Lazy BTB: Reduce BTB Energy Consumption Using Dynamic Profiling

Yen-Jen Chang
Dept. of Computer Science
National ChungHsing University, Taiwan

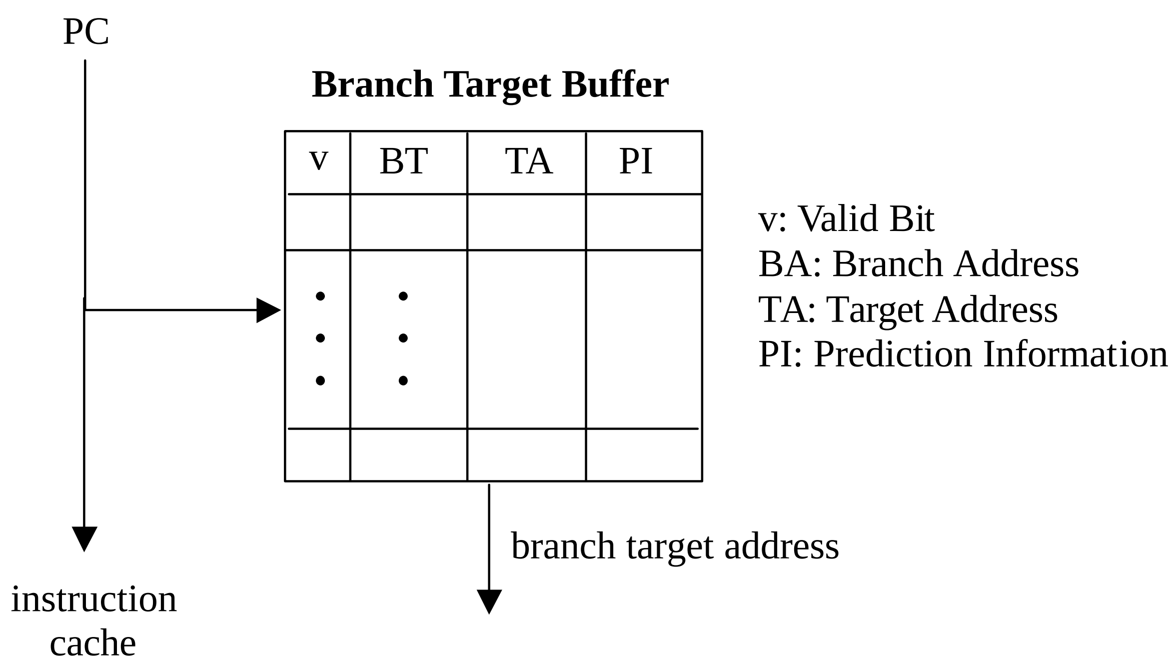
Outline

- Introduction
- Traditional BTB
- Lazy BTB
- Experimental Results
- Conclusions

1. Introduction

- The *branch target buffer* (BTB) is an essential component to the high performance processors with immunity from control hazard.
- Due to the high frequency of lookup, however, the energy dissipated in the BTB is usually considerable. For example, the Pentium Pro consumes about ***5%*** of the total processor energy in the equipped 512-entry BTB.
- In this paper, we propose an alternative BTB design, called ***lazy BTB***, to reduce the BTB energy consumption by filtering out the redundant lookups.

