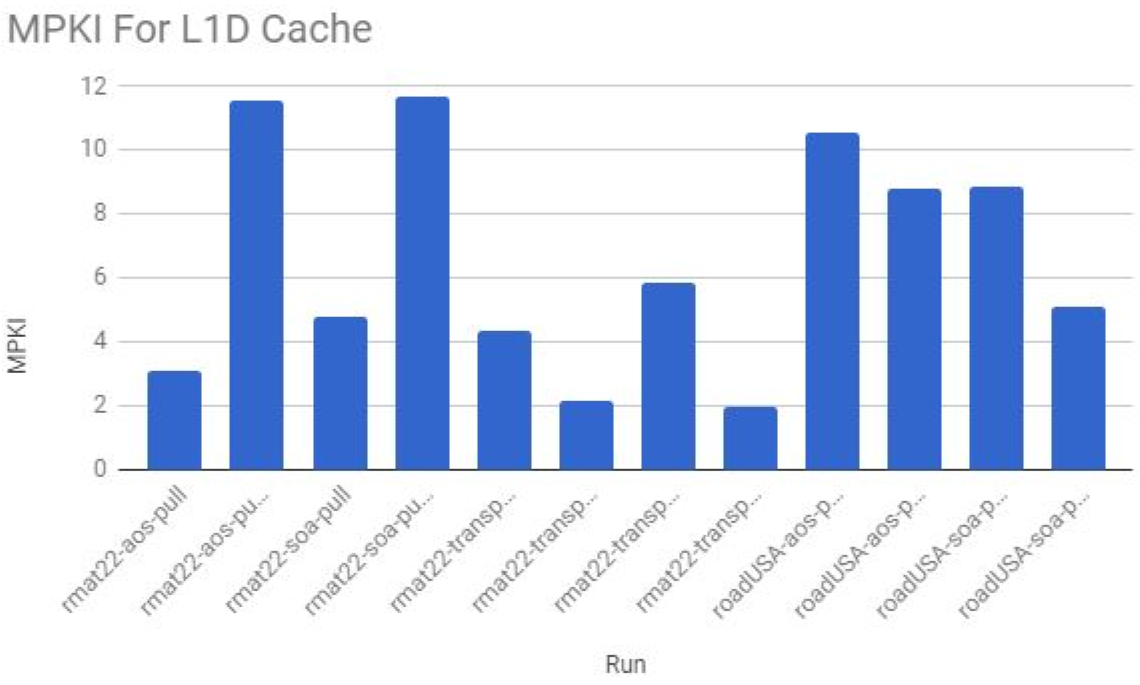
L2 Miss Rate

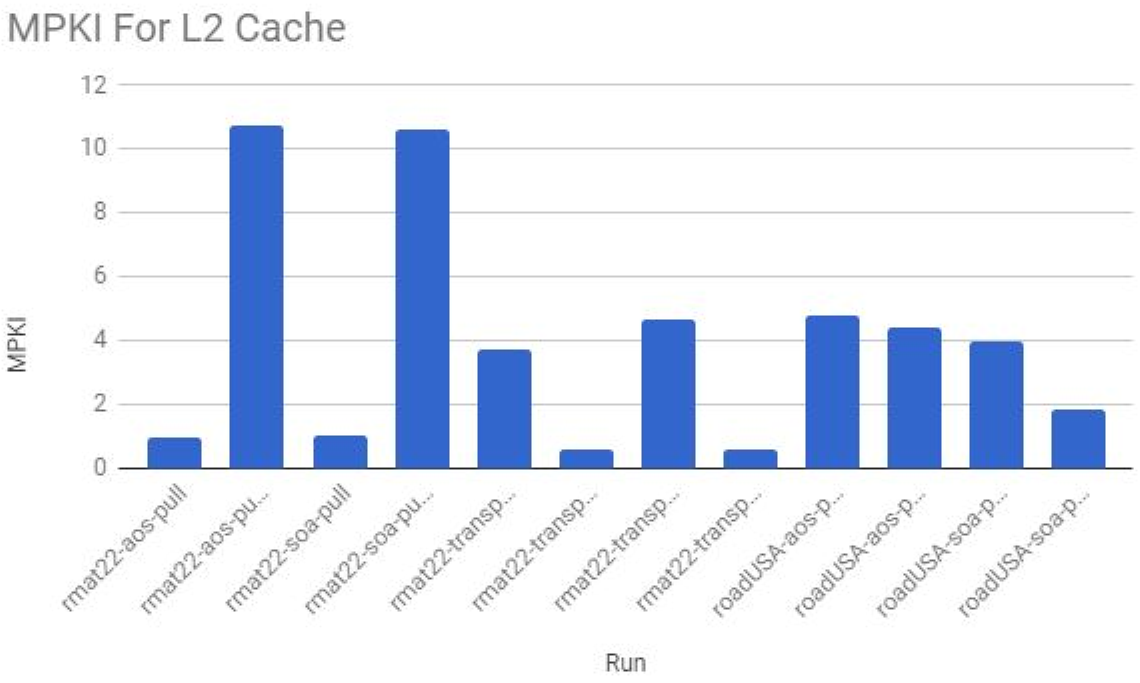


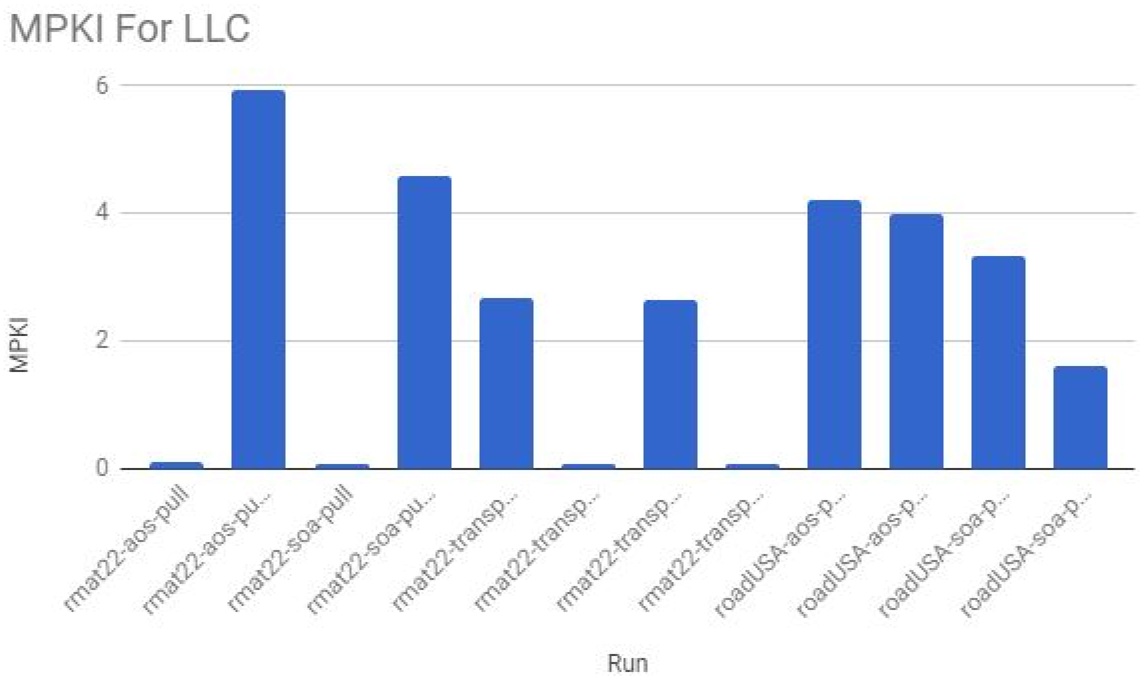


For rmat22, pull-style is better at cache misses, and node data representation seems to have no effect. For rmat22-transpose, push-style works better, and again we see no real effect of the node data representation. However, for roadUSA, pull-style is worse, but the node data representation has the main effect, with soa being better than aos. This is consistent across all cache types, except in the case of roadUSA with the LLC miss rate, where no matter the node data representation or the algorithm, miss rate is roughly the same. Cache miss rate is a good predictor of runtime, as the runs with a lower cache miss rate seem to perform better in terms of runtime.

f)

