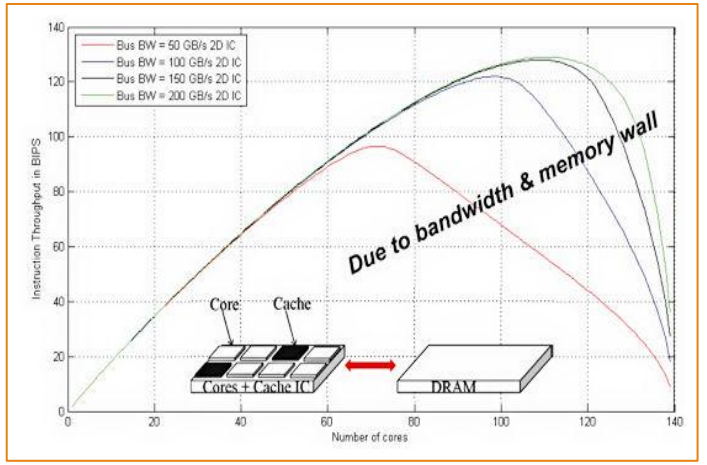
# A Cache Scheme Based on LRU-Like Algorithm

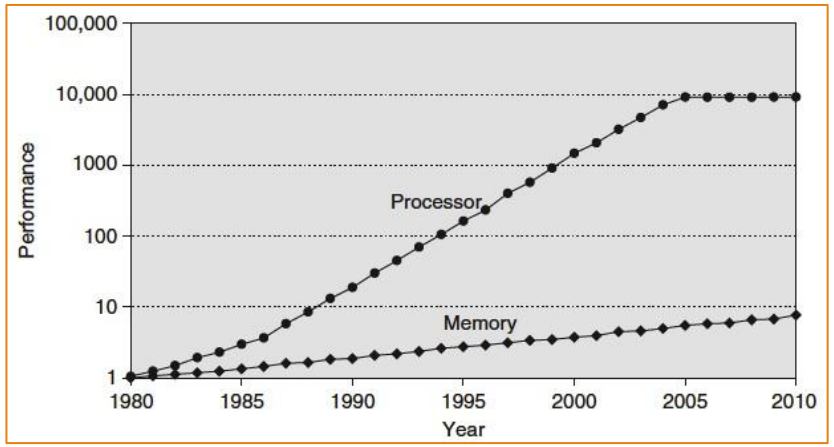
DONGXING BAO, XIAOMING LI, "A CACHE SCHEME BASED ON LRU-LIKE ALGORITHM", PROCEEDINGS OF THE 2010 IEEE CONFERENCE ON INFORMATION & AUTOMATION (ICIA),PAGES 2055-2060.

Ishan Dalal

# Memory Wall

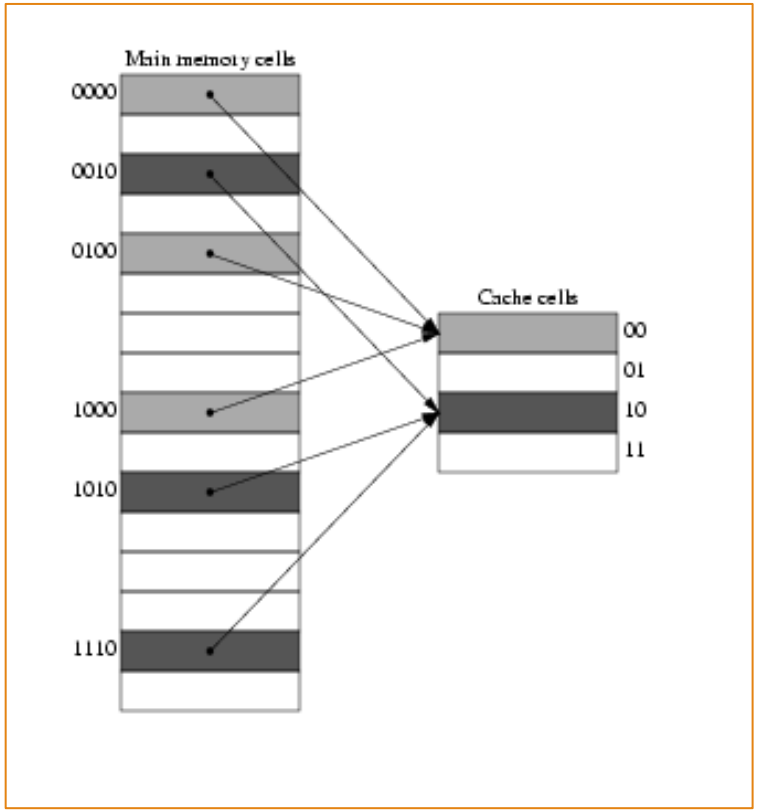
- The Growing Disparities between Processor & Memory Systems is making cache misses increasingly expensive.
- Memory Access Latencies have been a bottleneck for high performance microprocessors.
- Way to get around this issue
  - Improve cache memory by using intelligent control over on-chip cache management, in order to adapt caching decisions to dynamic accessing behavior.



# Direct-Mapped (DM) Cache

- Advantages :
  - Less complicated organization enables reduced hit time.
  - Reduce Design Cycle time
  - Enable variety of cache size configurations easily.
- Disadvantage:
  - Conflict Misses are more as types of cache configurations.



# Different Cache Designs

- Decoupled Cache
- Multiple-Access Cache
- Augmented Cache
- Multilevel Cache

- Augmented Cache Characteristics
  - Has a Direct-Mapped Cache & a smaller fully Associative Cache.
  - Both caches accessed in parallel with small access latency because of small capacity of fully associative cache.

