PROJECT 4 : RESEARCH PROJECT
TOPIC : TRACE CACHE

Sekanipuram Srikanthan, Ramachandran
AND
Baskar, Hari Vignesh

Section 1: Project Description :

In this project, we add a real Trace Cache to the current 721 simulator to increase the fetch bandwidth per cycle by storing the dynamic traces of instruction. The Trace Cache is configurable, as in the size and associativity. The Trace Cache is terminated if "m" branches are reached, "n" instructions exist or once calls, returns or jump direct is detected. Moreover, to reduce redundancy, we eliminate traces with no branches since it can be fetched from the instruction cache itself. The indexing of the Trace cache is done with respect to the First Instruction of the trace Line. We use the Least Recently Used Policy to replace the victim block. We have added a victim trace cache and analyzed that as well.
The current Branch predictor can supply "m" branches per cycle, although training isn't the same way as we are getting the predictions. Since, the first instruction of the trace line is used to index to the Trace Cache, it is pertinent that all branches in that line are trained with respect to this PC as well. To resolve this problem, we have two methods on how we have changed the branch prediction, one is to avoid PC and use the branch History alone, like the **GAg branch predictor** and the other is to include the first instruction with which the Trace Cache is used to index in the branch queue as explained in further section. All of these configurations are tested and the results are shown in Section 5.

Section 2: Simulator accomplishments table :

Major microarchitecture feature and/or major simulator config	**Works With Perfect Branch Pred**	**Works With Real Branch Pred**	**comments / other columns as applicable**
Trace Cache with existing GShare multiple branch predictor	100M	100M	AStar and BZip ran
Trace Cache with Victim Cache enabled	100M	100M	AStar and BZip ran
Trace Cache with GAg multiple branch Predictor	100M	100M	AStar and BZip ran
Trace Cache with corrected logic for Multiple Branch Predictor GShare	100M	100M	Haven't tested the accuracy of predictor

Section 3: Microarchitecture description :

SubSection 1 : Trace Cache with Existing GShare Branch Predictor:

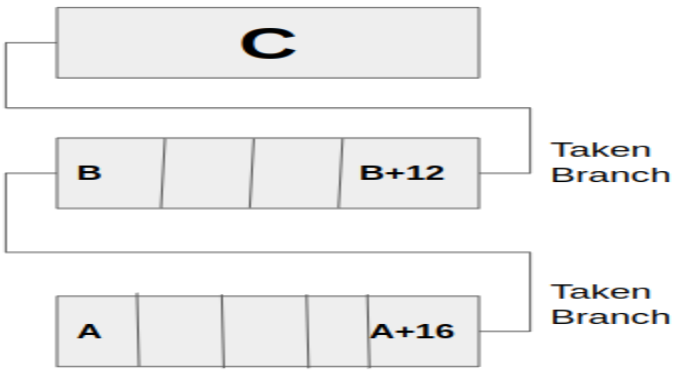
A configurable trace cache has been added to augment the instruction cache whose size and associativity can be varied and it has a total of “m” branches and “n” instructions (same as fetch width) and these parameters can be varied as well. The hit or miss in the Trace Cache is checked along with the instruction Cache in the fetch 1 stage. If there is a miss, the trace line starts its construction once the bundle reaches the fetch 2 stage and the dynamic instances of instructions are filled in the trace Cache until it reaches “m” branches or “n” instructions. To index it into the Trace Cache, we use the Trace Line’s first Instruction’s PC and the way tag is calculated is normal as in any cache. The LRU replacement policy is used.

SubSection 2 :GAg multiple Branch Predictor :

The existing multiple branch predictor is changed and the GAg predictor is included. This GAg branch predictor uses only the branch history to index into the table to update the 2 bit counters. Since it has lower contextual information, it doesn’t have a similar accuracy to GShare and has slightly lower accuracy as seen in the results section.