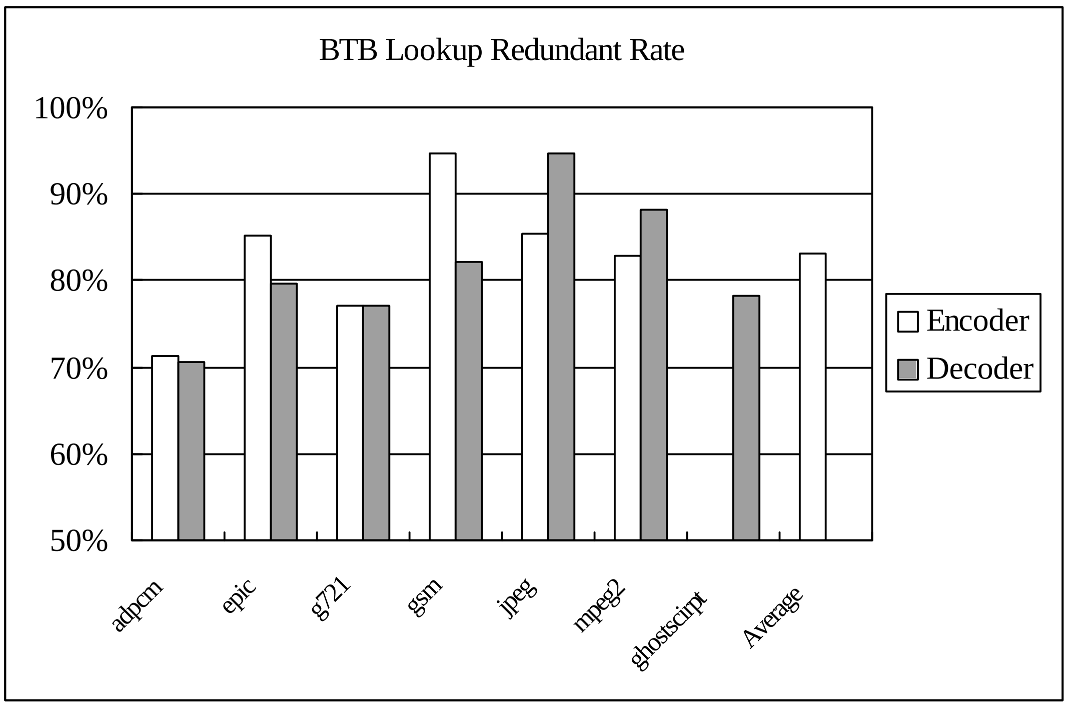
2. Traditional BTB



- In the traditional BTB lookup scheme, because the fetch engine has no sufficient information to distinguish the branch instructions, the BTB has to be looked up every instruction fetch.

Characteristics of the BTB Lookups

- Because the BTB lookup is necessary only for the branch instructions, in the traditional BTB an overwhelming majority of the lookups are redundant.
- Measured from *MediaBench*, the BTB lookup redundant rate is around ***83%*** on average.



3. Lazy BTB

- Unlike the conventional BTB, we propose an alternative BTB design, called ***lazy BTB***, which aims to reduce the number of redundant BTB lookups.

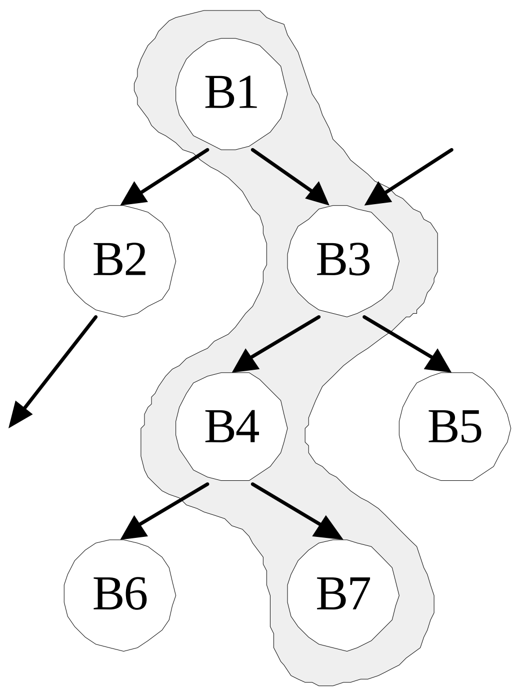
- The key idea behind our design is to look up the BTB only when the instruction is likely to be a taken branch.

- The lazy BTB design relies on the profiled taken trace from previous runs to skip the BTB lookup. A key issue in the realization of our design is how to profile the taken trace during program execution.

Basic Block vs. Taken Trace

- In contrast to the basic block, we define a ***taken trace*** as the instruction stream between the two consecutive taken branches. It can reflect the dynamic behavior of a program.