# Reusability of Data & Cache Replacement Schemes

- Studies have shown that very small number of load/store instructions responsible for large percentage of cache misses.
  - This makes Selective Caching very important.
- Most caching schemes focus on reusability of data for optimizations to hold reusable data on cache as long as possible.
- Replacement Algorithms like LRU, Random are used to decide which blocks should be replaced on cache miss.
- Disadvantage of LRU
  - Consumes large hardware resources on chip
  - Increase Hardware complexity.

# Proposed Scheme : LRU-Like Algorithm (1/2)

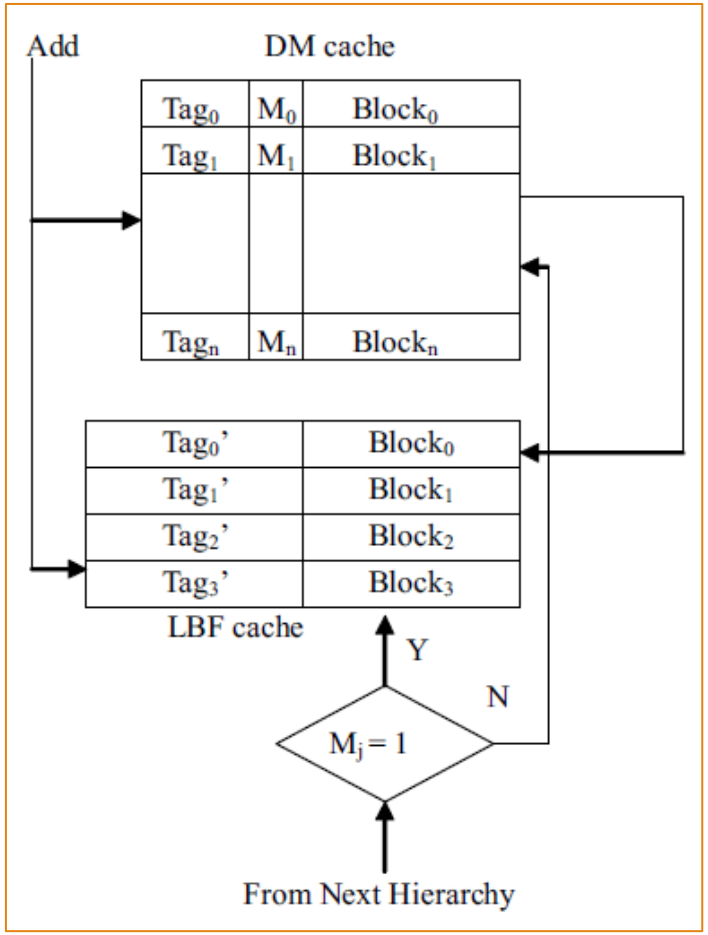
- Locality Principle
  - A block most recently used (MRU Block) could be accessed again soon. On a cache miss, block determined to be evicted should be least-recently used block (LRU block).
- LRU Design
  - Each way in a set is attached to a counter to record the accesses recently.
  - This enables to identify MRU & LRU block. When set is full, LRU block is replaced.
- LRU-Like Design
  - Based on Augmented Cache Design, buffer used to filter the LRU Blocks.
  - Holding all MRU Blocks in DM Cache in increase the hit rate.

# LRU-Like Algorithm (2/2)

- Suppose Bm block is in DM cache & other blocks are competing for same set are in buffer.
- One bit is assigned to each block in DM Cache to indicate MRU block.
- METHOD
  - On cache hit in DM cache, Bm is marked as MRU block.
  - On cache hit in Buffer, block of same set as needed block in DM cache is marked LRU block.
  - On cache miss, location of replacing block is indicated by the bit value in corresponding set.
  - If bit = 1 (MRU), replacing block filled in buffer.
  - If bit = 0 (LRU), replacing block filled in DM cache.

# LRU BLOCK FILTERING (LBF) CACHE SCHEME

- Least-Recently-Used Block Filtering Cache used.
- For DM Cache, tag of each block is added one MRU block judge bit (M bit), which is used to detect if block was accessed recently.
- Block in DM cache obtains more reusability. So evicted block from DM cache should be kept in L1 cache for longer time.



# Working of LBF (1/2)

- CPU accesses DM cache & buffer in parallel.
- Cache Hit
  - Hit in DM Cache : M bit for hit block set to 1.
  - Hit in buffer : M bit for corresponding set in DM cache cleared.
- Cache Miss
  - Fetched Block from next level of memory hierarchy is filled into DM cache if M bit for same set of miss block is 0.
  - Else, Block filled in buffer.

# Working of LBF (2/2)

- Block Transfer
  - When CPU waits for required block, evicted block from DM cache fills in the buffer.
- Cache Refilling
  - When block refilled into buffer for 1st time, M bit for same set in DM cache is cleared to 0.
  - When block refilled into DM cache for 1st time, M bit is set to 1.

# Effectiveness of Scheme for Different Reference Patterns

- **Conflict Between Loops**
  - **For conflicts between references inside two different loops, there is memory access pattern:** $(A_m A_h{}^9 B_m B_h{}^9)^{10}$

**Superscript denotes frequency usage of particular data word address scheme.**

- **Behavior of conventional DM-cache is**
  - **Miss rate = 20/200 = 10%**

# Contd..

- **Behavior of Augmented Cache Scheme**
  - When A refilled into DM cache, M bit is set. B is requested after nine hits of A, it will be refilled into the buffer.
  - For next nine cycles, no conflict occurs.
  - On other hand, if A is refilled into buffer for 1st time, M bit for corresponding set in DM cache is cleared. So B is refilled into DM cache.
  - For both refilling conditions, access behavior is $A_m A_m{}^9 B_m B_h{}^9 (A_h{}^{10} B_h{}^{10})^9$
  - Miss rate = 2/200 = 1.0 %

Thus LBF cache may gain more hits in this condition.