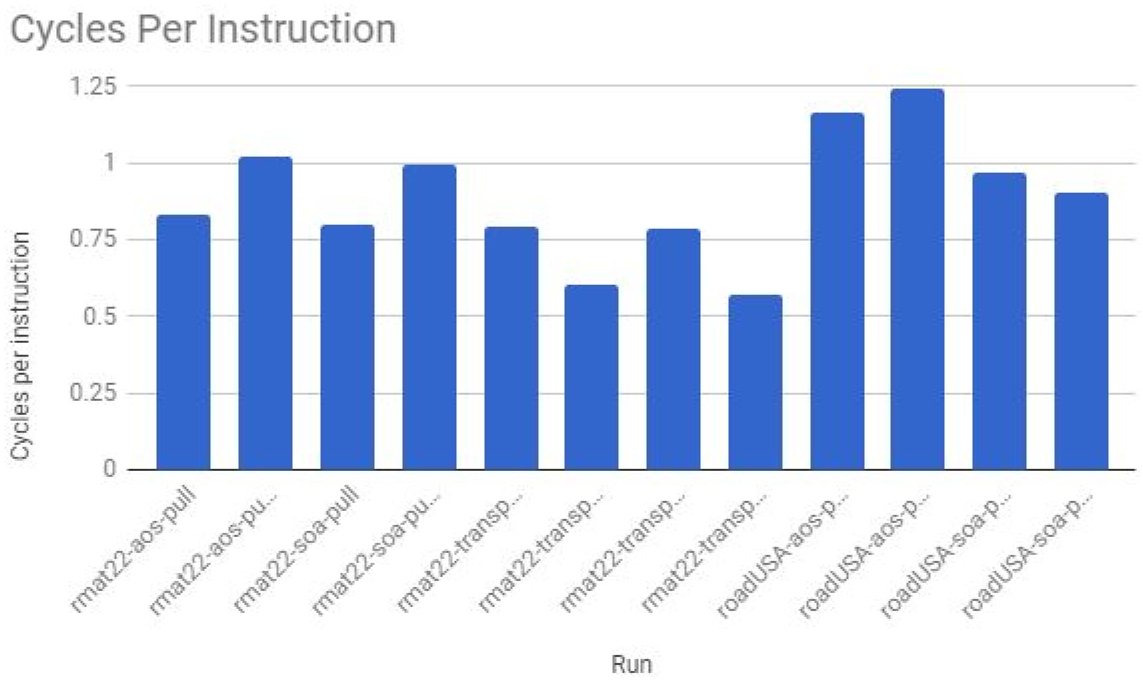

MPKI For L1D Cache
MPKI
12
10
8
6
4
2
0
rmat22-aos-pull
rmat22-aos-pu...
rmat22-soa-pull
rmat22-soa-pu...
rmat22-transp...
rmat22-transp...
rmat22-transp...
rmat22-transp...
roadUSA-aos-p...
roadUSA-aos-p...
roadUSA-soa-p...
roadUSA-soa-p...
Run


MPKI For L2 Cache
MPKI
12
10
8
6
4
2
0
rmat22-aos-pull
rmat22-aos-pu...
rmat22-soa-pull
rmat22-soa-pu...
rmat22-transp...
rmat22-transp...
rmat22-transp...
rmat22-transp...
roadUSA-aos-p...
roadUSA-aos-p...
roadUSA-soa-p...
roadUSA-soa-p...
Run



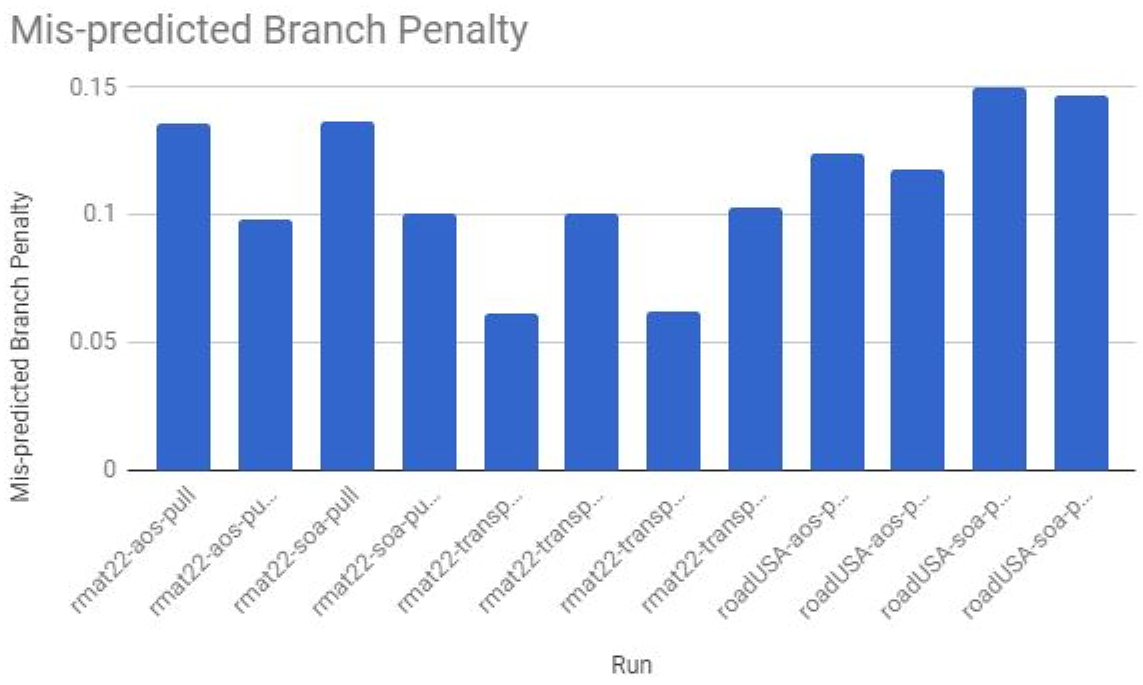
MPKI seems to match up with runtime very well, as we the familiar pattern of push-style being worse for rmat-22 but better for rmat-22-transpose. The roadUSA runtimes match up with the roadUSA MPKIs for the vaious caches as well. It suggests that MPKI is a good indicator of performance, and that the bottleneck in the microarchitecture comes from cache misses. Having a high rate of cache hits is vital if you want good performance.