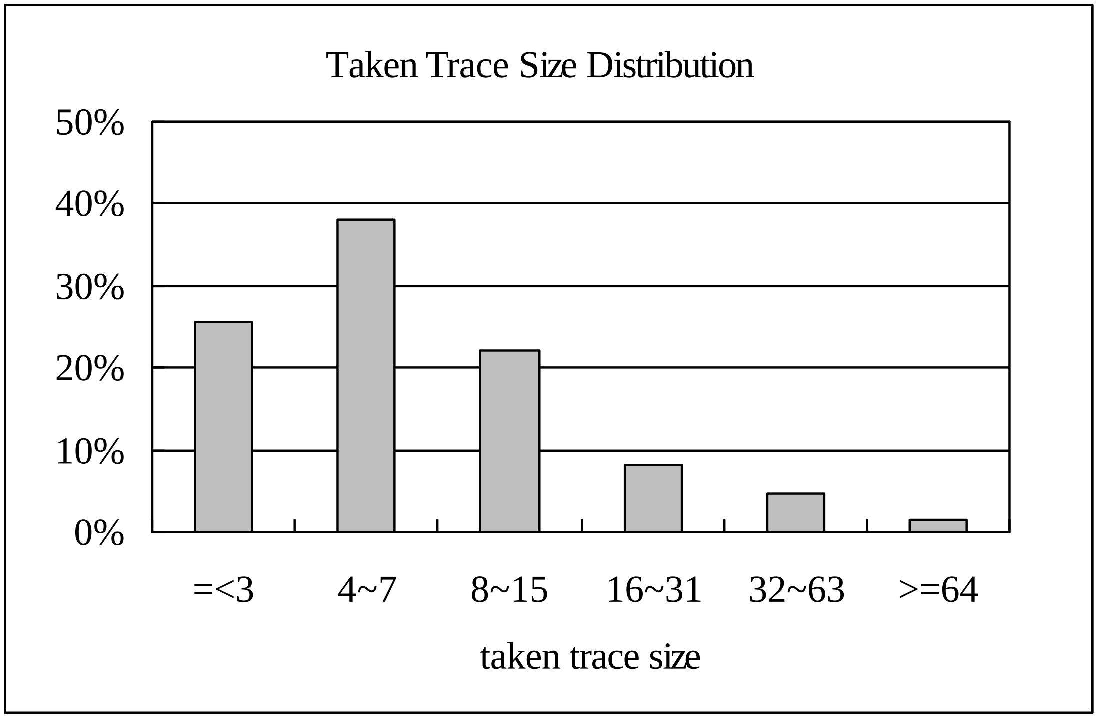
- A taken trace, by definition, contains more than one basic block.



Hardware Augmentations

1) The conventional BTB has to be augmented with an extra field for each entry, called ***taken trace size (TTS)*** field, which is used to record the size of the next taken trace.

For the best tradeoff between the energy efficiency and hardware cost, the TTS field width is fixed at 6-bit throughout this paper.



