18-344 Recitation 4

Lab2 - Memory Hierarchy

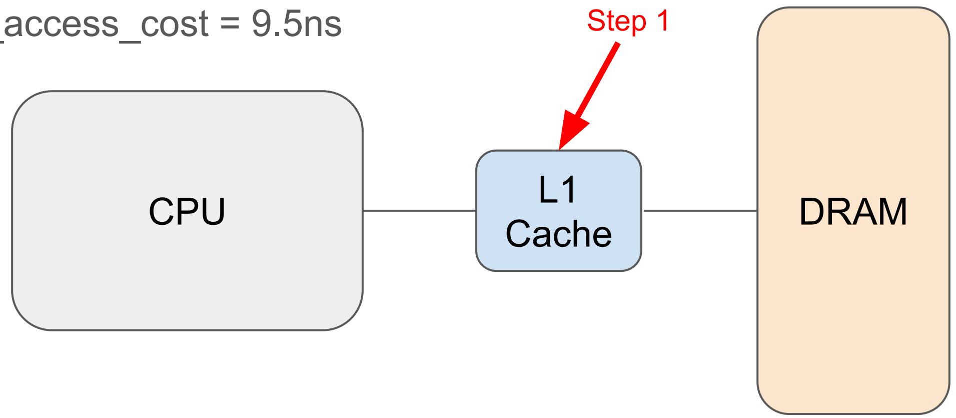
Step 1: Implement a Set-Associative Cache

- Simulate a set-associative cache, which takes in memory loads/stores from Pin, and updates the cache state after each memory access.
- The cache should have three configurables parameters:
  - Cache size, Block size, and Associativity
- You will sweep across reasonable values for these parameters.
- For each cache configuration, you must profile the SPEC workloads to get
  - total accesses, hit rate, and miss rate
- You will have to implement the required file I/O in the pintool yourself

Step 1: Implement a Set-Associative Cache Cont.

- You will then use Destiny to generate the access/miss latencies for each cache configuration.
- Using this data, you can now calculate the AMAT per cache configuration. AMAT (1-level cache) Formula:
    - AMAT = L1_hit_rate * L1_access_cost + L1_miss_rate * (L1_miss_cost + DRAM_access_cost)
    - Use DRAM_access_cost = 9.5ns

Example AMAT calculation in Lecture 7

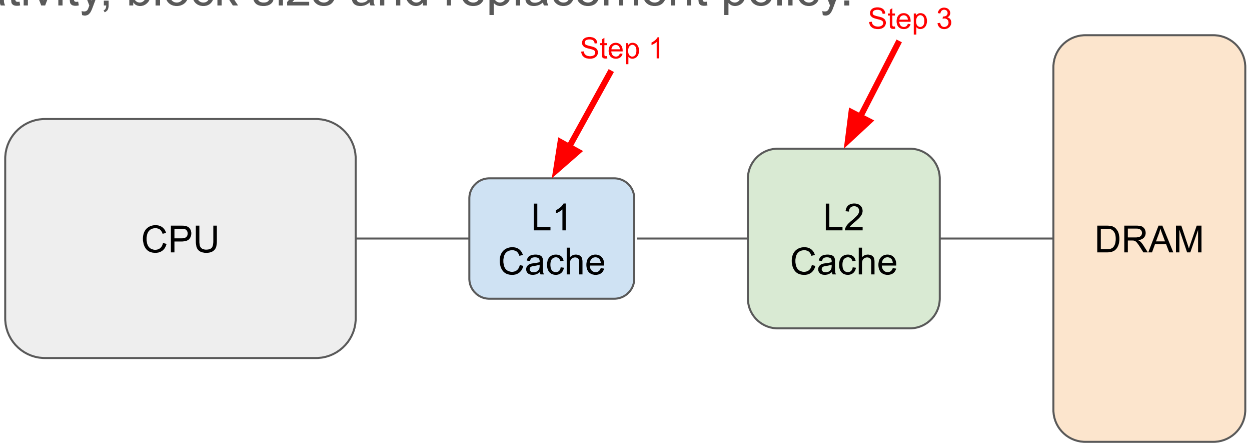


Step 2: Implement cache replacement policies

- Implement the replacement policies from class, and any other replacement policy that you think might improve performance.
  - random, LRU, and bit-PLRU
- Select three different reasonable values for cache size, associativity and block size, and compare the miss rates for each replacement policies.
- Comment on the absence or presence of a trend in the plot

Step 3: Implement a two-level cache hierarchy

- Extend the parametrized set-associative cache structure written in Step 1 to implement a two-level cache hierarchy
- Each level of the cache hierarchy can take a variable value for the cache size, associativity, block size and replacement policy.