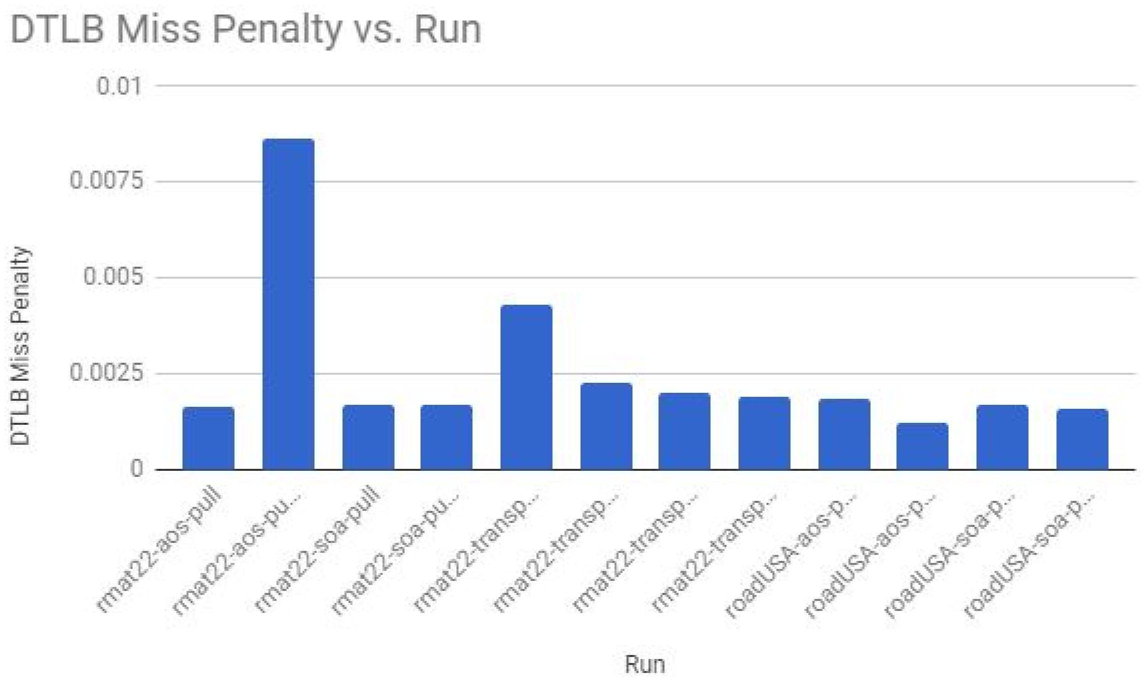
g)

