A report on simulation results using SimpleScalar (Version 3.0) Simulator

CONTENTS

By
Gautam Chopra
905620730
gautam@vt.edu
ECE Department
Virginia Tech

Simulator & Benchmarks Used

SimpleScalar micro-architectural simulation suite is used for the simulations.
Version 3.0 of this simulator is used on Ubuntu 12.4 OS (which is installed as a virtual machine on a Mac). The PISA ISA is used to run the benchmarks.

The benchmarks used in the analysis are as follows:
fpppp – this is a quantum chemistry based benchmark
gcc – C compiler
go – based on the game go
vpr – FPGA circuit placement and routing

These are from the SPEC2000 benchmark suite.

All of the above were used to execute on the PISA architecture here (which is similar to MIPS architecture)

Cache Analysis

Using the sim-cache simulator for cache analysis.

Part I : Analysis on substitution policies:

The replacement policies being investigated are as follows:

1. **LRU** – This is called the least recently used algorithm, where the page that was accessed farthest in terms of time, is removed.
2. **FIFO** – This removes the page that has been in the cache the longest
3. **Random** – Randomly choose a page to evict from memory

For this analysis, I would take a 16 KB L1 cache, which is split into 8KB L1 instruction cache and 8KB L1 data cache.
Also, I will not keep any L2 cache in this case.

Defining a 8KB data cache, with LRU poliy is done as folows:
-cache:dl1 dl1:256:32:1:l

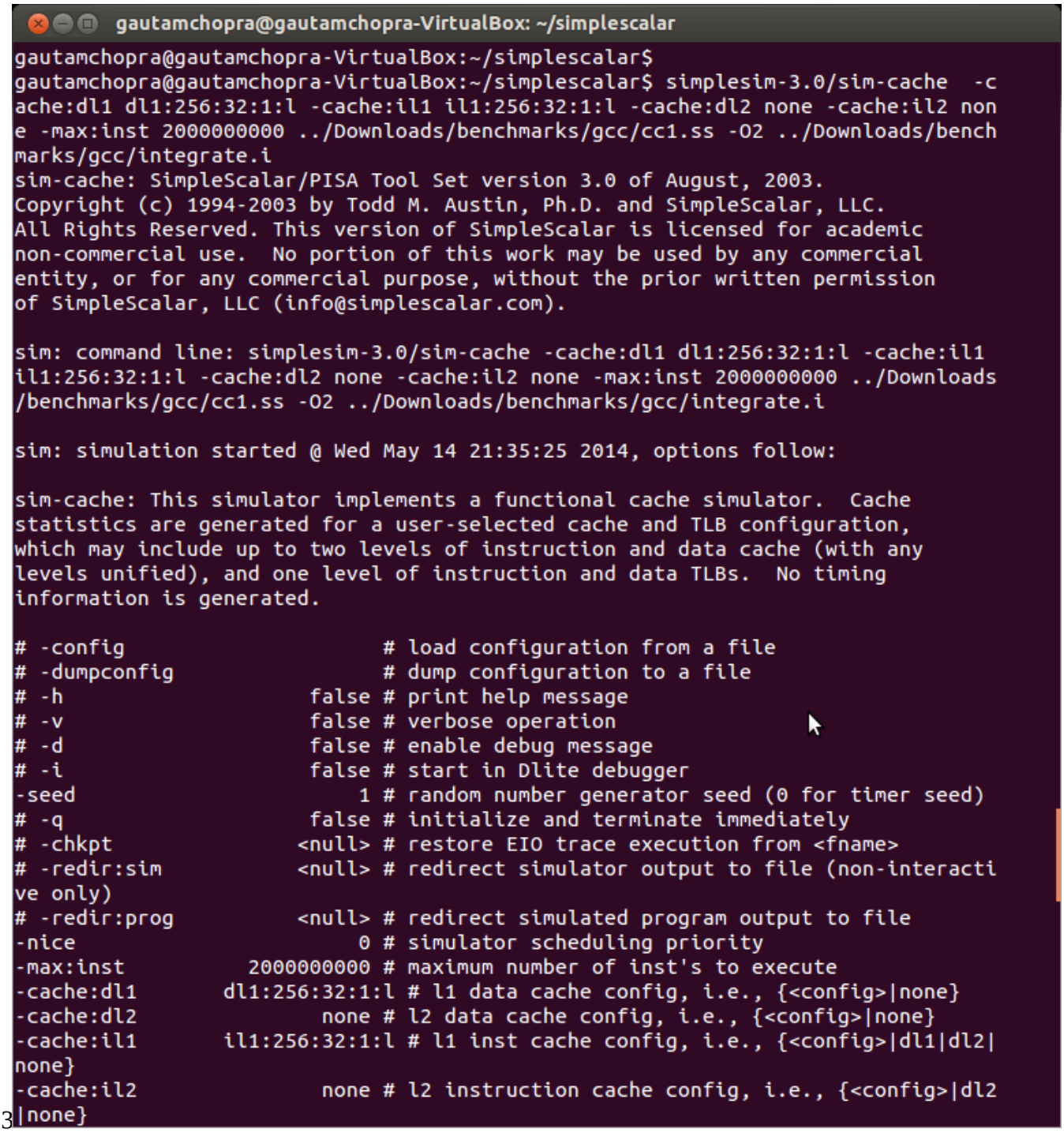
Hence the complete definitions look as follows

For LRU policy:
simplesim-3.0/sim-cache -cache:dl1 dl1:256:32:1:l -cache:il1 il1:256:32:1:l -cache:dl2 none -cache:il2 none

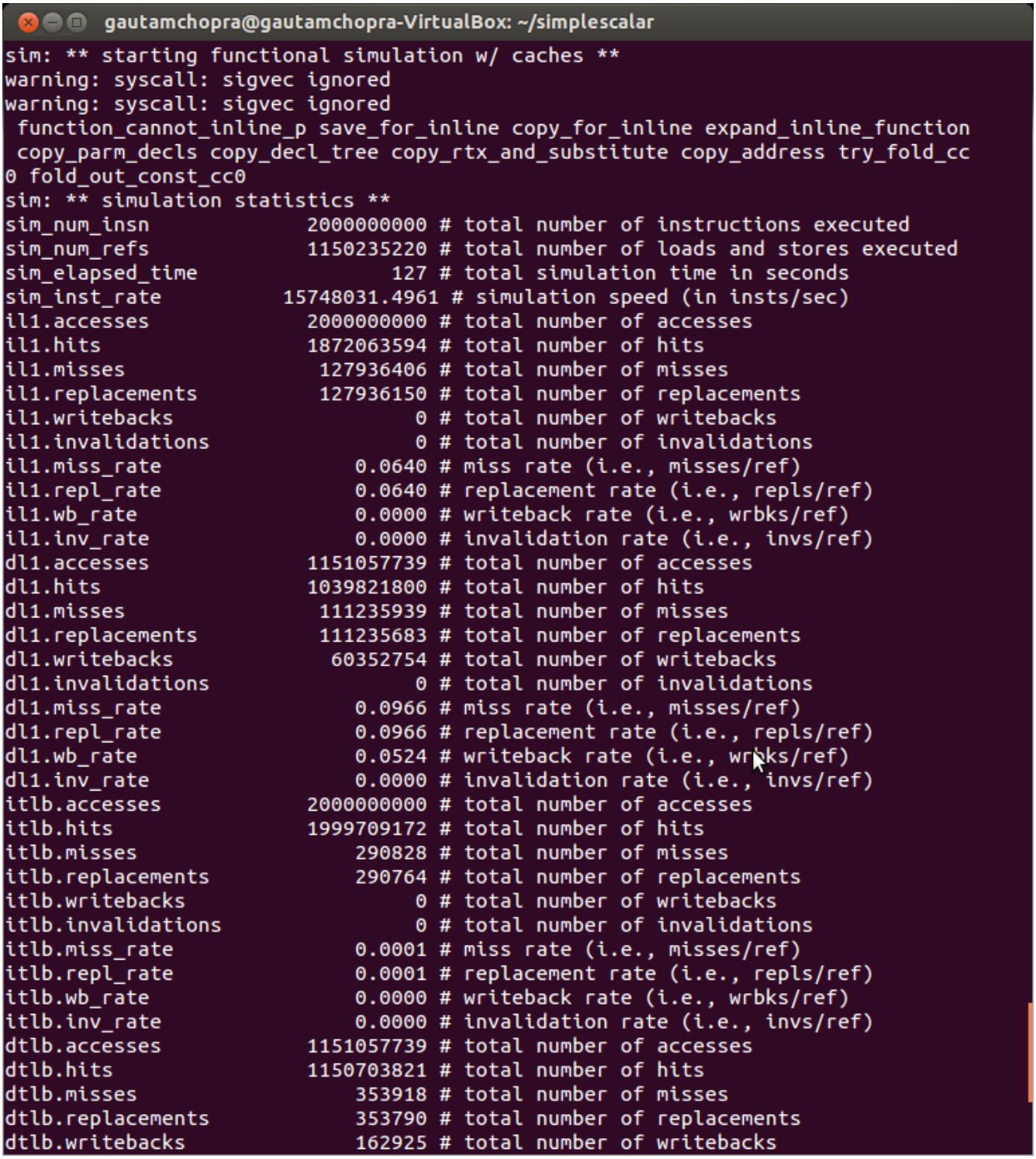
For FIFO policy:
simplesim-3.0/sim-cache -cache:dl1 dl1:256:32:1:f -cache:il1 il1:256:32:1:f -cache:dl2 none -cache:il2 none