# Conflict Between Inner & Outer Loops

- Conflict arises between reference inside a loop with another reference outside inner loop.
  - Memory access pattern is $(\boldsymbol{A_m A_h}^{\boldsymbol{9}} \boldsymbol{B_m})^{\boldsymbol{10}}$
  - Miss rate = 2/11 = 18%.

- Using Augmented Cache Scheme. Same refilling process guarantees both block A & B will be reserved in L1 cache.
  - Behavior of LBF cache would be $\boldsymbol{A_m A_h}^{\boldsymbol{9}} \boldsymbol{B_m} (\boldsymbol{A_h}^{\boldsymbol{10}} \boldsymbol{B_h})^{\boldsymbol{9}}$
  - Miss rate = 2/110 = 1.82 %

# Conflicts within Loops

- There are two references A & B within a single loop mapping to same location in cache.
  - Behavior of DM cache is $(\boldsymbol{A_m B_m})^{\boldsymbol{10}}$
  - Miss rate = $M_{DM}$ = 20/200 = 100%
- For LBF cache, after initial A miss & B miss in L1 cache, next time A & B will be put in DM cache and buffer.
  - Behavior is $\boldsymbol{A_m B_m {A_m}^9 {B_h}^9}$
  - Miss rate = 2/20 = 10 %

Thus LBF cache may improve hit rate of direct-mapped cache.

# Simulation Environment

- Sim-outorder in SimpleScalar toolset used as Simulator. Mlcache used to replace corresponding cache program in sim-outorder.c
- Simulation was done to evaluate performance of L1 Dcache. Icache & bus were supposed to ideal.
- All benchmarks for simulation were from SPEC95.

| Processor & Memory Characteristics | |
|---|---|
| Fetch Mechanism | Fetches up to 4 instructions in program order per cycle. |
| Branch Predictor | Bimodal Predictor with 2048 entries. |
| Issue Mechanism | Out-of-order issue of up to 4 operations per cycle, 16 entry re-order buffer, 8 entry load/store queue. |
| Functional Units | 4 integer ALUs, 4 FP ALUs, 1 integer MULT/DIV, 1 FP MULT/DIV |
| Data Cache | Write-back, write-allocate, 32-byte lines, 4 read/write ports, non-blocking |

# Performance Metrics

- Effective Access time
  - If h is hit rate in L1 cache, t is access time of L1 cache & teff is effective access time, then Effective access time = $\boldsymbol{t_{eff} = h * t + (1 - h) * misspenalty.}$

  Access time of L1 cache is one cycle, & miss penalty is 18 cycles.

- ***Miss rate = L1 Dcache miss number / L1 Dcache reference number***
- ***Speedup = $t_{eff}$ of the target structure / $t_{eff}$ of the base structure***
- Bus traffic that indicates words L1 Dcache swaps with next level of memory hierarchy is another important metric.
- ***Relative Bus Traffic = Bus Traffic of base structure – Bus Traffic of target structure.***
  - If relative bus traffic is more, speed of processor is faster.

# Results

- DM cache, 2-way associative cache victim and assist cache are simulated to be compared to LBF cache.
- All augmented caches for compare are 8 KB DM cache & 1 KB buffer. Block size is 32B for all caches.

| MISS RATES OF 6 L1 DCACHE SCHEMES | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | compress | gcc | li | ijpeg | perl | hydro2d | su2cor | swim |
| DM:8K | 0.0673 | 0.0462 | 0.0231 | 0.0564 | 0.0597 | 0.1195 | 0.0891 | 0.4068 |
| DM:16K | 0.0557 | 0.0288 | 0.0178 | 0.0221 | 0.0373 | 0.1063 | 0.0796 | 0.1424 |
| 8K2W | 0.0563 | 0.03 | 0.0148 | 0.0448 | 0.0288 | 0.1112 | 0.0768 | 0.3904 |
| Victim | 0.0553 | 0.0259 | 0.0141 | 0.0111 | 0.0246 | 0.1094 | 0.0709 | 0.0473 |
| Assist | 0.0562 | 0.0287 | 0.0148 | 0.0118 | 0.0281 | 0.1095 | 0.0722 | 0.0472 |
| LBF | 0.0543 | 0.0265 | 0.0137 | 0.0104 | 0.0235 | 0.1078 | 0.0723 | 0.0463 |