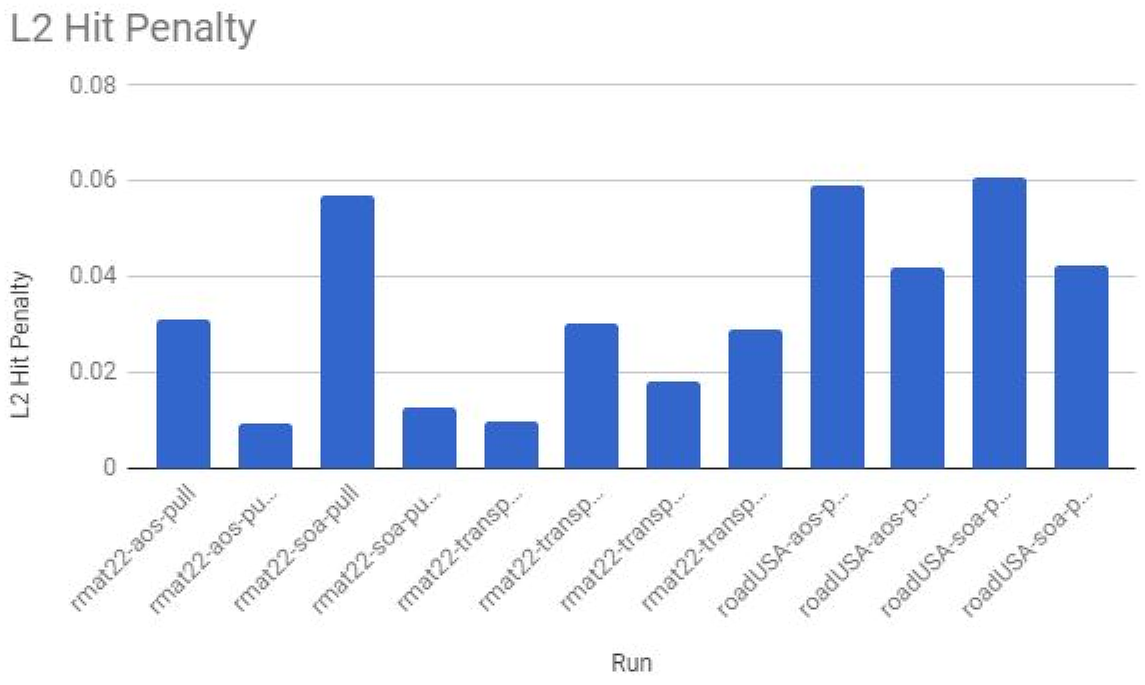


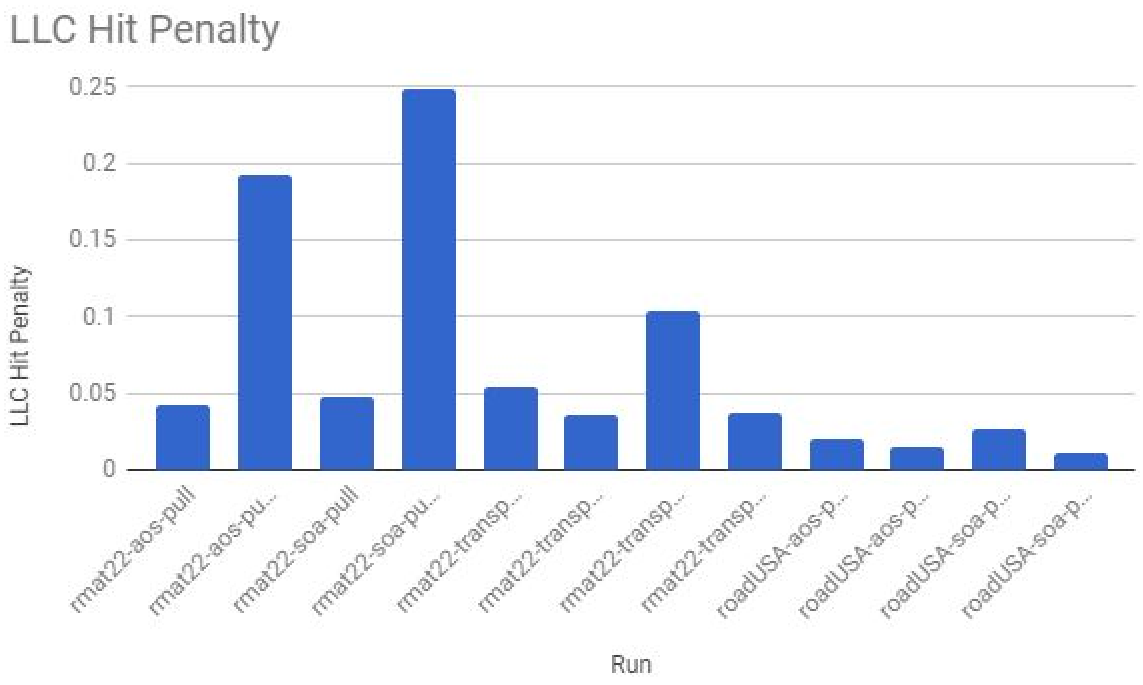
This matches runtime very well. Lower CPI corresponds to lower runtime, which makes sense!