For random policy:
simplesim-3.0/sim-cache -cache:dl1 dl1:256:32:1:r -cache:il1 il1:256:32:1:r -cache:dl2 none -cache:il2 none

A screen-shot of cache specification and simulation for GCC benchmark is as shown below:



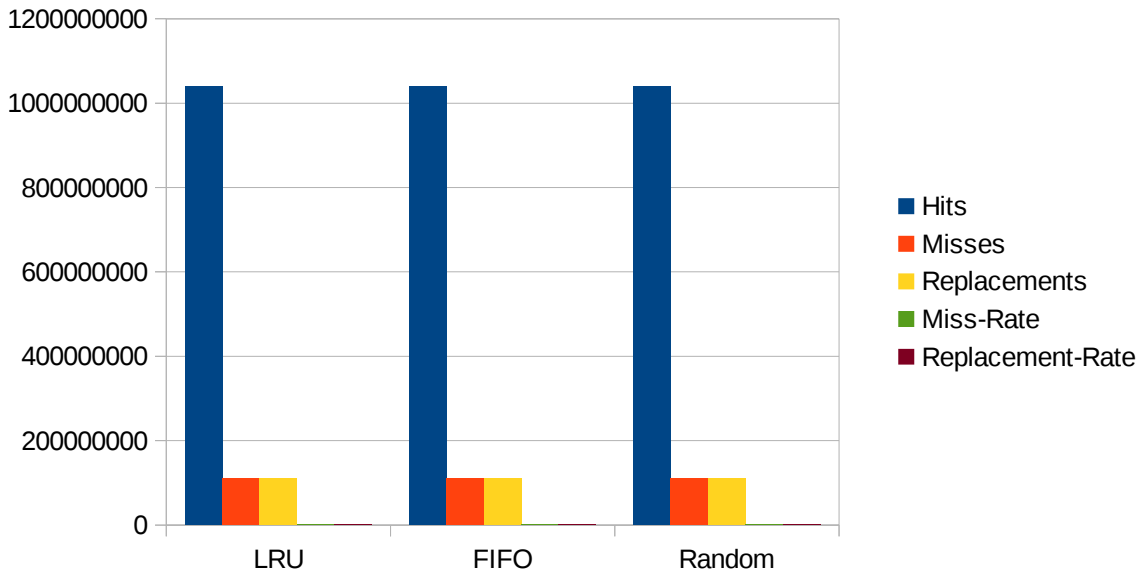
Simulation output screen-shots for GCC benchmark look as shown:



The simulation results are compared for the three given replacement policies, and the graphical results for all the benchmarks are as follows:

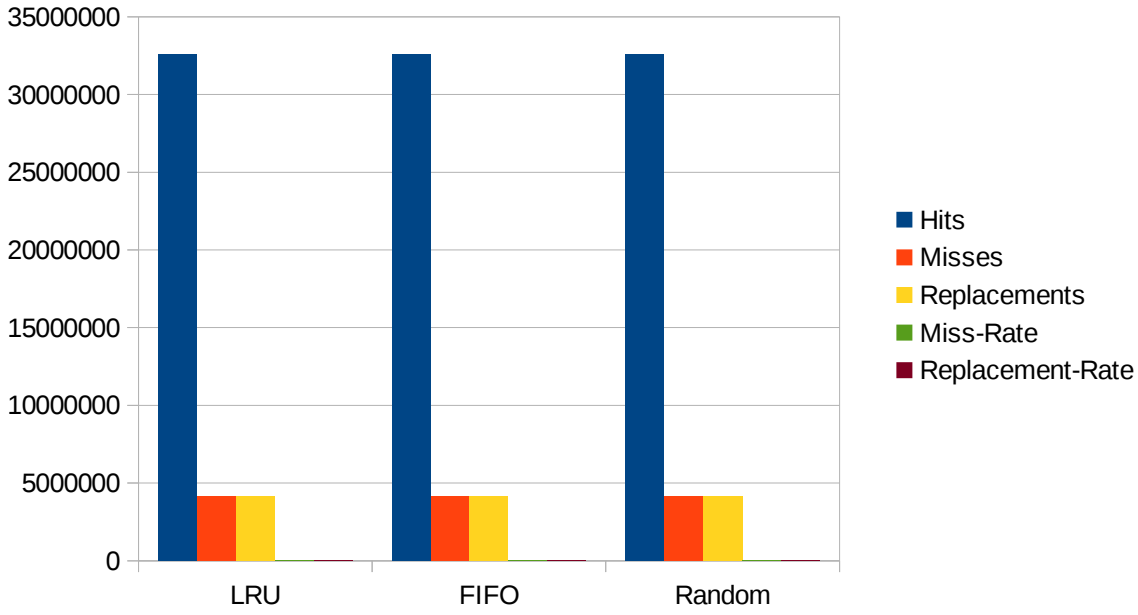
To study results from here on, I would focus on the data cache result analysis, as they turn out to be more interesting than the instruction cache.