SubSection 3 : GShare Multiple Branch Predictor Correction with respect to Trace Cache

As explained in the description, apart from using the GAg branch predictor, we also tried to add the Trace Line’s first instruction PC into the branch Queue. The first instruction of a branch in a basic block might not be the same as the first instruction of the trace line and hence, if they are different, we add the trace line’s first instruction’s PC into the branch queue and train it along with training its own fetch bundle’s first instruction PC. The latter training is used if it misses in Trace Cache. For example, consider the below example where a new trace line starts with fetch PC A. For the branch with PC A+16, only A is included in the branch queue (already in 721 sim) but for branch with PC B+12, both PC’s A and B are included in the branch queue and are trained accordingly. Along with PC’s the corresponding BHR’s are also stored.



During commit, if there are two PC’s and two BHR’s (one for fetch bundle and another for Trace Cache first Instruction), both of them are used to index into the GShare Branch Predictor’s 2 bit counters.

Note : The flag might be set for even the first branch of the trace line if the trace line has a jump direct before. This is one of the corner cases.

Section 4: Simulator configurations :

FW	16	**FQ**	64	Trace Cache Policy fior victim blocks	LRU
DW	16	**AL Size**	1024	**RW**	16
IW	16	**LSQ**	512	**IQ**	1024

The benchmark used was for AStar for analysis purposes :

Graph 1 Configuration :

Trace Cache Size	8Kb	Max Branches	3	Num Instructions	16

Graph 2 Configuration :

Trace Cache Associativity	4	Max Branches	3	Num Instructions	16

Graph 3 Configuration :

Trace Cache Associativity	4	Trace Cache Size	4KB	Num Instructions	16

Graph 4 Configuration :

Trace Cache Associativity	4	Trace Cache Size	4KB	Max Branches 3	3

Graph 5 : Configuration :

This is the same as the previous expect that the branch Predictor is changed to GAg.

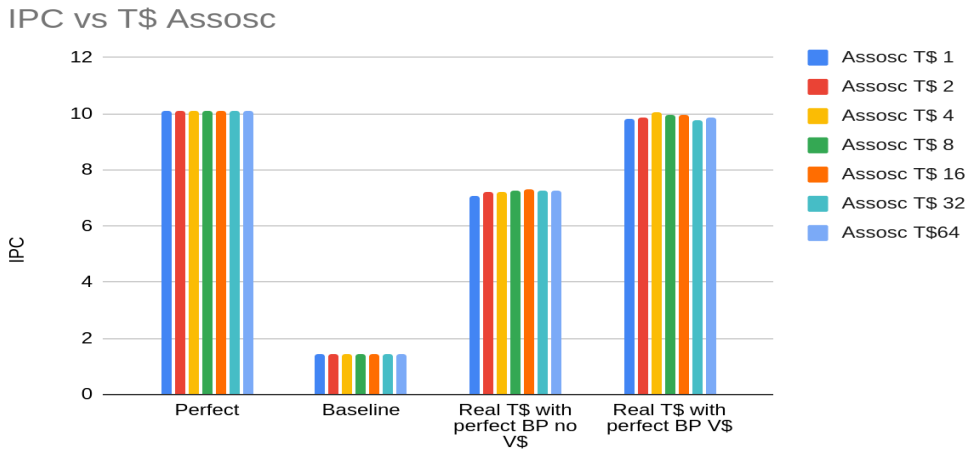
Table 1 :

Branch Prediction	Real	**Trace Cache Size**	1024
Data Cache	Perfect	**Associativity Trace Cache**	2

Section 5: Results and analysis :