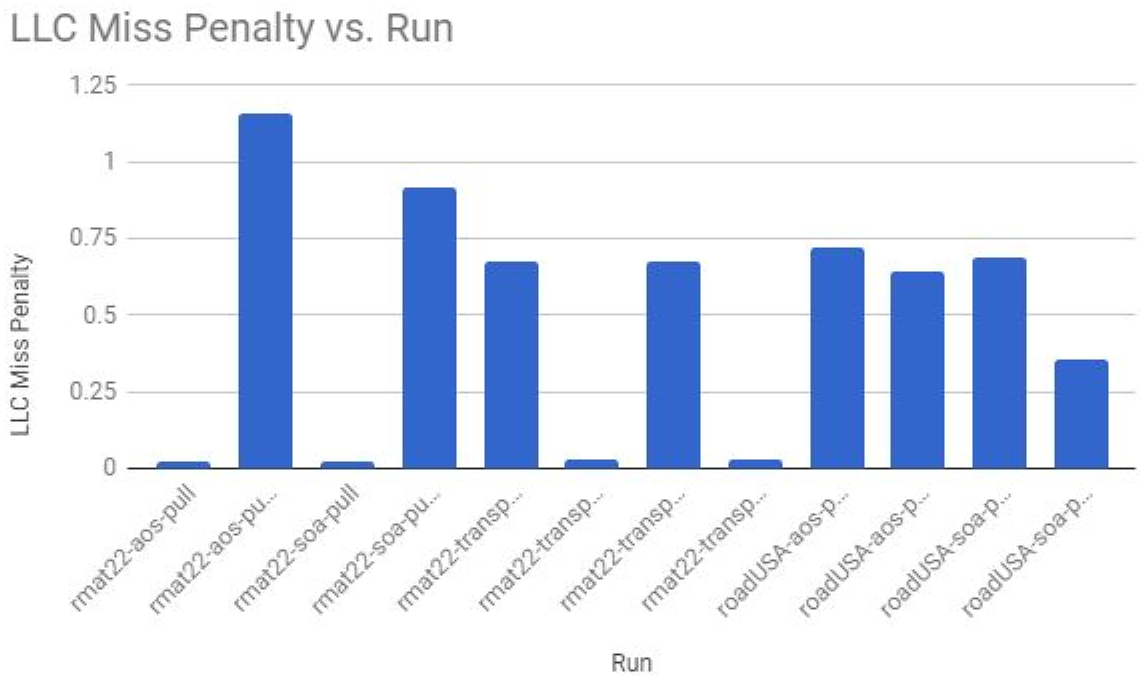
h)




DTLB Miss Penalty vs. Run
0.01
0.0075
0.005
0.0025
0
DTLB Miss Penalty
rmat22-aos-pull
rmat22-aos-pu...
rmat22-soa-pull
rmat22-soa-pu...
rmat22-transp...
rmat22-transp...
rmat22-transp...
rmat22-transp...
roadUSA-aos-p...
roadUSA-aos-p...
roadUSA-soa-p...
roadUSA-soa-p...
Run


L2 Hit Penalty
0.08
0.06
0.04
0.02
0
L2 Hit Penalty
rmat22-aos-pull
rmat22-aos-pu...
rmat22-soa-pull
rmat22-soa-pu...
rmat22-transp...
rmat22-transp...
rmat22-transp...
rmat22-transp...
roadUSA-aos-p...
roadUSA-aos-p...
roadUSA-soa-p...
roadUSA-soa-p...
Run





Some of the numbers (especially for LLC miss penalty) are greater than 1. This makes sense as if we miss LLC, that means we have to go to disk, which is costly. As for why the penalties don't sum to 1, this could be because pipelined instructions could skew the values. Another explanation could be because some events have higher weight than others (for example, an

LLC miss means going to disk, which is costly, whereas a L2 miss might get a hit in LLC, which is not as costly). The values concur with what we saw at the microarchitectural level: cache misses are hugely influential on performance.