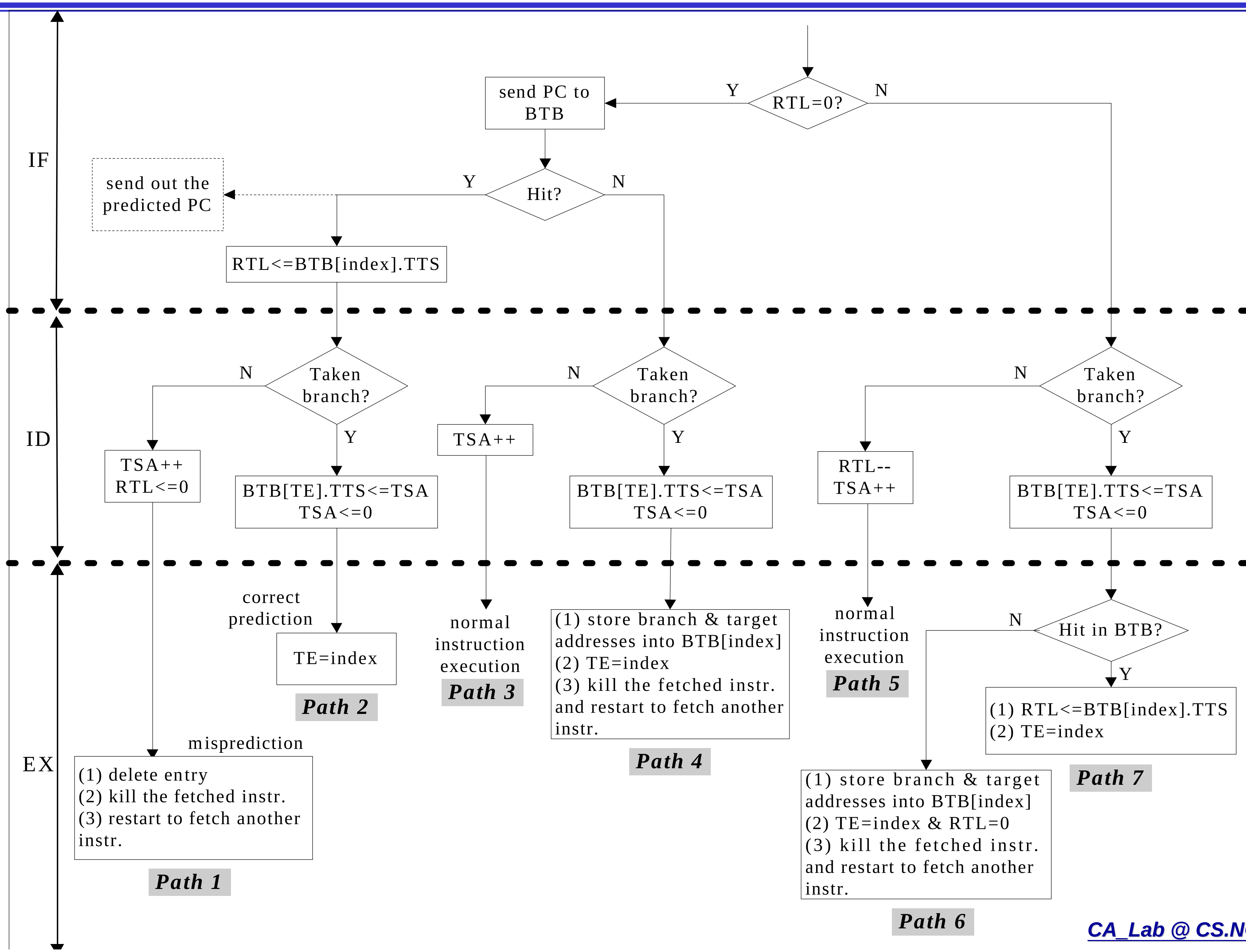
Hardware Augmentations

2) We need a counter, called ***remainder trace length* (RTL)**, to indicate whether the currently fetched instruction locates within a taken trace or not.

3) Another counter, called ***trace size accumulator* (TSA)**, is needed to accumulate the taken trace size during program execution.

4) A temporal register, called ***target entry* (TE)**, is needed to remember the index of the previous hit/allocated BTB entry during program execution.

Dynamic Taken Trace Profiling



The seven possible paths in the lazy BTB scheme.

Possible Paths	BTB Lookup	Hit/Miss	Prediction	Actual Branch	BTB Looup in EX	Penalty Cycles
Path 1	Y	Hit	taken	not taken	-	2
Path 2	Y	Hit	taken	taken	-	0
Path 3	Y	Miss	-	not taken	-	0
Path 4	Y	Miss	-	taken	-	2
Path 5	-	-	-	not taken	-	0
Path 6	-	-	-	taken	Y/Hit	3/4
Path 7	-	-	-	taken	Y/Miss	1/2

4. Experimental Results

- We use SimpleScalar toolset to model a baseline processor that closely resembles **StrongARM** processor.

Processor Configuration	
Issue width	1 intr. per cycle
Intruction window	2-RUU, 2-LSQ
Function units	1 Int ALU, 1 Int Mult/Div
	1 FP ALU, 1 FP Mult/Div
L1 instruction cache	16KB, 32-way, 32B blocks
L1 data cache	16KB, 32-way, 32B blocks
TLB (iTLB & dTLB)	128-entry, 4-way
Branch perdictor	2-Level 1K-entry
BTB	512-entry, 4-way
Return address stack	8-entry
Penalty Parameters	
L1 hit latency	1 cycle
Branch misprediction	2 cycles
Memory access latency	8 cycles for the first chunk
	2 cycles for the rest of a burst access
TLB miss penalty	30 cycles

Path Distributions

- The path distributions have a strong impact on the energy efficiency of the lazy BTB.

The large percentage of path 5 is preferred.

Benchmark	path 1~4	path 5	path 6~7
adpcm_en	37.53%	59.41%	3.06%
adpcm_de	32.83%	64.63%	2.54%
epic_en	13.89%	85.68%	0.43%
epic_de	15.95%	83.39%	0.66%
g721_en	18.00%	81.11%	0.89%
g721_de	17.72%	81.42%	0.86%
gsm_en	15.00%	84.45%	0.56%
gsm_de	11.35%	88.50%	0.15%
jpeg_en	14.57%	84.92%	0.51%
jpeg_de	14.44%	85.07%	0.49%
mpeg2_en	30.79%	66.90%	2.31%
mpeg2_de	17.37%	81.81%	0.82%
ghostscirpt	13.17%	86.48%	0.35%
Average	19.43%	79.52%	1.05%

Total Energy Consumption of BTB Lookups

- The metric used to evaluate the energy efficiency is the simple total energy consumption of BTB lookups.