# Miss Rates

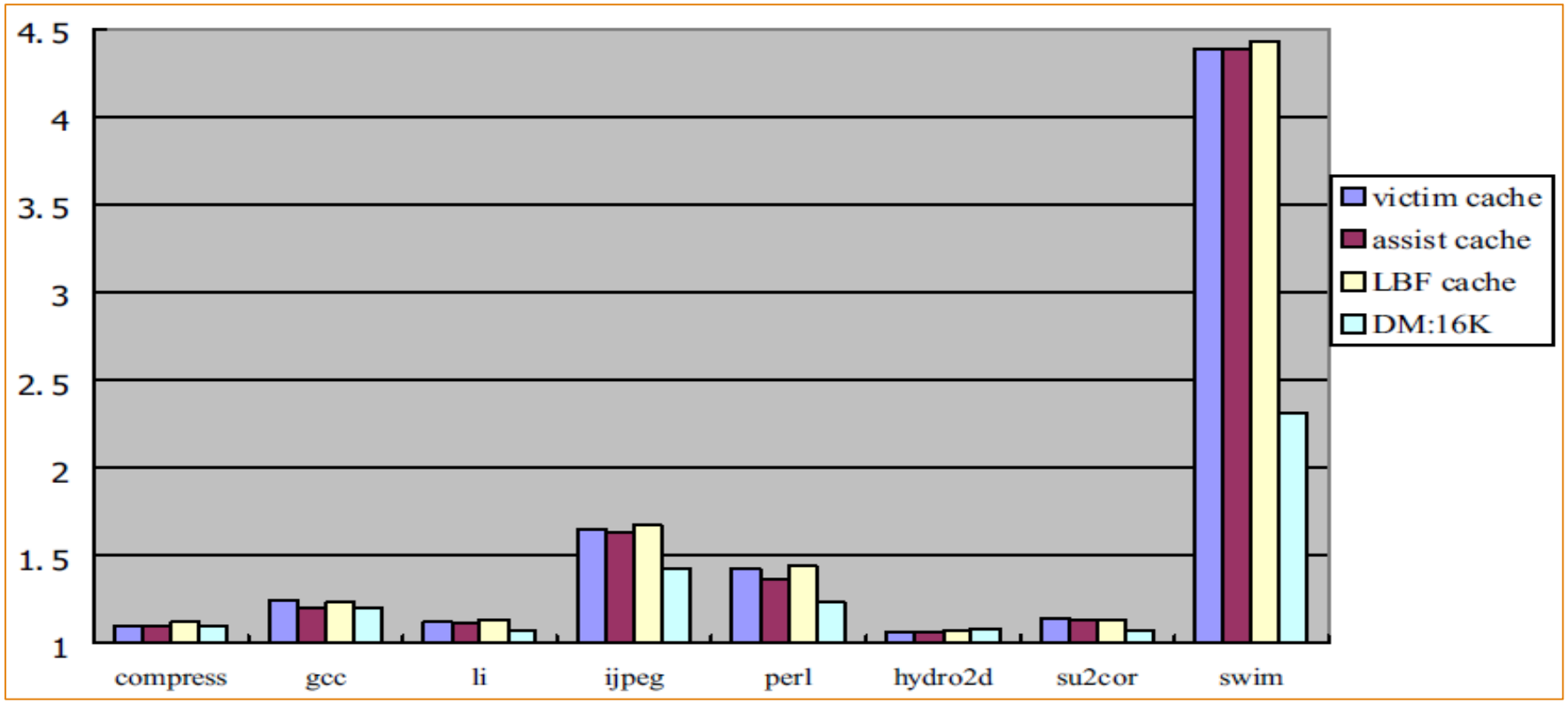
- Miss rates of LBF cache are basically lower than those of assist cache except that of su2cor.
- Also miss rates of LBF are lower than those of victim cache except for gcc & su2cor.
- Among three cache schemes, average miss rates of LBF cache, victim cache & assist cache are 5.34%, 5.38% & 5.48% respectively.
- Compared to 16 KB DM cache & 8 KB 2-way associative cache, the average miss rate of (8+1) KB LBF cache decreases about 26% & 53% respectively.

# Bus Traffic

| Relative Bus traffic of four L1 Dcache Schemes (million word) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | compress | gcc | li | ijpeg | Perl | hydro2d | su2cor | Swim |
| LBF | -3.3 | -6.4 | -3.0 | 0.5 | 2.07 | -129.8 | -90.9 | 177.9 |
| Victim | -11.1 | -64.4 | -122.9 | -6.4 | -14.3 | -319.8 | -309.1 | -449.5 |
| Assit | -8.2 | -30.1 | -57.9 | -0.3 | -3.4 | -274.9 | -212.9 | 124.8 |
| 8K2W | 0.1 | -0.4 | 16.4 | -2.1 | 2.4 | -8.8 | 11.4 | -144.0 |

- 16KB DM cache is considered as base structure.
- Accesses to bus for LBF cache is the least among four cache structures.

# Speedup




➢8KB DM cache is base structure. Among the four, speedup of LBF cache is highest.

# Hardware Overhead

- LBF is easier to implement than victim & assist cache.
- LBF cache only utilizes a one-direction path to transfer evicted blocks while victim cache needs two paths.
- On judgement of data transfer, assist cache needs the help of software interpreter while LBF cache doesn't require one.

# Shortfalls !

- There are no individual bits per block used in Direct Mapped Cache in Victim Cache Scheme. So LBF has little hardware costs than victim cache.
- Simulations were done only for L1 Dcache. Same for L1 Icache could have been carried out since instructions generally exhibit better locality than data. So other schemes could have performed better.
- Performance comparisons could have been made with other Direct-Mapped schemes like Non-Temporal Streaming (NTS), Memory Address Table (MAT) & Allocation By Conflict (ABC) to gain a better understanding.

# Conclusion

- LRU-Like algorithm introduces new augmented cache structure called LBF and it gives better simulation results than victim & assist cache.
- Results show miss rates of LBF cache are better than that of Direct Mapped & 2-way Associative Cache.
- LBF also performs better than other augmented cache designs like victim & assist cache.

