Jeffrey Lau, Giovanni Lupo, Zacharia Hammad
11/25/2024
ECEC 412
Prof. Anup Das
Project 4

ECE 412 Project 4

SHiP: 531.deepjseng_r_llc.mem_trace

cache_size	assoc	Hit rate
128	4	66.181830%
256	4	81.822419%
512	8	93.004667%
1024	16	95.412173%
2048	16	95.824792%
2048	4	95.474110%
128	16	72.851181%

LRU: 531.deepjseng_r_llc.mem_trace

cache_size	assoc	Hit rate
128	4	76.939192%
256	4	88.972509%
512	8	94.859542%
1024	16	95.751690%
2048	16	95.958330%
2048	4	95.906338%
128	16	78.541262%

LFU: 531.deepjseng_r_llc.mem_trace

cache_size	assoc	Hit rate
128	4	75.574815%

256	4	88.382205%
512	8	94.017041%
1024	16	94.542108%
2048	16	95.262287%
2048	4	95.820579%
128	16	72.702271%

SHiP: 541.leela_r_llc.mem_trace

cache_size	assoc	Hit rate
128	4	30.109947%
256	4	56.537185%
512	8	89.641995%
1024	16	98.501923%
2048	16	99.454722%
2048	4	98.884928%
128	16	30.182191%

LRU: 541.leela_r_llc.mem_trace

cache_size	assoc	Hit rate
128	4	42.554451%
256	4	70.743799%
512	8	94.443418%
1024	16	99.011727%
2048	16	99.627833%
2048	4	99.455119%
128	16	55.102674%

LFU: 541.leela_r_llc.mem_trace

cache_size	assoc	Hit rate
128	4	48.440187%
256	4	70.096169%
512	8	87.869778%
1024	16	94.520435%
2048	16	99.132343%
2048	4	99.136207%
128	16	51.064270%

SHiP: 548.exchange2_r_llc.mem_trace

cache_size	assoc	Hit rate
128	4	72.686601%
256	4	99.983818%
512	8	99.985065%
1024	16	99.985094%
2048	16	99.985094%
2048	4	99.985094%
128	16	99.984463%

LRU: 548.exchange2_r_llc.mem_trace

cache_size	assoc	Hit rate
128	4	99.926982%
256	4	99.984771%
512	8	99.985094%
1024	16	99.985094%
2048	16	99.985094%
2048	4	99.985094%
128	16	99.990552%

LFU: 548.exchange2_r_llc.mem_trace

cache_size	assoc	Hit rate
128	**4**	**72.732198%**
256	**4**	**99.984948%**
512	**8**	**99.985094%**
1024	**16**	**99.985094%**
2048	**16**	**99.985094%**
2048	**4**	**99.985094%**
128	**16**	**99.990552%**

[Cache Set]

| Block A (Sig:S1, PC:2) | Block B (Sig:S2, PC:0) | Block C (Sig:S3, PC:3) | Block D (Sig:S4, PC:1) |

↑

Candidate for Eviction

```
globalPatternTable = {}

function accessCache(address):
    setIndex = extractSetIndex(address)
    tag = extractTag(address)
    signature = generateSignature(address)
    set = cache[setIndex]

    if tag in set:
        // Cache hit
        updatePredictionCounter(signature, true)
        set.blocks[tag].outcome = true
    else:
        // Cache miss
        if isSetFull(set):
            victimTag = selectVictim(set)
            updatePredictionCounter(set.blocks[victimTag].signature, set.blocks[victimTag].outcome)
            evictBlock(set, victimTag)
        insertBlock(set, tag, signature)
        set.blocks[tag].outcome = false
```

```
function generateSignature(address):
    // Generate a signature based on the address
    return hash(address)

function updatePredictionCounter(signature, outcome):
    if signature not in globalPatternTable:
        globalPatternTable[signature] = initialCounterValue()
    if outcome:
        incrementCounter(globalPatternTable[signature])
    else:
        decrementCounter(globalPatternTable[signature])

function selectVictim(set):
    // Use prediction counters to select a victim
    for tag in set:
        signature = set.blocks[tag].signature
        if globalPatternTable[signature] <= threshold:
            return tag
    // Fallback to another policy (e.g., LRU)
    return selectLRUVictim(set)

function initialCounterValue():
    return saturatingCounterMax / 2

function incrementCounter(counter):
    if counter < saturatingCounterMax:
        counter += 1

function decrementCounter(counter):
    if counter > 0:
        counter -= 1
```

We conducted experiments to evaluate the Signature-based Hit Predictor (SHiP) cache replacement policy across various cache configurations using three benchmarks: **531.deepjseng**, **541.leela**, and **548.exchange2**. Our findings indicate that SHiP effectively adapts to different workload patterns, achieving higher hit rates as cache sizes and associativities increase. For benchmarks with predictable access patterns, such as **548.exchange2**, SHiP achieved hit rates exceeding 99.99% even at smaller cache sizes when associativity was high. In workloads with more complex or dynamic access patterns, like those seen in **531.deepjseng** and **541.leela**, SHiP's performance significantly improved with larger caches, demonstrating its ability to learn and predict data reuse. Overall, SHiP outperformed traditional policies like Least Recently Used (LRU) and Least Frequently Used (LFU) in certain scenarios, highlighting its potential as a robust and adaptive cache replacement strategy for optimizing system performance across a variety of workloads.

Comparison of Caching Policies

LRU (Least Recently Used):

- **Pros:** Easy to implement and effective for workloads that exhibit temporal locality (recently accessed items are likely to be accessed again).
- **Cons:** May not perform well when access patterns change frequently.
-

LFU (Least Frequently Used):

- **Pros:** Works well for workloads where certain data is accessed frequently.
- **Cons:** Can lead to cache pollution, as infrequently accessed blocks may remain in the cache for too long.
-

Signature Predictor

- **Pros:**Adapts to varying access patterns by learning the reuse behavior of blocks.
- **Cons:** More complex to implement, with additional overhead required to maintain prediction counters.

Conclusion of Design

The design of each cache replacement policy was carefully developed to optimize hit rates based on the specific characteristics of the workload:

- Least Recently Used (LRU) was anticipated to perform well for applications that exhibit strong temporal locality.
- Least Frequently Used (LFU) aimed to capture frequently accessed data but required mechanisms to prevent outdated data from occupying the cache.
- Signature Based Predictor attempted to predict the usefulness of data blocks by learning from past access patterns, providing an adaptive approach to cache replacement.

By implementing these policies with efficient data structures and algorithms, we ensured that the cache system could quickly and accurately decide which blocks to retain or evict, ultimately enhancing overall system performance.