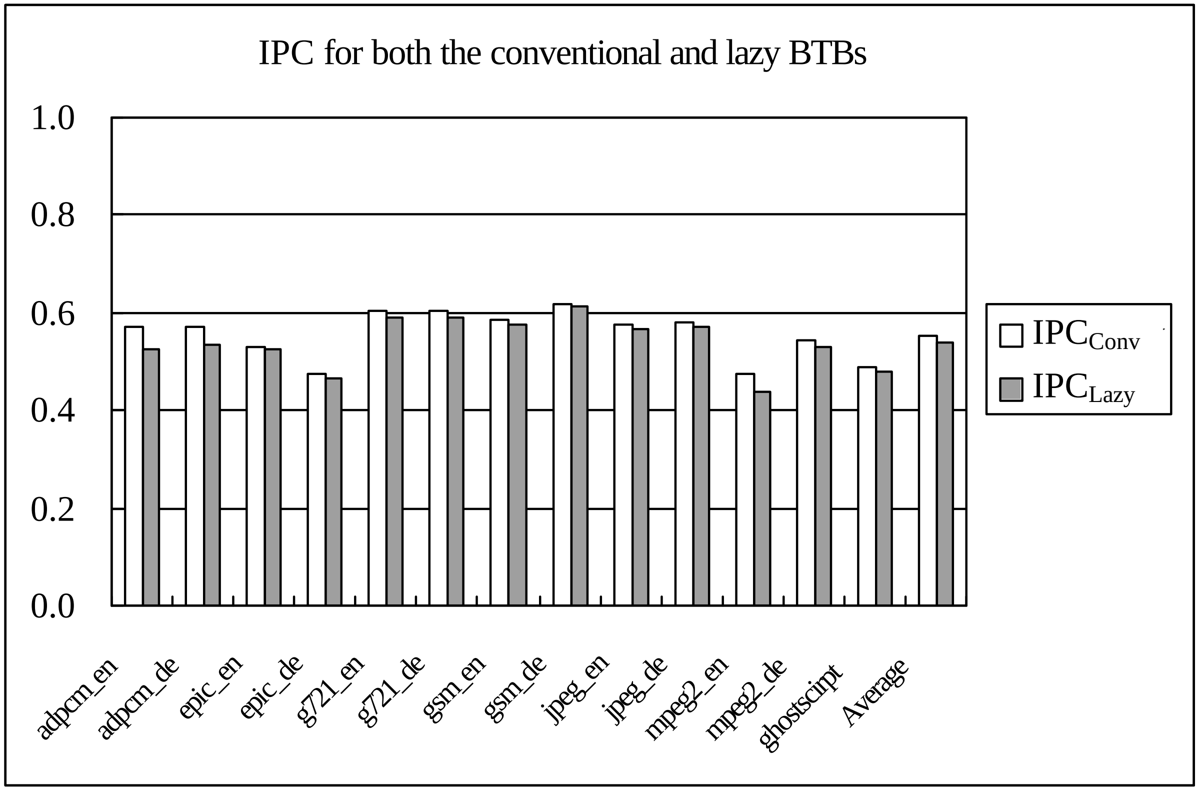
	BTB_{Conv}	BTB_{Lazy}	Reduction
adpcm_en	290.8	126.9	56.35%
adpcm_de	239.2	90.7	62.09%
epic_en	25.3	3.7	85.25%
epic_de	3.2	0.6	82.73%
g721_en	131.9	26.1	80.22%
g721_de	128.5	25.0	80.56%
gsm_en	896.6	144.4	83.90%
gsm_de	305.7	35.6	88.35%
jpeg_en	48.6	7.6	84.41%
jpeg_de	12.3	1.9	84.58%
mpeg2_en	544.4	192.8	64.59%
mpeg2_de	82.2	15.6	80.99%
ghostscirpt	557.1	77.3	86.13%
Average	251.2	57.6	77.09%

By filtering out most redundant BTB lookups, the lazy BTB can reduce the total energy consumption of BTB lookups by 56%~88% for MediaBench.

Performance Impact

- Compared to the conventional BTB, only the paths 6 and 7 result in the extra penalty cycles. The paths 6 and 7 are, therefore, referred to as *unfavorable path*.

Our design results in roughly 1.7% IPC degradation on average.



5. Conclusions

- By using the developed dynamic taken trace profiling technique, the lazy BTB can achieve the goal of ***one BTB lookup per taken trace*** instead of one BTB lookup per basic block.

- The results show that without noticeable performance difference from the conventional BTB, our design can reduce the total energy dissipated in BTB lookups up to ***88%*** for the *MediaBench* applications.

Thank You

Q & A