# QUESTIONS ??

# Fixed Segmented LRU Cache Replacement Scheme with Selective Caching

KATHLENE MORALES & BYEONG KIL LEE, PROCEEDINGS OF INTERNATIONAL PERFORMANCE COMPUTING & COMMUNICATIONS CONFERENCE (IPCCC), 2012 IEEE 31ST INTERNATIONAL, PAGES 199-200.

**Ishan Dalal**

# Introduction

- Cache Replacement Policies like LRU (Least Recently Used), Random, FIFO etc. play an important role in bridging the gap in speed between CPU & memory.
- Some policies better than the more common LRU perform significantly better in reducing cache misses.
  - But hardware cost of implementing them is more.
- For mobile computing & SOC technologies, such hardware costs are unacceptable.
- Challenge is to design a cache replacement algorithm that is low cost but at the comparable performance.

# Proposed Scheme : Segmented LRU

- **Idea : If Line has been accessed while occupying the cache space, it should be more difficult to be evicted than a line that has never been accessed.**
- **Implementation : Adding a reference bit into each cache line.**
- **This Bit divides the cache set into two segments :**

1. **Protected Segment**
2. **Probationary Segment.**

# Implementation & Advantage

- All lines entering the cache are initially part of probationary segment.
- If cache hit occurs on a line in probationary segment, that line is promoted to the protected segment.
- On a cache miss, victims for replacement are chosen from probationary segment.
- Lines from protected segment are victimized only if the probationary segment is empty.

- Advantage over LRU
  - Better exploits temporal locality of lines by protecting more frequently used lines.

# Enhancements to SLRU

1. Apply fixed number of lines in protected & probationary segments to emphasize on the protected segments.
2. Selective caching method that prevents predicted dead blocks from entering the cache.

# SLRU with fixed Segmented Sizes (1/4)

- **Based on Observations, it was found that only 1-2 lines occupied the protected segments.**
- **Study proposed to fix a constant number of protected & probationary ways to increase the protected segments & avoid dynamic segmentation cost.**
- **In contrast to normal SLRU, this scheme also handles the eviction of lines from the protected segment.**
  - **When cache hit occurs in probationary segment , it is promoted to protected segment.**
  - **If protected segment is full, then a line is evicted from it to maintain fixed ratio.**
  - **That Line is evicted using LRU.**

# SLRU with fixed Segmented Sizes (2/4)

- Notation N:P is used for fixed SLRU policy where N is number of protected segments & P is number of probationary segments.
- N+P must equal the associativity of cache.
- N:P affects the cache performance.
- In order to find optimum ratio, multiple simulations were performed using SPEC 2006 benchmarks with default cache configuration
  - 1k 16-way set associative cache, 64 bit block size.
- All traces were simulated for 100 million instructions, after fast-forwarded 40 billion instructions.

# SLRU with fixed Segmented Sizes (3/4)

| Benchmark | Best Segmentation Ratio |
|---|---|
| Bzip2 | 11:5 |
| mcf | 15:1 |
| Hmmer | 14:2 |
| Gcc | None |
| Sjeng | 9:7 – 15:1 |
| Namd | 11:5 |
| Groamcs | 14:2 |
| Milc | None |
| Solpex | 11:5 |
| povray | 8:8 – 14:2 |

# SLRU with fixed Segmented Sizes (4/4)

- Table shows best performing segmentation ratios for benchmarks.
- Three benchmarks such as bzip2, milc & namd have lowest CPI with ratio of 11:5.
- Two benchmarks such as sjeng & povray have their lowest CPI over range of segment ratios, which included 11:5.
- Three other benchmarks including hmmer, mcf & gromacs have their lowest CPI at segment ratio of 11:5, but overall higher segment ratios performed better for these benchmarks.
- For milc & gcc, the segment ratios did not affect the CPI.
- Based on results, 11:5 was chosen to be an optimum ratio.

# SLRU with Selective Caching

- Selective Caching selects instructions to be bypassed based on their reference history.
- Bypassing “Dead” Blocks which are blocks that are replaced in the cache before they are accessed can effectively increase performance.
- Study predicts the blocks that were considered dead last time they occupied cache will tend to be dead blocks the next time they occupy the cache.
- By bypassing these blocks, cache does not have to needlessly evict lines that are more likely to be accessed to make room for dead blocks.