1. GCC



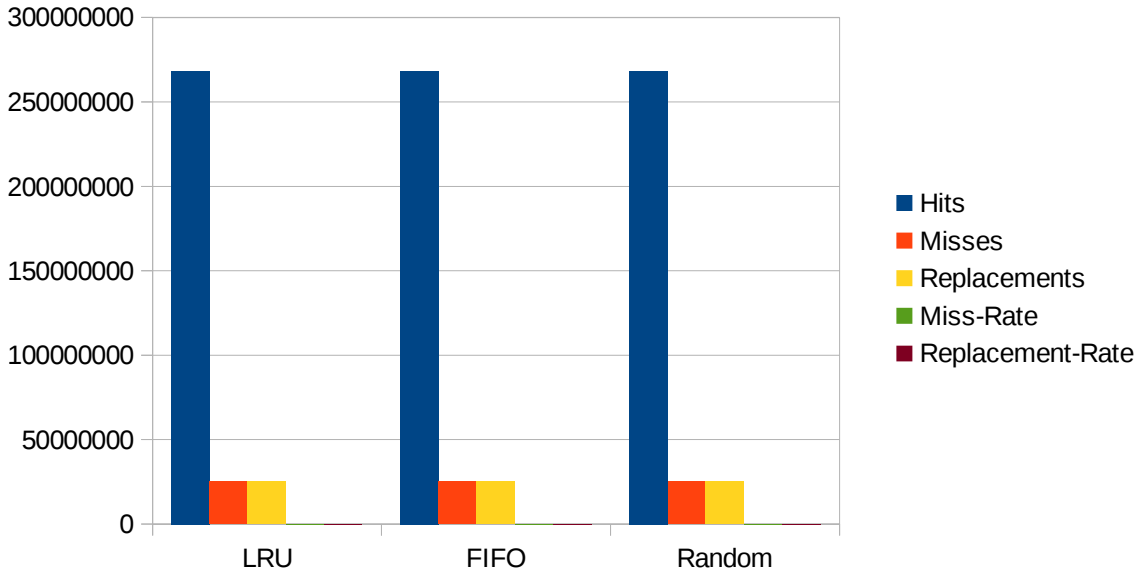
2. GO

	Hits	Misses	Replacements	Miss-Rate	Replacement-Rate
LRU	32566852	4149890	4149634	0.113	0.113
FIFO	32566852	4149890	4149634	0.113	0.113
Random	32566852	4149890	4149634	0.113	0.113



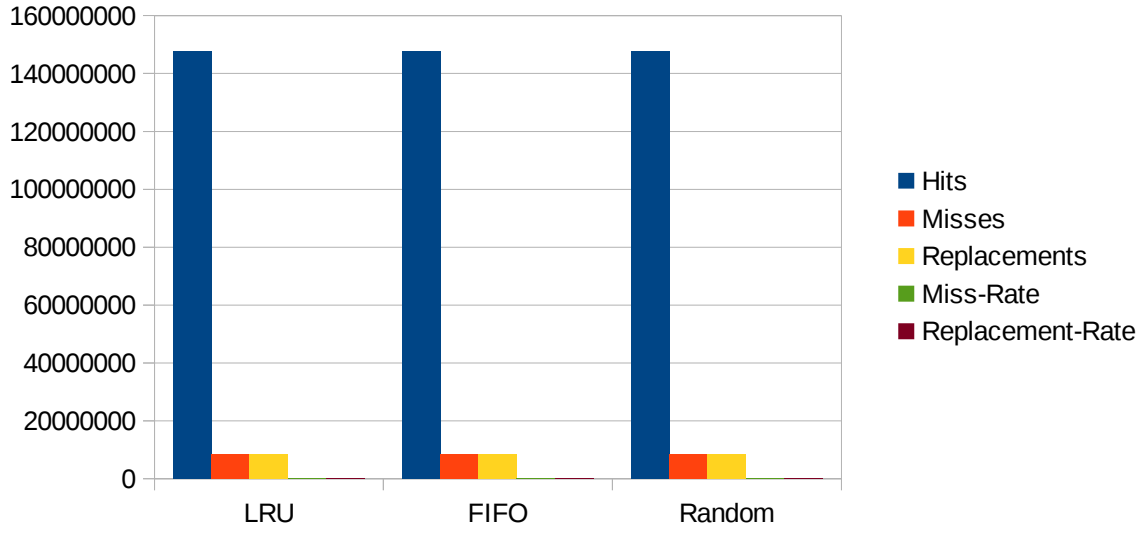
3. VPR

	Hits	Misses	Replacements	Miss-Rate	Replacement-Rate
LRU	268315538	25465098	25464842	0.0867	0.0867
FIFO	268315538	25465098	25464842	0.0867	0.0867
Random	268315538	25465098	25464842	0.0867	0.0867



4. FPPPP

	Hits	Misses	Replacements	Miss-Rate	Replacement-Rate
LRU	147735856	8438620	8438364	0.054	0.054
FIFO	147735856	8438620	8438364	0.054	0.054
Random	147735856	8438620	8438364	0.054	0.054



RESULTS:

As we can see from the 4 benchmarks above, the replacement policies don't show any differences in terms of the parameters considered. (Except simulation time and simulation rate, everything else matches).

Part II : Analysis on associativity

In order to test associativity, let us again take the data cache, but with a different base configuration, which is a 32KB cache:

-cache:dl1 dl1:1024:32:1:l

Now, to check the associativity, we would change the 4th parameter to 1,2,4,8 … (in multiples of 2)

*Here, the number of sets would be managed(reduced) to keep total cache size same as associativity increases.

Hence, the setup for the associativity test is as shown:

1-way:
simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:1:l -cache:dl2 none -cache:il2 none

2-way:
simplesim-3.0/sim-cache -cache:dl1 dl1:512:32:2:l -cache:dl2 none -cache:il2 none

4-way:
simplesim-3.0/sim-cache -cache:dl1 dl1:256:32:4:l -cache:dl2 none -cache:il2 none

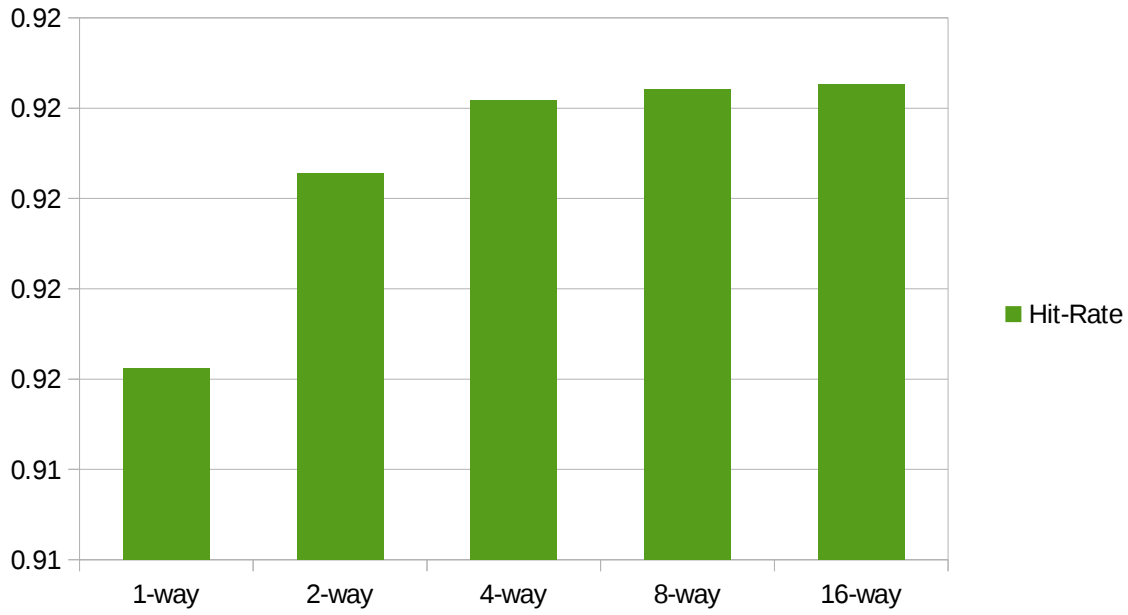
8-way:
simplesim-3.0/sim-cache -cache:dl1 dl1:128:32:8:l -cache:dl2 none -cache:il2 none

16-way:
simplesim-3.0/sim-cache -cache:dl1 dl1:64:32:16:l -cache:dl2 none -cache:il2 none

The results compare the hit rate and miss rates for each benchmark:

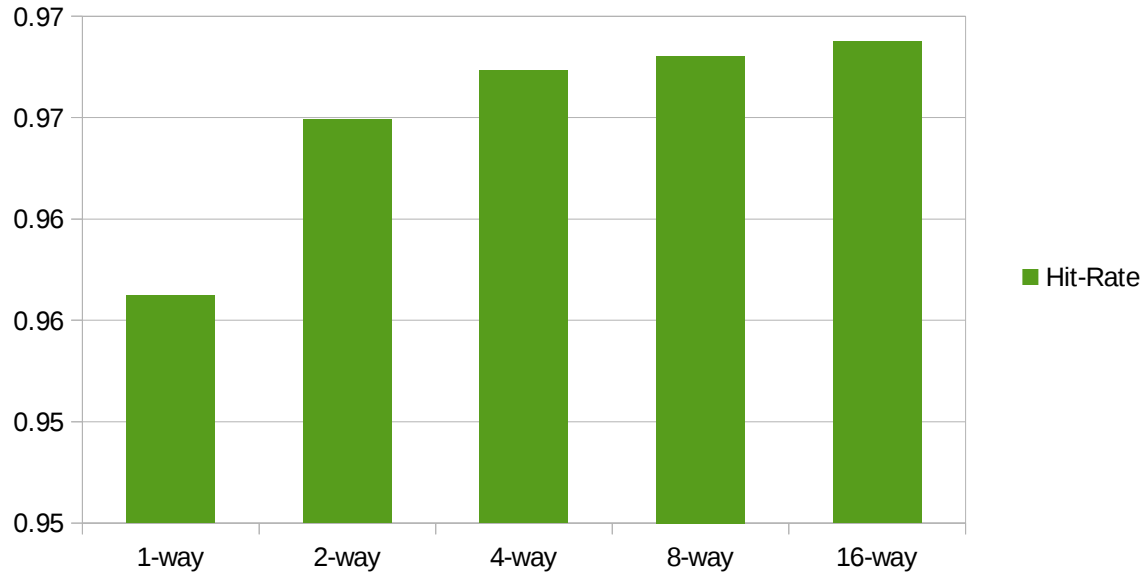
1. GCC

	Accesses	Hits	Misses	Hit-Rate	Miss-Rate
1-way	1151057739	1053363221	97694518	0.91512631	0.0849
2-way	1151057739	1055842048	95215691	0.917279831	0.0827
4-way	1151057739	1056770345	94287394	0.918086304	0.0819
8-way	1151057739	1056910324	94147415	0.918207913	0.0818
16-way	1151057739	1056978928	94078811	0.918267514	0.0817



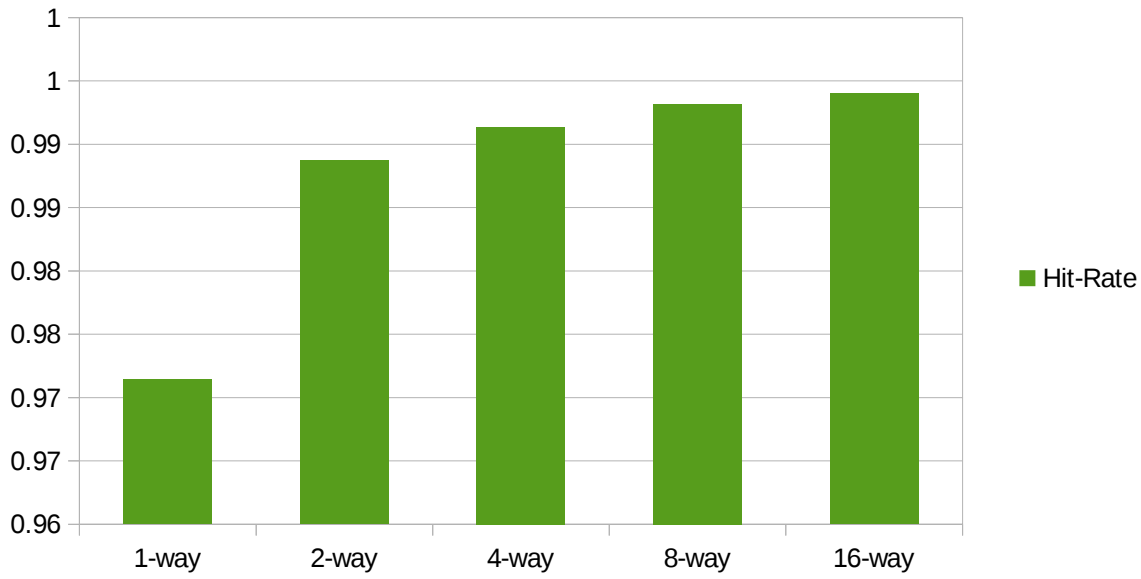
2. VPR

	Accesses	Hits	Misses	Hit-Rate	Miss-Rate
1-way	293780636	280922189	12858447	0.956231128	0.0438
2-way	293780636	283477983	10302653	0.964930796	0.0351
4-way	293780636	284180490	9600146	0.96732206	0.0327
8-way	293780636	284392174	9388462	0.968042611	0.032
16-way	293780636	284600247	9180389	0.968750871	0.0312



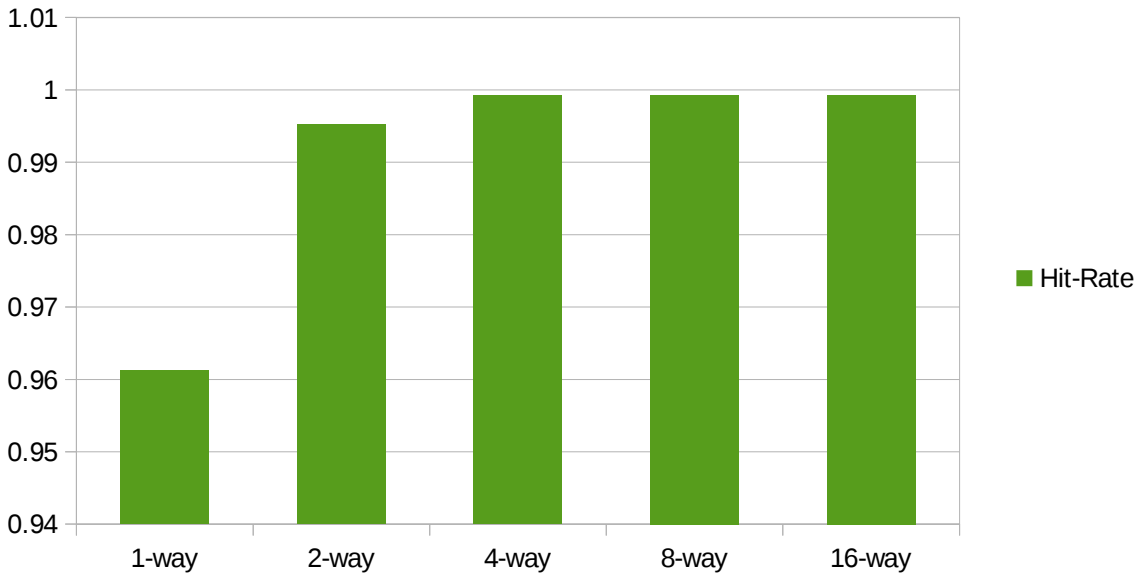
3. GO

	Accesses	Hits	Misses	Hit-Rate	Miss-Rate
1-way	36716742	35667787	1048955	0.971431153	0.0286
2-way	36716742	36302494	414248	0.988717735	0.0113
4-way	36716742	36398798	317944	0.991340626	0.0087
8-way	36716742	36465829	250913	0.993166251	0.0068
16-way	36716742	36496784	219958	0.994009327	0.006



4. FPPPP

	Accesses	Hits	Misses	Hit-Rate	Miss-Rate
1-way	156174476	150120857	6053619	0.961238103	0.0388
2-way	156174476	155425019	749457	0.995201156	0.0048
4-way	156174476	156053428	121048	0.999224918	0.0008
8-way	156174476	156063840	110636	0.999291587	0.0007
16-way	156174476	156065019	109457	0.999299136	0.0007



RESULTS:

As we can see from the above analysis, the number of misses and the miss rate reduces and the hit rate increases as the associativity is increased. This is seen with keeping the total cache size the same.

The GCC and VPR benchmarks show the best results.