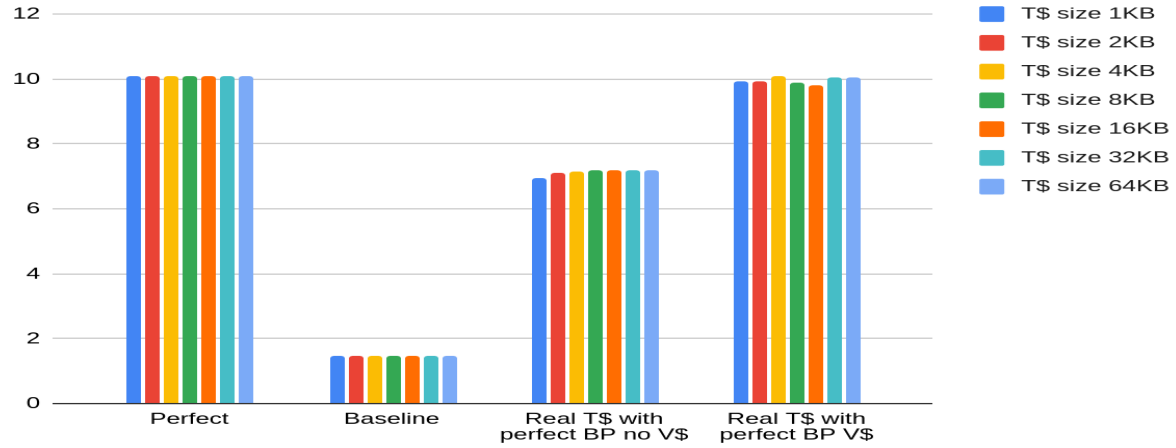
SubSection 1 : Trace Cache with Existing GShare Branch Predictor:

Graph 1 : Varying the associativity for the Trace Cache .



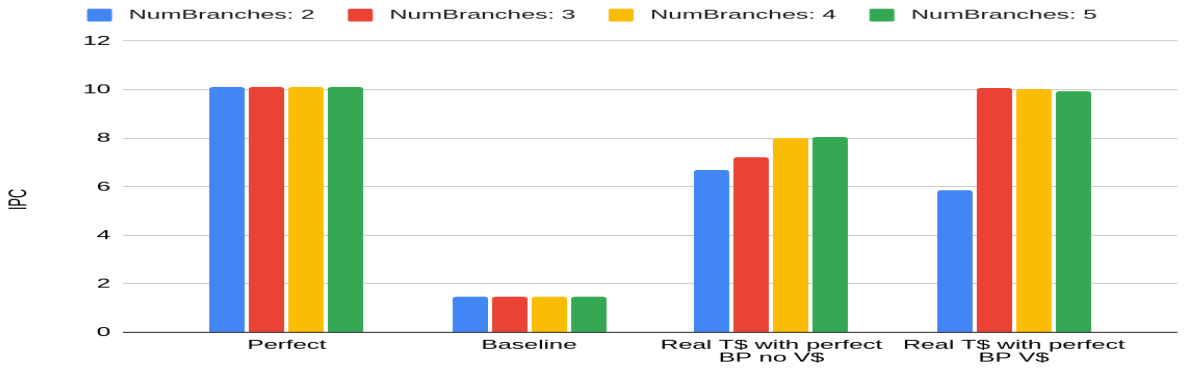
As it can be seen from the graph, varying the associativity does increase the IPC for real Trace Cache but does not cause very significant increase. VThere is a boost when the victim cache is included for perfect branch prediction at least.

Graph 2 : Varying the size of the Trace Cache :



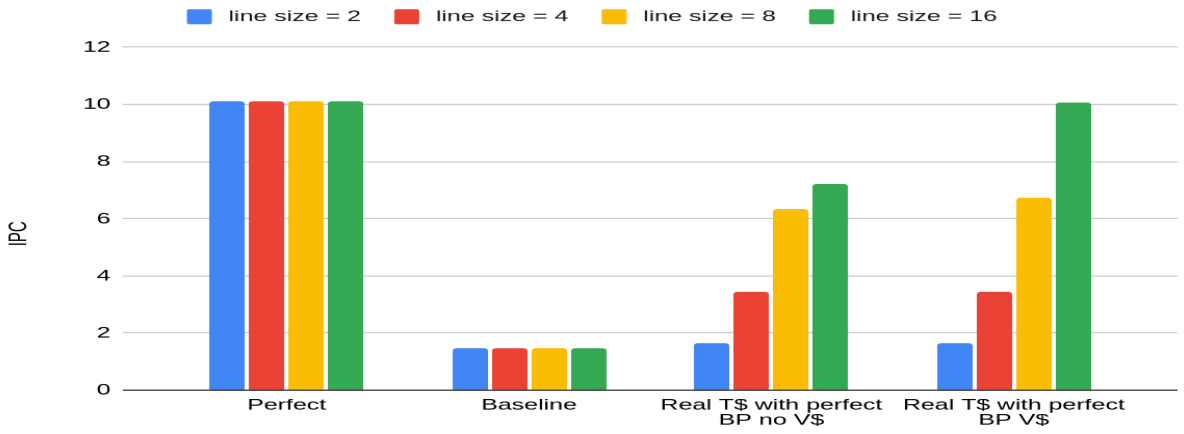
Similar to associativity, varying the size doesn't benefit the Trace Cache significantly.