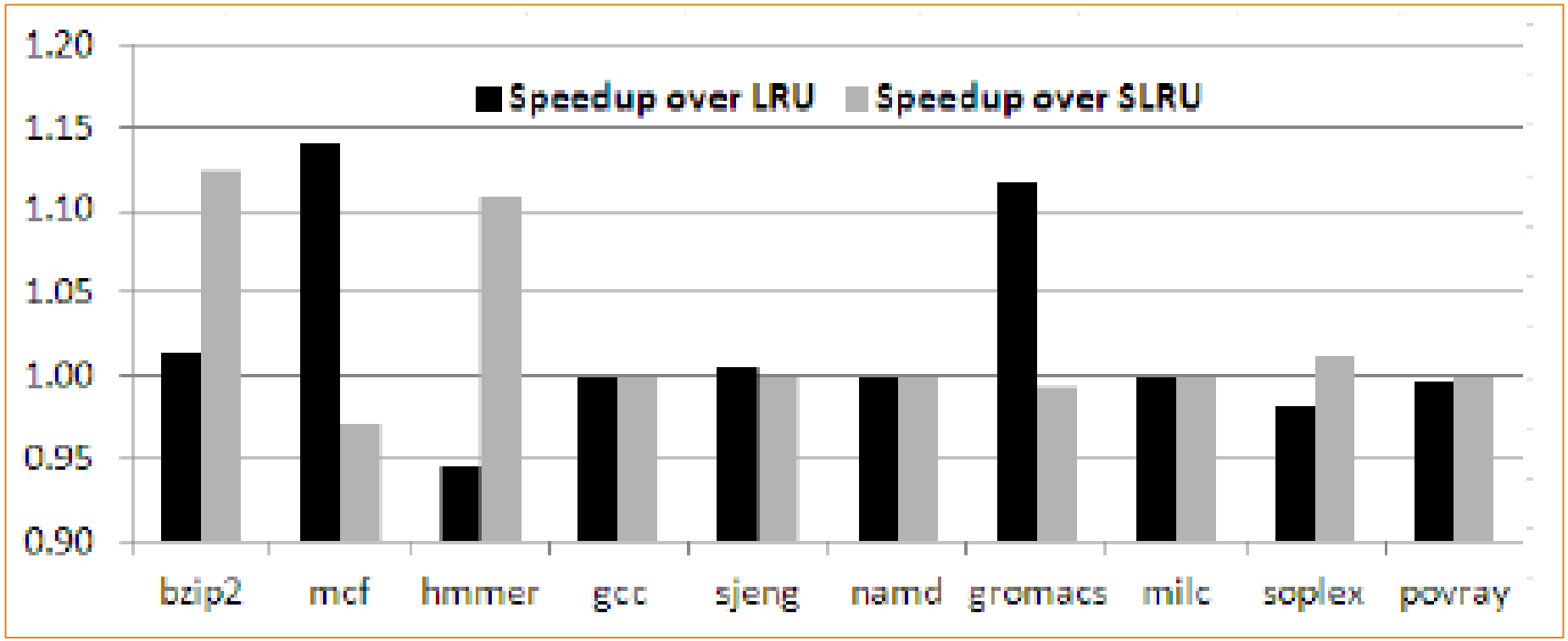
# Implementation

- An additional one bit per line is required for implementing this to fixed SLRU.
- This bit will represent whether a line has been accessed at least once while in cache.
- If it has not, its tag will be updated to a table, called bypass table.
- Based on experiments, it was found bypass table to be most effective when it held a max of 16 tags.
- After tag has been used to bypass cache once, the line is cleared from table.
- Tags were first written to any empty lines in the table. If no empty lines are available, tags are overwritten to the first line in table.
- Selective caching can be implemented without increasing cache access latency by treating bypass table as an extra way in cache.

# Simulations : Fixed SLRU with Selective Caching

- Fixed SLRU with selective caching was compared with both LRU & SLRU using 5 floating point benchmarks & 5 integer benchmarks.
- Simulations were performed with proposed replacement algorithm applied to LLC (Last Level Cache).
- All other level of caches applied with LRU.
- 1M 16-way set-associative LLC cache with 64 bit block size was used.

# Results


1.20
1.15
1.10
1.05
1.00
0.95
0.90
Speedup over LRU
Speedup over SLRU
bzip2
mcf
hmmer
gcc
sjeng
namd
gromacs
milc
soplex
povray

# Results Analysis

- For single core simulations, average speedup of 1.67% over LRU (max – 14 %) & 1.85 % over standard SLRU (max – 12.5%).
- Only Hmmer experienced a significant decrease in CPI versus LRU.
- Additional enhancements to this algorithm like aging, random promotion etc could result in better performance replacement algorithm.

# Hardware Costs (1/2)

- For a 16-way cache, additional 6 bits per cache line is required.
- Four bits are required to indicate stack position, one bit is used to distinguish between probationary & protected segment & one bit is used to mark whether a line has been referenced while in cache.
- Partial tag of 16 bits can accurately represent 64 bits. So bypass table requires 256 bits of memory.
- In total, single core implementation for a 16-way cache requires 96 bits per cache set & 256 bits for the bypass table.

# Hardware Costs (2/2)

- Compared to other replacement policies, hardware requirements for this algorithm are relatively low.
- Dueling Segmented SLRU policy achieved speedup over LRU up to 8.6 % but at cost of 102 bits per set, plus another 23.4K of additional hardware.
- Compared to LRU, proposed algorithm only requires an extra 32 bits per set & 256 bits for the bypass table.

# Shortfalls !

- Compared to LRU & SLRU, It takes extra hardware to implement. Thus this algorithm cannot be used in mobile, system-on-chip implementation and other apps where high hardware costs are not acceptable.
- Study haven't compared the performance of Dueling segmented SLRU with Fixed SLRU although they have compared their respective hardware costs.
- Different benchmarks have different optimum segmentation ratios. So instead of using the same segmentation ratio, performance adjustable segmentation ratio could have been used for better results.
- Miss rates has not been compared between Fixed SLRU, normal SLRU & normal LRU.
- Insufficient information about the implementation of Selective Caching. For example – data about the wait time used by the study to determine which blocks are dead etc. is not provided.

# Conclusion

- Fixed SLRU with selective caching shows potential for high performance.
- Since the proposed algorithm did not use ideal segment ratio for every benchmark, an enhancement that adjusts the segment ratio based on performance could improve speedup.
- Although goal of the study was reduce on the hardware costs associated with cache replacement policies, it was not able to effectively achieve it.