Also we see that, initially, increasing the associativity from 1-way to 2-way, shows a considerable amount of improvement in hit rate, but this improvement further decreases in its amplitude (although continues to improve).

Part III : Study of the effect of sets in caches

In order to do this study, let us again take the data cache which has base configuration as shown and is a 32KB cache:

-cache:dl1 dl1:1024:32:1:l

The sets are reduced gradually as shown below:

1024 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:1:l -cache:dl2 none -cache:il2 none

512 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:512:32:1:l -cache:dl2 none -cache:il2 none

256 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:256:32:1:l -cache:dl2 none -cache:il2 none

128 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:128:32:1:l -cache:dl2 none -cache:il2 none

64 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:64:32:1:l -cache:dl2 none -cache:il2 none

32 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:32:32:1:l -cache:dl2 none -cache:il2 none

16 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:16:32:1:l -cache:dl2 none -cache:il2 none

8 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:8:32:1:l -cache:dl2 none -cache:il2 none

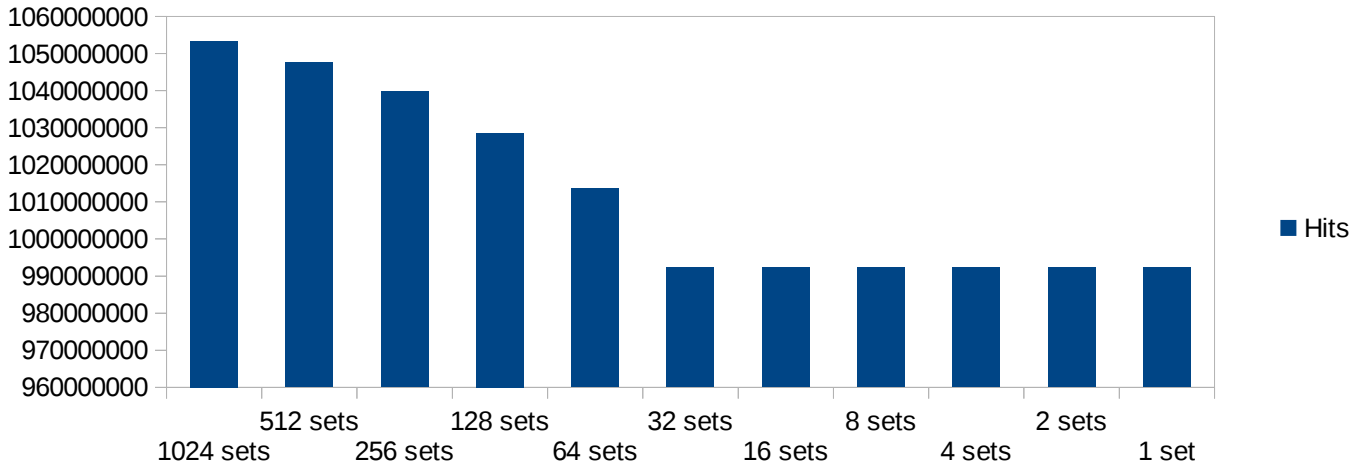
4 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:4:32:1:l -cache:dl2 none -cache:il2 none

2 sets:
simplesim-3.0/sim-cache -cache:dl1 dl1:2:32:1:l -cache:dl2 none -cache:il2 none

1 set:
simplesim-3.0/sim-cache -cache:dl1 dl1:1:32:1:l -cache:dl2 none -cache:il2 none

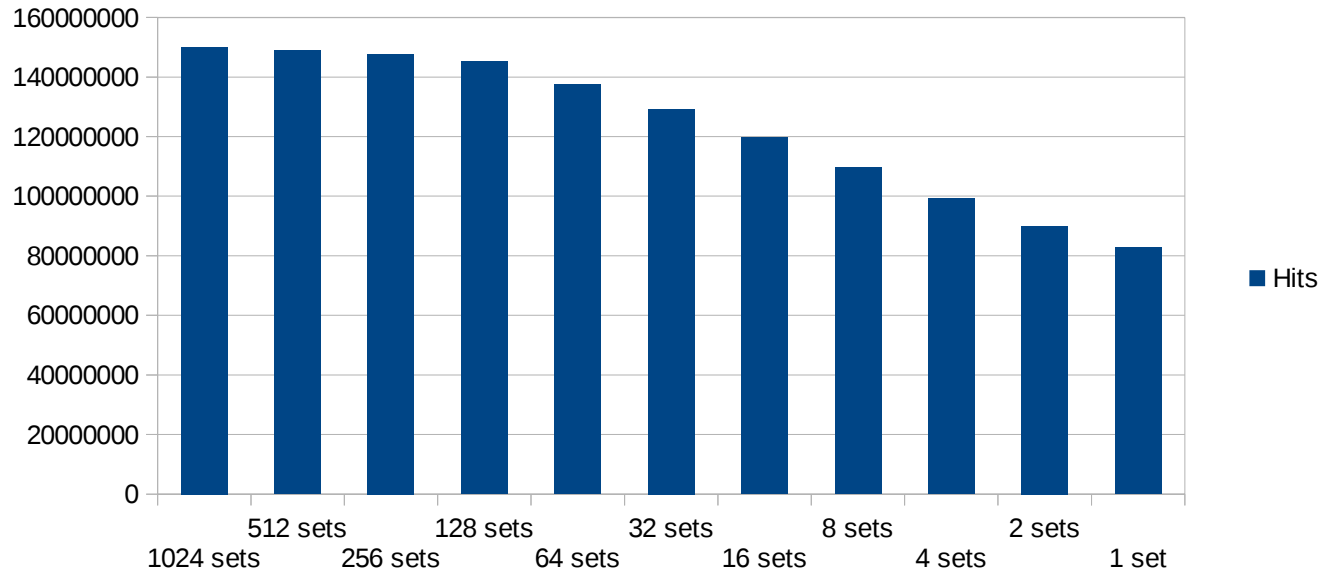
1. GCC

	Hits	Miss-rate
1024 sets	1053363221	0.0849
512 sets	1047550544	0.0899
256 sets	1039821800	0.0966
128 sets	1028672877	0.1063
64 sets	1013640301	0.1194
32 sets	992317899	0.1379
16 sets	992317899	0.1379
8 sets	992317899	0.1379
4 sets	992317899	0.1379
2 sets	992317899	0.1379
1 set	992317899	0.1379