Step 4: Design Space Exploration Tool

- Write a design space iteration tool
- Explore the design space created by the different configurations for the cache hierarchy. A configuration can be established as:
  - {(L1 associativity, L1 block size, L1 size, L1 replacement),
    (L2 associativity, L2 block size, L2 size, L2 replacement) }
- Your design space iteration tool should search the space of configuration tuples to find the optimal configuration
- You should evaluate each cache's leakage power, dynamic access power, and access latency using the Destiny memory modeling tool.
- Using these numbers from Destiny, you should report a configuration's performance as the Average Memory Access Time (AMAT)

Step 4: Design Space Exploration Tool Cont.

Again: Example AMAT calculation in Lecture 7

- AMAT (2-level cache) = L1_hit_rate * L1_access_cost + L1_miss_rate * (L1_miss_cost + L2_hit_rate * L2_access_cost + L2_miss_rate * (L2_miss_cost + DRAM_access_cost))
    - Still use DRAM_access_cost = 9.5ns
- Total Storage Budget: Your system has a total budget of 5MB of cache that you can split across the layers of cache in your system (only search within this budget)
- Minimization goal: Your design space iteration tool should minimize area and power if comparable configurations are equal in their performance; always opt for the lower area or lower power configuration.

Step 4: Design Space Exploration Tool Cont.

- Implementability: You must argue in your write-up that the cache hierarchy that you have proposed is implementable using supporting evidence from the Destiny tool.
  - Is the power consumed while doing reads and writes reasonable?
  - Is the tag storage overhead reasonable?
  - Are access latencies reasonable?
  - Your argument should consider that L1 accesses should be only a few cycles, meaning L1 access latency (especially for reads) is critical.

Evaluation & Submission Artifacts

For full details refer to lab2_handout.txt

- You will submit your cache simulator, your design space search scripts, and a coherent README that is sufficiently detailed for your TA to run your simulator on other test workloads.
- You should include a quantitative summary of your design space search, including a design space plot including a Pareto Frontier cataloging the set of design points that you considered in your design space search.
- You should include a bar plot for a set of Pareto optimal cache configurations that shows the AMAT for each of the SPEC2017 benchmarks that you run.
- You should include a sensitivity study that shows how performance varies with the important dimensions of your cache

About Averages

- Different types of averages

$$ArithmeticMean(a_1, a_2, a_3, \ldots, a_N) = \frac{\sum_i^N a_i}{N}$$

$$GeometricMean(a_1, a_2, a_3, \ldots, a_N) = \sqrt[N]{\prod_i^N a_i}$$

$$HarmonicMean(a_1, a_2, a_3, \ldots, a_N) = \frac{N}{\sum_i^N \frac{1}{a_i}}$$

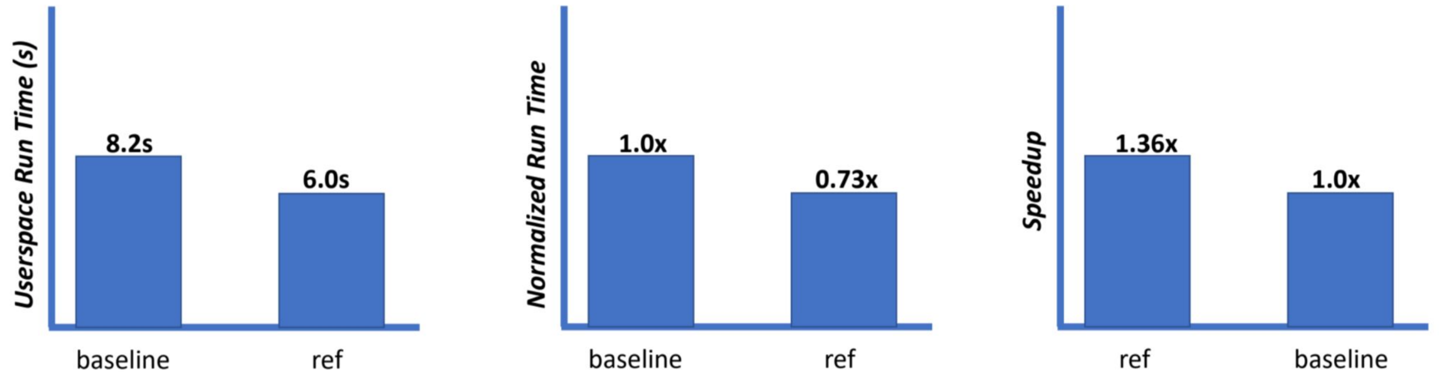
- Good for different evaluations
  - Time → Arithmetic Mean
  - Speedup → Geometric Mean
  - Rates → Harmonic Mean
- Unweighted Average
  - No benchmark is more important or prevalent than any other in the wild despite differences in time during tests.
- Weighted Average
  - Runtime of each test indicates its prevalence/importance in the wild. Tests that run longer count for more