# Questions ??

# Proposed Enhancements to Fixed Segmented LRU Cache Replacement Policy

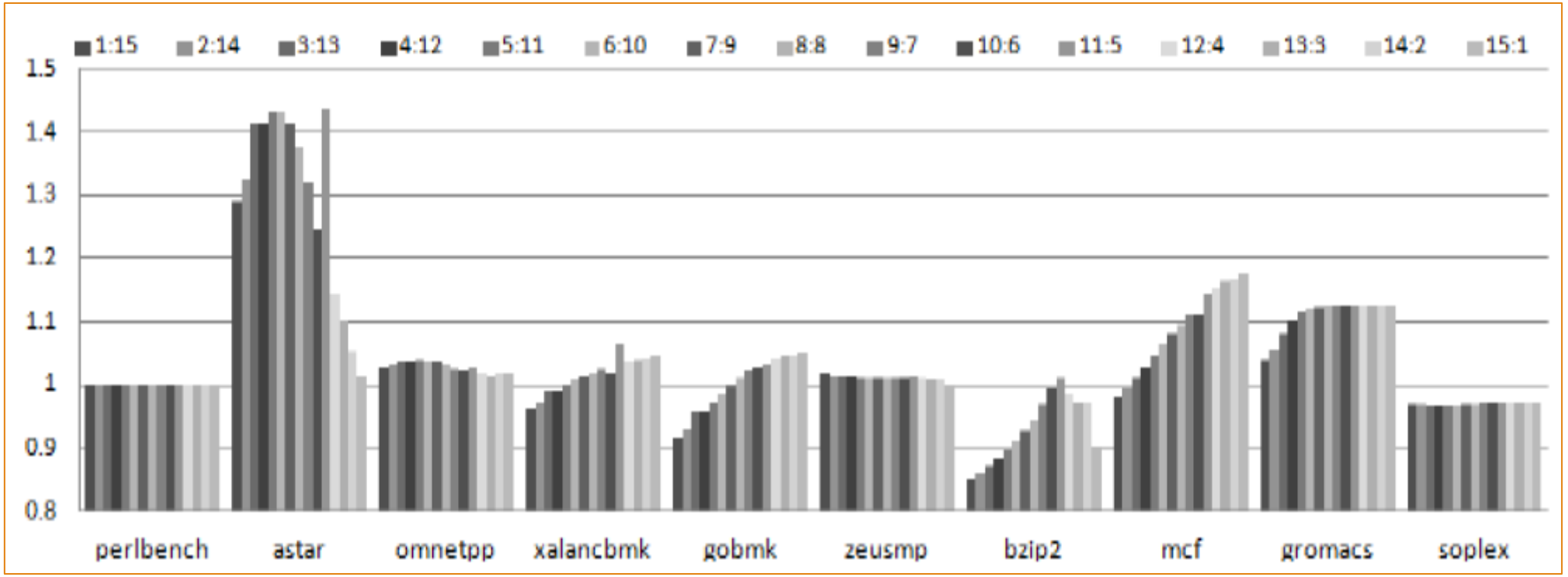
KATHLENE HURT & BYEONG KIL LEE, PROCEEDINGS OF INTERNATIONAL PERFORMANCE COMPUTING & COMMUNICATIONS CONFERENCE (IPCCC), 2013 IEEE 32ND INTERNATIONAL, PAGES 1-2.

# Introduction

- Due to Ineffective Simulation environment in the previous Paper, results obtained were not fully analyzed which resulted in incomplete understanding of the fixed SLRU with selective caching scheme.
- The current study has an improved simulation environment and focuses on three enhancements to the algorithm : cache bypassing, random promotion, & aging.

# SLRU With Fixed Segment Size

- Multiple Simulations were carried out to find the optimum ratio of protected to probationary segments.