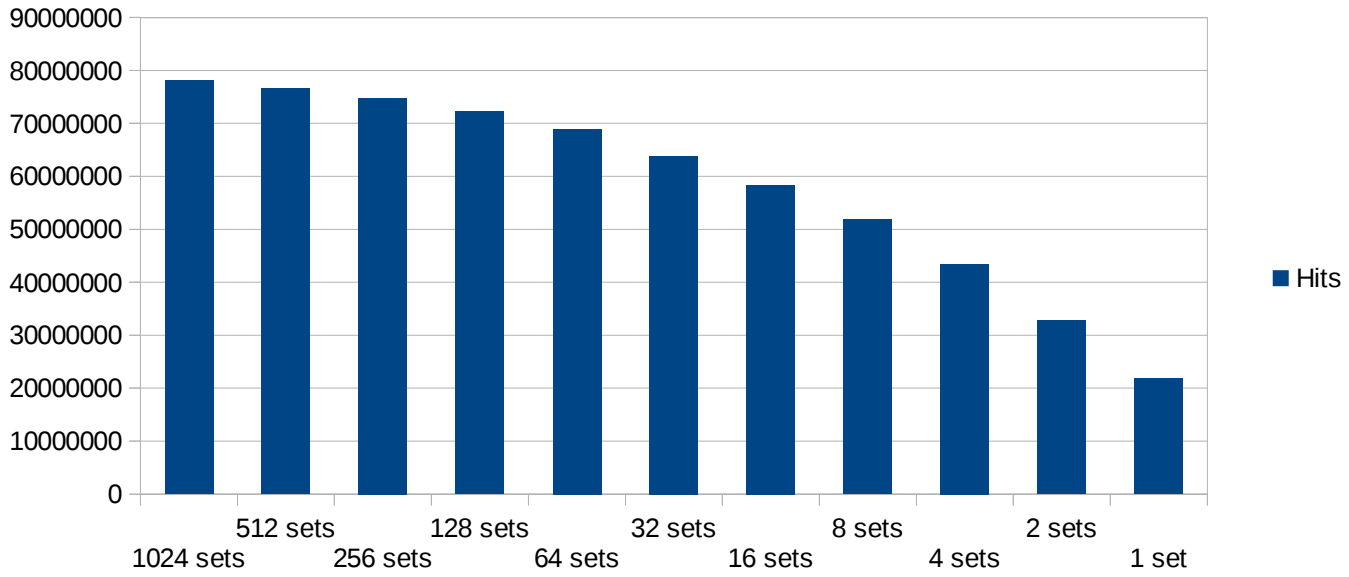
2. FPPPP

	Hits	Miss-rate
1024 sets	150120857	0.0388
512 sets	148905836	0.0465
256 sets	147735856	0.054
128 sets	145269647	0.0698
64 sets	137516374	0.1195
32 sets	129298275	0.1721
16 sets	119772830	0.2331
8 sets	109484753	0.299
4 sets	99130650	0.3653
2 sets	89983966	0.4238
1 set	82743089	0.4702



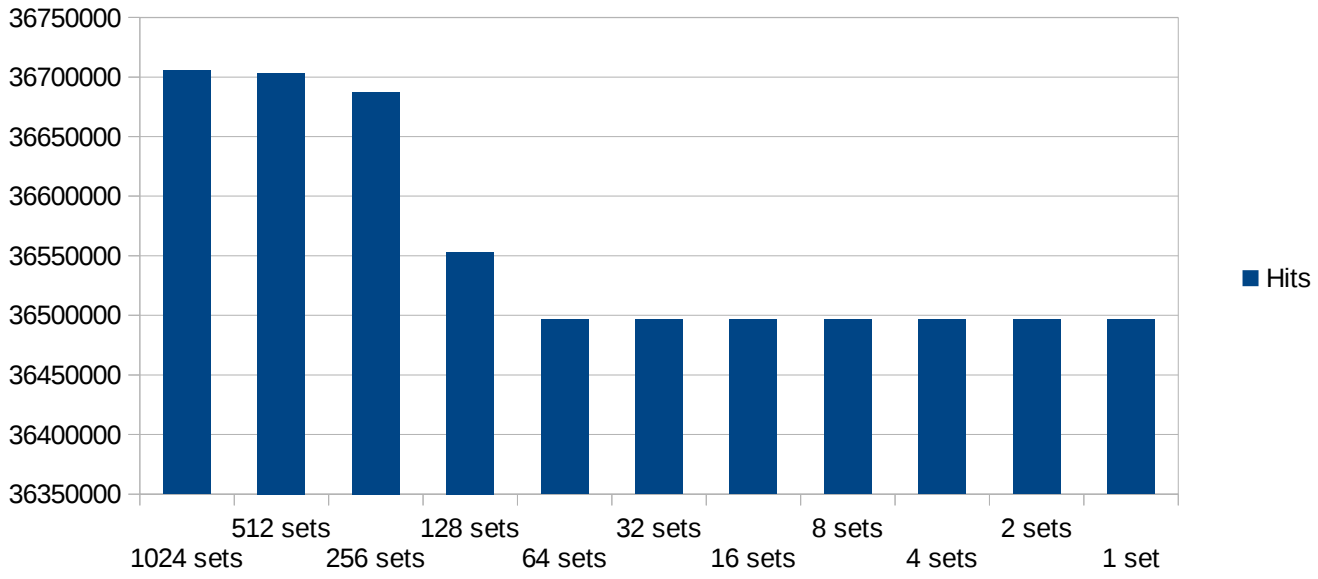
3. VPR

	Hits	Miss-rate
1024 sets	78146655	0.0336
512 sets	76596376	0.0528
256 sets	74802723	0.075
128 sets	72232983	0.1068
64 sets	68914226	0.1478
32 sets	63855402	0.2104
16 sets	58356972	0.2784
8 sets	51907370	0.3581
4 sets	43442521	0.4628
2 sets	32743365	0.5951
1 set	21832619	0.73