Graph 3 : Varying the number of Branches per Trace Line :



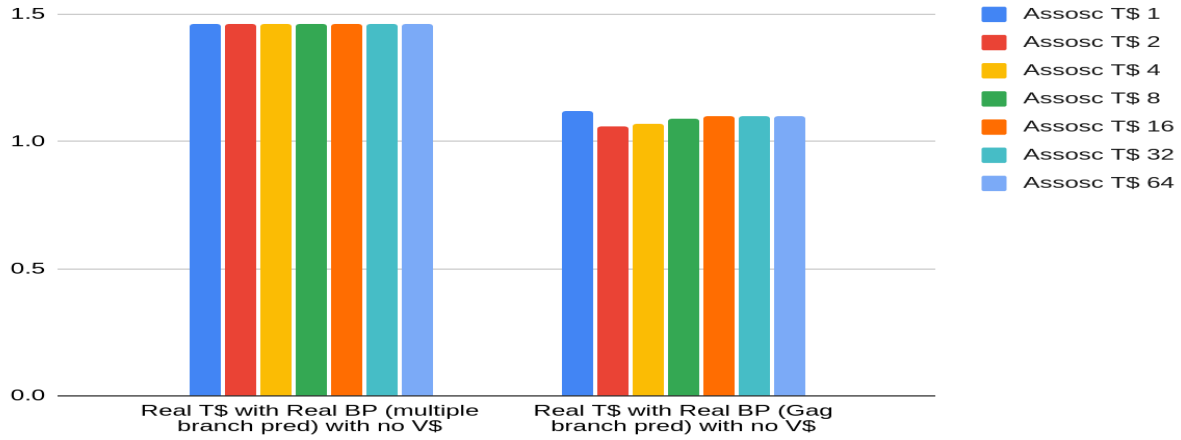
Number of Branches should be kept optimal, not too high nor too low.

Graph 4 : Varying the Fetch Width of the Trace Cache :



Trace Cache is highly effective only when it's fetch width is high since more dynamic instruction can be fetched.

SubSection 2 :GAg multiple Branch Predictor :



The GAg branch predictor performs low because it uses only the Branch History with less context information.

SubSection 3 : GShare Multiple Branch Predictor Correction with respect to Trace Cache

The Table below shows the results of the current GShare predictor in 721 sim after including the correction logic explained in Section 3 :

	A Star		**Bzip chicken**		**Leela**	
	Exisiting GShare	**Correction Logic**	**Exisiting GShare**	**Correction Logic**	**Exisiting GShare**	**Correction Logic**
Trace Cache hits	1685828	1774470	265237	265258	1215149	1218571
IPC	1.41	1.42	10.03	10.03	2.27	2.27
Increase in Trace Cache Hit rate		5.258		0.008		0.282

However, the IPC does not increase much, the hit rate has increased by 5% for AStar alone. The other hit rates aren't that much and we think the reason is because bzip is one of the benchmarks which has a lot of taken branches. SInce the existing GShare predictor is initialized to weakly taken by default, it itself is doing good.

Section 6: Future work :

1) The Branch Predictor correction logic explained in SubSection 3 should be corrected and analyzed further to check for performance bugs. Also, further multiple branch predictors should be explored as it's clear that Trace Cache highly depends on the accuracy of the Branch Prediction.