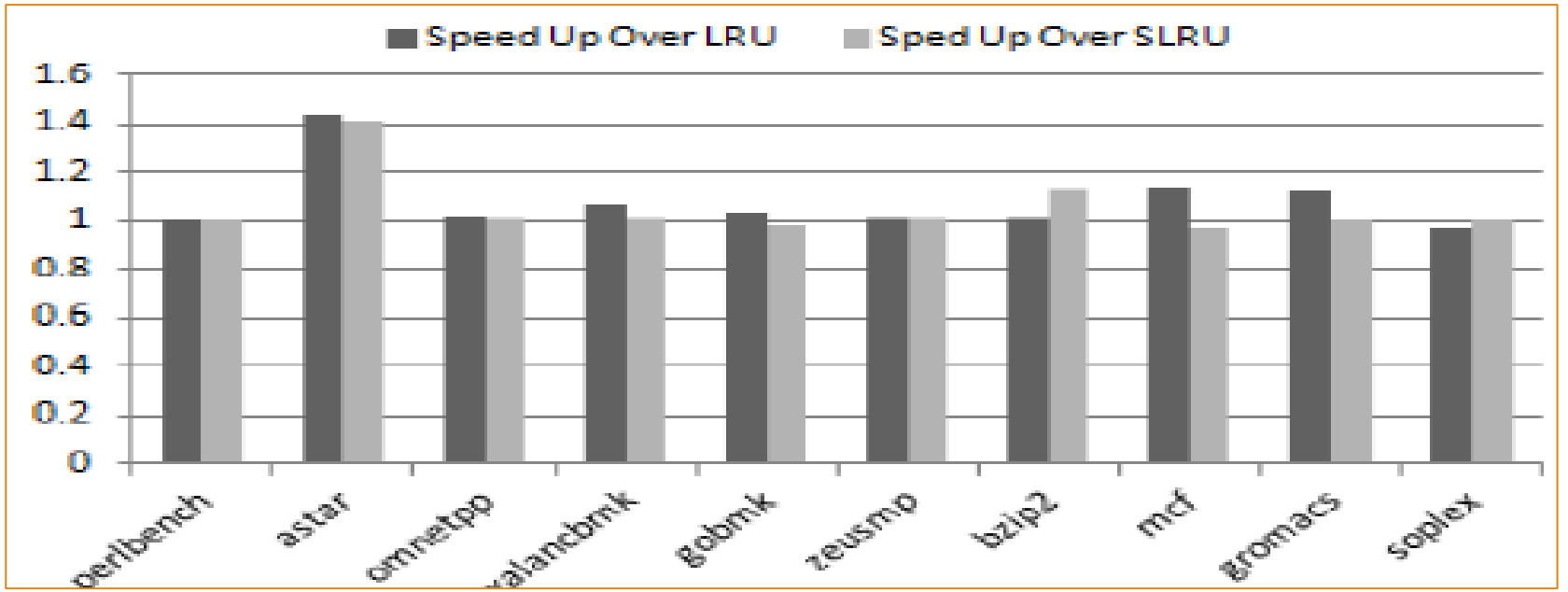


# Results : Simulation (1/3)

- Five benchmarks experienced their greatest speedups over LRU while using 11:5 segment ratio.
- Gobmk, MCF had their greatest CPI at 15:1 while zeusmp & gromacs experienced their greatest CPI at 1:15.
- From information, 11:5 is the ideal segment ratio to use in algorithm.
- When using 11:5, no benchmark experienced a decrease greater than 2.9% in CPI compared to its ideal segment ratio.

# Results : Simulation (2/3)

- Graph shows CPI speedup over LRU & SLRU when fixed SLRU algorithm was applied using 11:5 segment ratio.



# Results : Simulation (3/3)

- Average Speedups of 7.00% & 4.22% are achieved over LRU & SLRU.
- Maximum speedup of 43.55% over LRU & 41.59% over SLRU are also achieved.
- Over LRU, only Soplex experienced a decrease in CPI (2.7%).
- Over SLRU, two benchmarks experienced a decrease in CPI, but neither was greater than 2.97%.
- When using the ideal segment ratio for every benchmark, an average speedup of 7.54% over LRU & 4.73 % over SLRU was achieved.

# Enhancement 1 : Cache Bypassing

- Based on experiments, bypass table is found to be most effective when it held a maximum of 64 tags.
- An average speedup of 7.15% over LRU & 5.11% over SLRU was achieved with a maximum speedup of 48.97% & 46.94% respectively. It is 0.15% higher than LRU & 0.89% higher over SLRU than fixed SLRU alone.
- Of then ten benchmarks, 6 of them experienced a slight decrease in CPI (up to 3.01%) compared to 11:5 fixed SLRU.
- Since so many benchmarks experienced a decrease in CPI, and overall speedup was negligible. So Cache Bypassing is not an effective enhancement to fixed SLRU.

# Enhancement 2: Random Promotion

- Previous Study showed randomly promoting lines from probationary segment to protected segment can increase performance in SLRU.
- Enhancement of randomly promoting lines to protected segment as they enter the cache is proposed.
- From experiments, it was found that even for small probabilities (0.001%), nine benchmarks experienced decrease in CPI.
- Only Astar showed benefit from random promotion as it achieved max speedup of 61.7% over LRU, which is 41.8% increase over fixed SLRU alone, using a 20% random promotion probability.
- Thus overall, it was found that random promotion was found to be not beneficial enhancement for fixed SLRU.

# Enhancement 3: Aging

- Idea: Ability to remove lines from protected segment and return them to probationary segment.
- When a line is promoted to protected segment by a cache hit in the probationary segment, the LRU line in the protected segment is demoted to probationary segment and it is set to be evicted on next cache miss.
- Aging alters this algorithm by maintaining this lines stack position. So this line will not be evicted on next cache miss and it would be given an increased chance to be promoted back into protected segment.

# Results : Aging

- Average speedup over LRU of 4.78% & an average speedup over SLRU of 2.14% with maximum speedups of 17.10 % & 15.51% respectively.
- Only bzip2 experienced a performance boost from aging with 11.66% CPI speedup over fixed SLRU.
- This is a large decrease in CPI compared to 11:5 fixed SLRU alone.
- Thus from these results, aging is not an effective enhancement to add to fixed SLRU.

# Overall Result Analysis

- Only Enhancement out of the 3 that showed overall increase in CPI was cache bypassing.
- But too many benchmarks experienced a decrease in CPI for cache bypassing to be considered an effective enhancement.
- Random Promotion & aging were shown to decrease performance when applied to fixed SLRU.
- Overall, none of enhancements that were employed to SLRU were found to be effective.

# Shortfalls !

- More information about process of chosing the number of tags in bypass table is required because previous study used 16 tags while current study uses 64 tags.
- Instead of using 11:5 fixed segmentation ratio for all benchmarks, study could have used performance adjustable segmentation ratios to getter better results.
- Although Study has shown through results why enhancements didn't work for fixed SLRU, no analysis has been carried regarding the same.
- More details about the implementation about aging enhancement could have been given. For example – Data about the number of chances given to demoted line in probationary segment on a cache miss.

# Conclusion

- Fixed SLRU using 11:5 segment ratio achieved a 7.00% average speedup over LRU & 4.22% over SLRU.
- Maximum speedup of 43.55% over LRU & 41.59% over SLRU was obtained with improved simulation environment.
- This is significant considering fixed SLRU does not require any additional hardware over SLRU and only one extra bit over LRU.
- None of enhancements explored here were found to be beneficial to fixed SLRU.

# Questions ??

# Thank You !