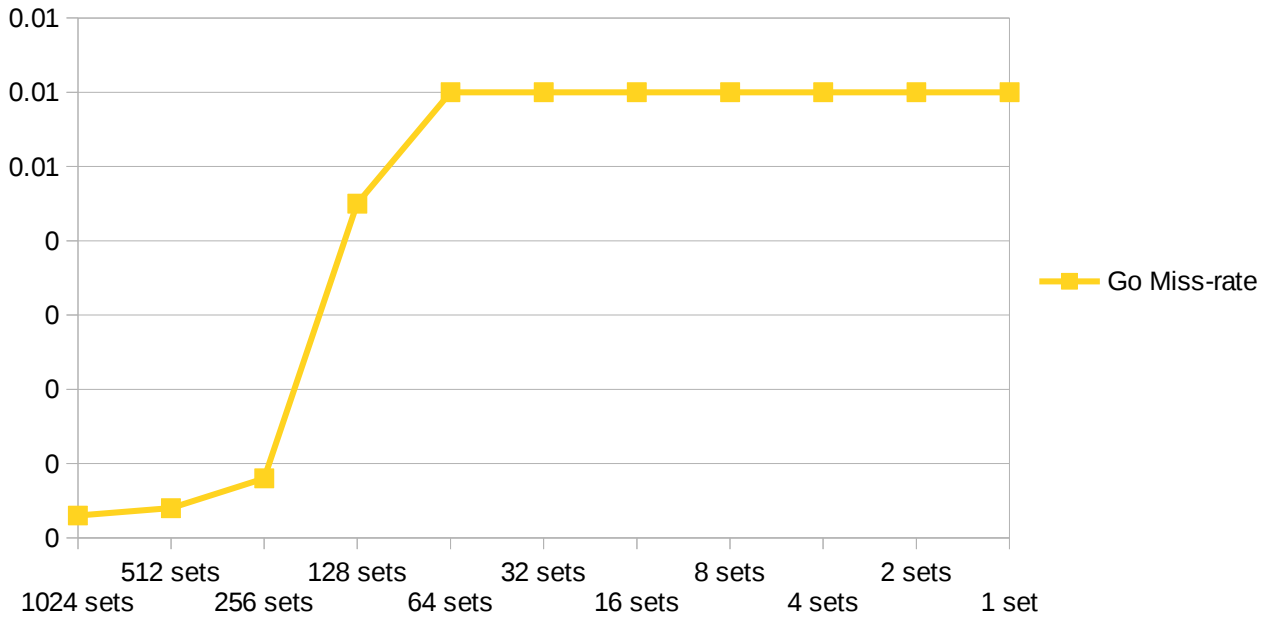


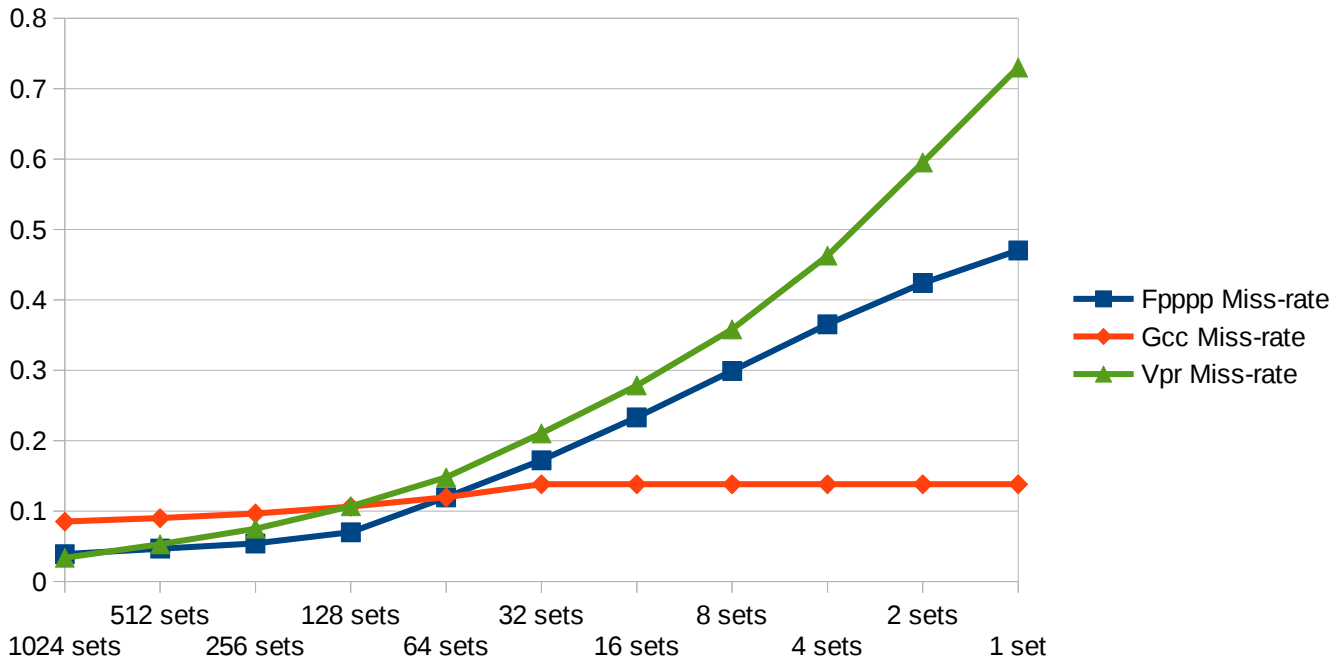
4. GO

	Hits	Miss-rate
1024 sets	36705244	0.0003
512 sets	36703390	0.0004
256 sets	36687415	0.0008
128 sets	36553134	0.0045
64 sets	36496784	0.006
32 sets	36496784	0.006
16 sets	36496784	0.006
8 sets	36496784	0.006
4 sets	36496784	0.006
2 sets	36496784	0.006
1 set	36496784	0.006



Following shows a plot of miss rates, as the number of sets are varied (go is plotted separated due to its low scale) :





RESULTS:

The above results show us that as we decrease the number of sets in a cache, the miss rate increases, almost exponentially. The best results can be seen for the VPR and FPPPP benchmarks.

Branch Predictor Study

The following is a study on Branch Predictors using SimpleScalar:

Here we would analyze the following types of branch predictions:

1. Not-Taken

Always predict the branch as not-taken.

2. Taken

Always predict the branch as taken.

3. Bimodal

This has 2 bit counters.
-bpred:bimod <size>
Here the size gives the number of entries in the branch target buffer.
I would test this for sizes 1k and 2k.

4. 2-Level

This is a 2-level adaptive predictor.
Here, there are 2 levels to the table.

This can be specified as:

-bpred:2lev <l1 size> <l2 size> <hist size>

l1 isze gives the size of the I level table
l2 gives the size of 2nd level table
hist size gives the width of the 1st level table
We test the standard config here, which is 1, 1k, 8 for l1 size, l2 size and hist size respectively

5. Combined

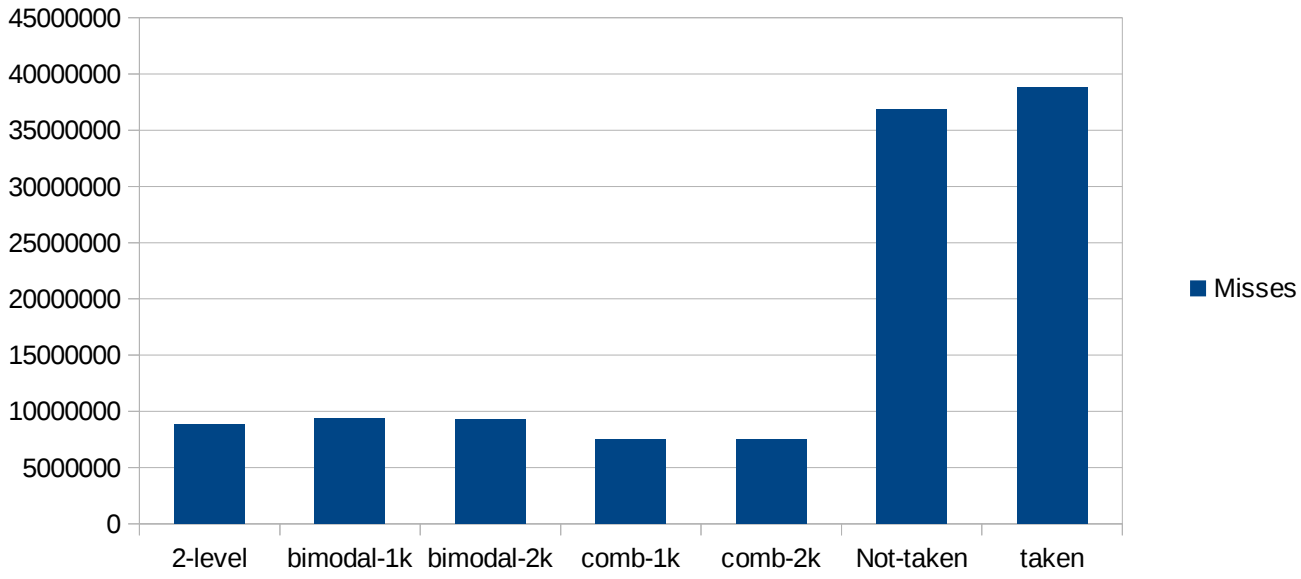
This combines a bimodal and 2 level predictor.

-bpred:comb <size>

We test this for 2 sizes. 1K and 2K.

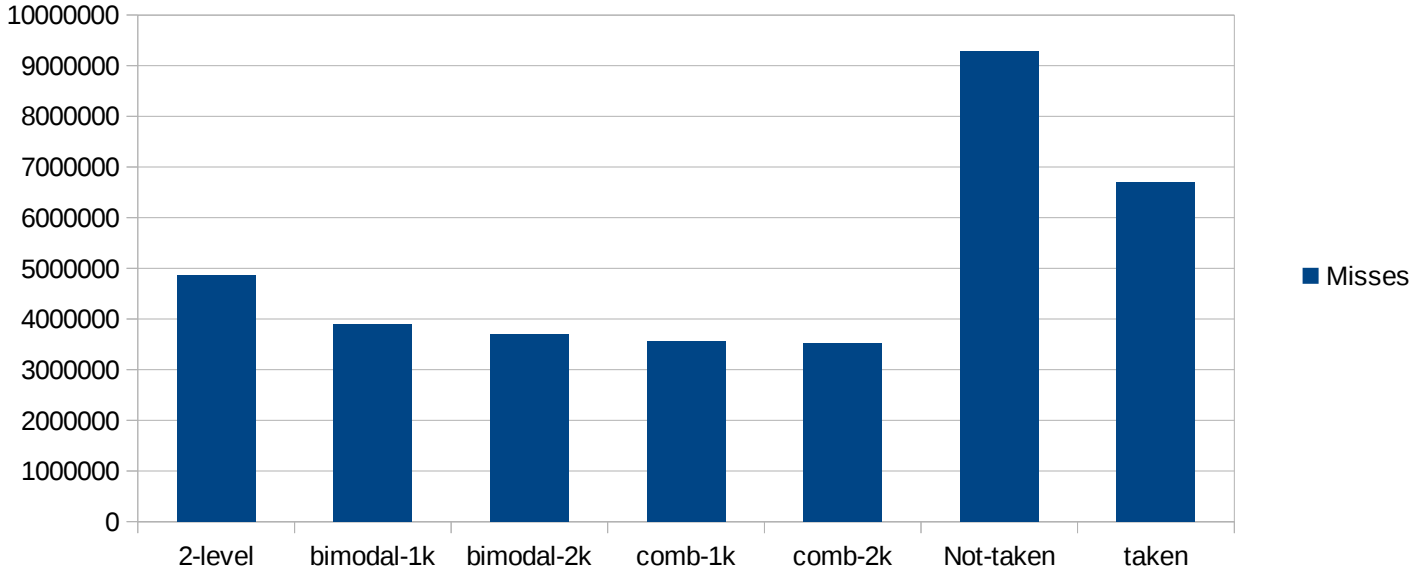
1. VPR

	Misses
2-level	8882408
bimodal-1k	9341390
bimodal-2k	9327331
comb-1k	7523816
comb-2k	7521993
Not-taken	36897108
taken	38818161