About Baselines

- Not going to use Oracle Sun Fire V490 as your baseline unless you’re submitting your results to SPEC
- Choose a good, simple baseline to compare to
- Select a highly optimized baseline
  - Avoid “Straw Man” comparisons
- Select a comparable baseline
  - Avoid “apples-to-oranges” comparisons

About Graphs

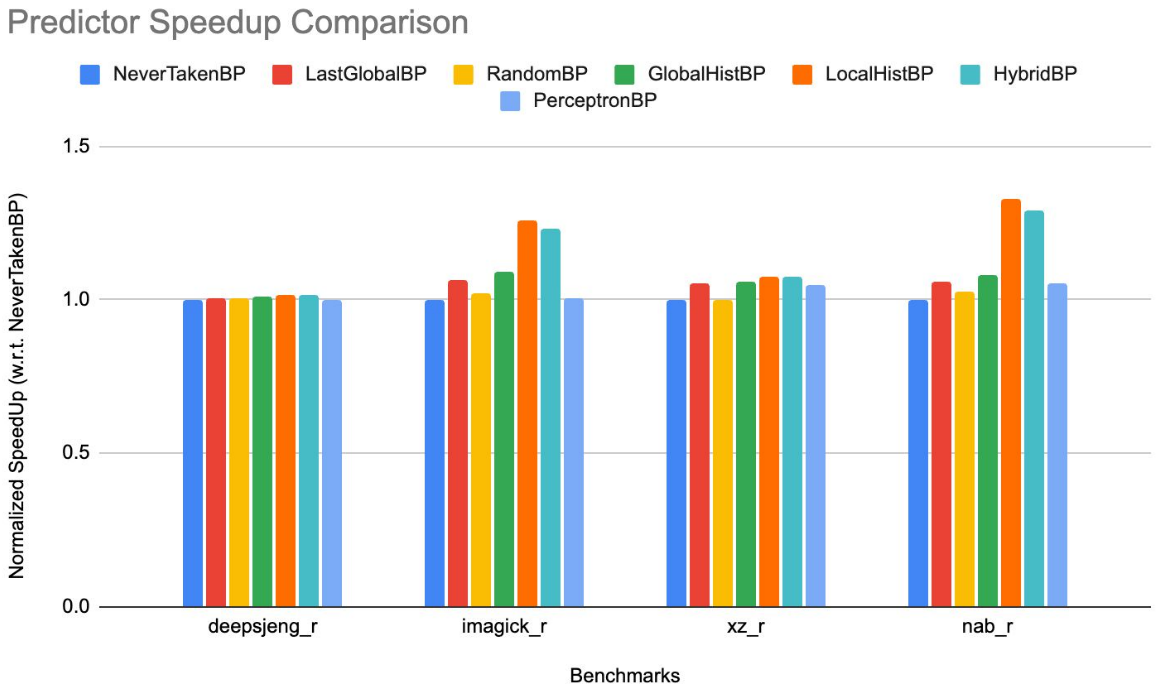
- Think about what you are trying to show



These are equivalent results, with different presentations

About Graphs Cont.

- Example of Branch Predictor Speedup Comparison



GDB with PinTool!

Step 1: Start GDB with PinBin

```
$ gdb pinbin
```

```
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from
/afs/ece.cmu.edu/class/ece344/opt/pin/pin-3.18-98332-gaebd7b1e6-gcc-linux/intel64/bin/pinbin...(
no debugging symbols found)...done.
(gdb)
```

Step 2: start your Pintool in another terminal

(make sure it’s on the same machine)

```
$ pin -pause_tool 10 -t obj-intel64/pintool.so -o stats.txt
-- cmd
```

```
Pausing for 10 seconds to attach to process with pid [18344]
To load the debug info to gdb use:
*****************************************************************
set sysroot /not/existing/dir
file
add-symbol-file /path/to/obj-intel64/bp_pintool.so 0x7f95279d70a0
*****************************************************************
```

Step 3: Add Symbol File

```
(gdb) attach 18344

(gdb) add-symbol-file /path/to/obj-intel64/bp_pintool.so
0x7f95279d70a0

add symbol table from file
"/path/to/obj-intel64/bp_pintool.so" at
    .text_addr = 0x7f7eaa8f60a0
    .data_addr = 0x7f7eaacf1880
    .bss_addr = 0x7f7eaacf2240
(y or n) y
Reading symbols from
/path/to/obj-intel64/bp_pintool.so...(no debugging symbols
found)...done.
```