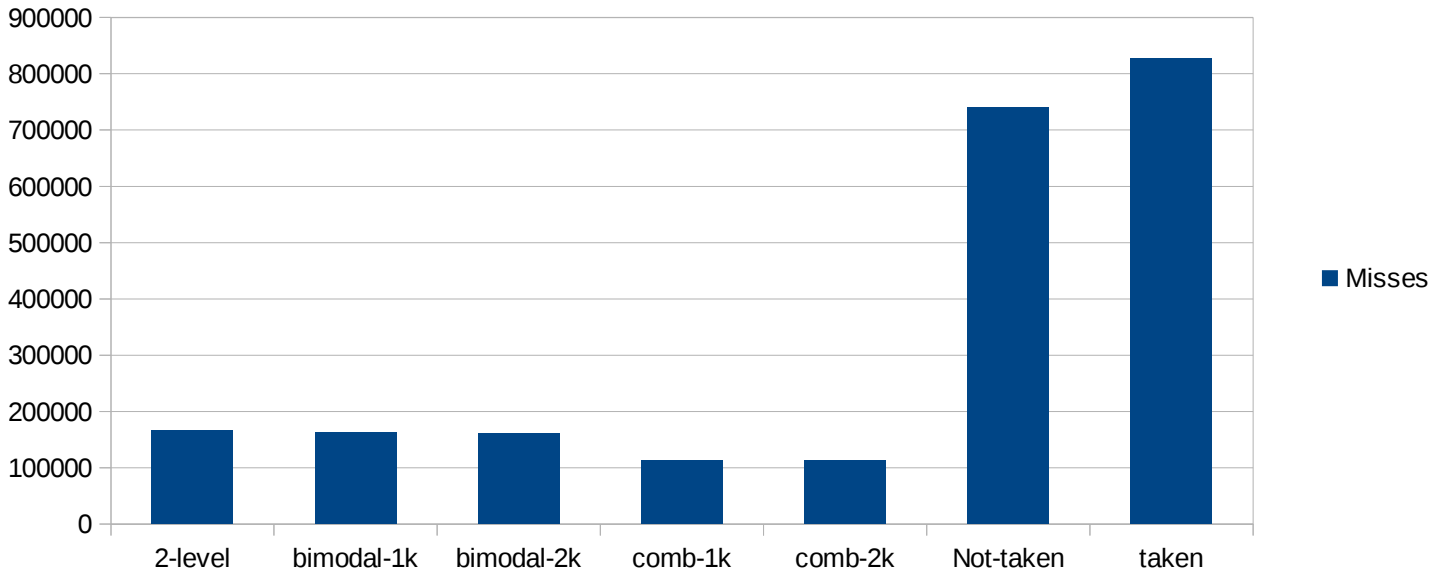
2.GO

	Misses
2-level	4874462
bimodal-1k	3889272
bimodal-2k	3705611
comb-1k	3567010
comb-2k	3523632
Not-taken	9282322
taken	6702456



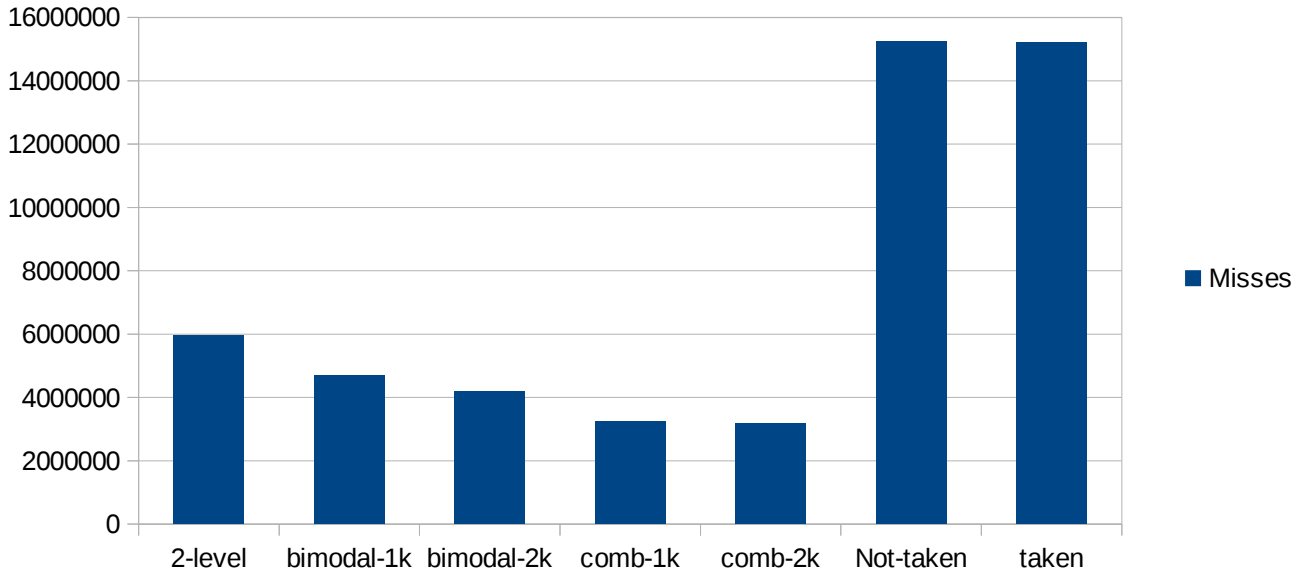
3. FPPPP

	Misses
2-level	167027
bimodal-1k	162034
bimodal-2k	160709
comb-1k	113791
comb-2k	113671
Not-taken	740056
taken	826614



4. GCC

	Misses
2-level	5952968
bimodal-1k	4695081
bimodal-2k	4183315
comb-1k	3247435
comb-2k	3188231
Not-taken	15235377
taken	15214282



RESULTS:

As we can see from the results above, in all the cases, taken and not-taken branch predictors show the maximum number of branch misses. This is obvious as there is no intelligent prediction taking place here. But we also know that hardware implementation this type of a predictor is easier.

This is followed by bimodal 1k and the bimodal 2k versions. This is just a 2 bit counter based predictor, giving better efficiency than taken and not taken predictors, but lesser than the others disused further.

The 2 level predictor gives lesser/comparable number of misses than bimodal predictor in case of VPR and FPPPP, but is more than bimodal in cases of GCC and GO. The reason is the number of instructions executed in the benchmark are not large enough to bring out the difference in the case of GCC and GO. But overall, 2-level predictors are an improvement over bimodal predictors. Although theres is a trade-off in terms of hardware overhead also.

The combined predictor shows us the best results with the least number of misses. There are slight improvements in the 2K version of this predictor compared to the 1K version. And over longer execution times, the 2K version turns out to be more advantageous.

REFRENCES:

www.simplescalar.com
The SimpleScalar Hacker's Guide by Todd Austin