Assisting Cache Replacement by Helper-Threading for MPSoCs

Masaaki Kondo

Graduate School of Information Science and Technology, The University of Tokyo

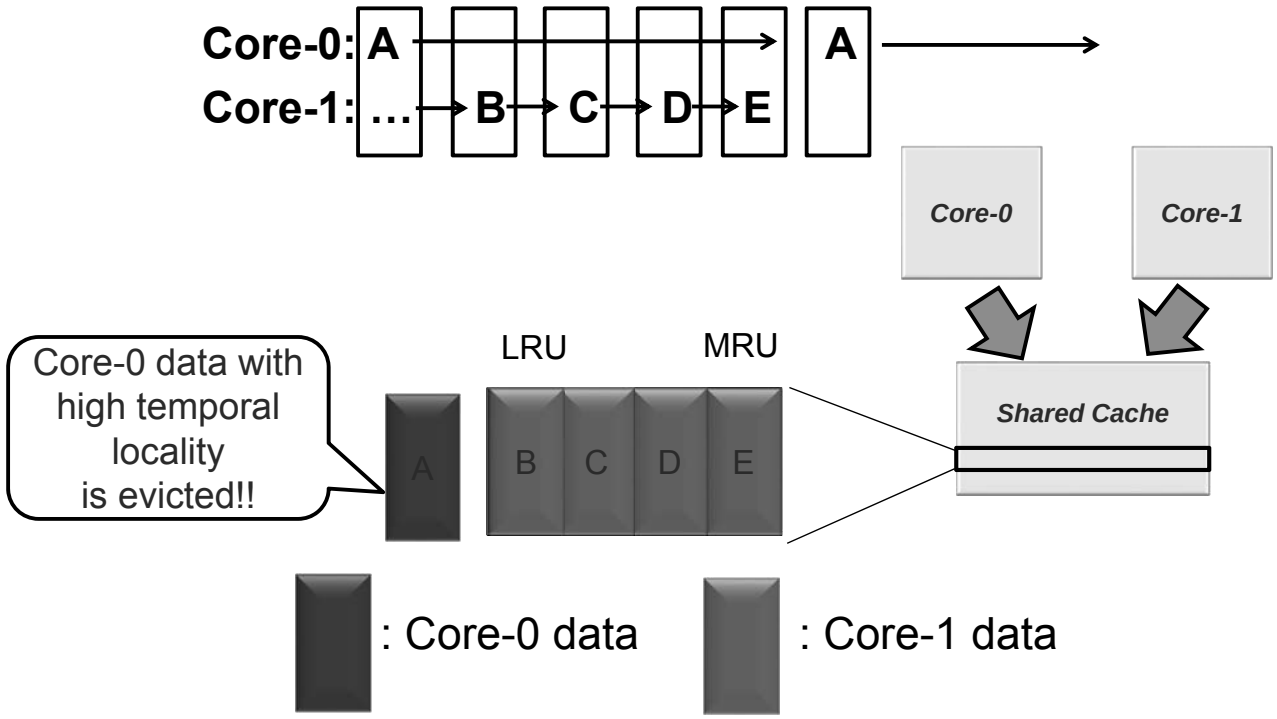
Background

- Increasing number of cores in MPSoCs
  - ✓ No more increase in clock speed
  - ✓ Performance improvement by parallel processing
  - ✓ Not easy to fully utilize all the available cores
- Typically with a shared last-level cache
  - ✓ ☺ Effective use in cache area
  - ✓ ☹ Destructive performance degradation by cache conflict
    - Multiple threads contend cache area due to difference of their access patterns

Aggravated with increasing number of cores

Cache Conflict

- Example memory access sequence

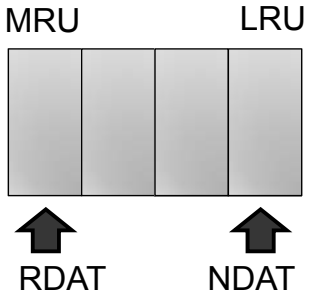


Objective and Strategy

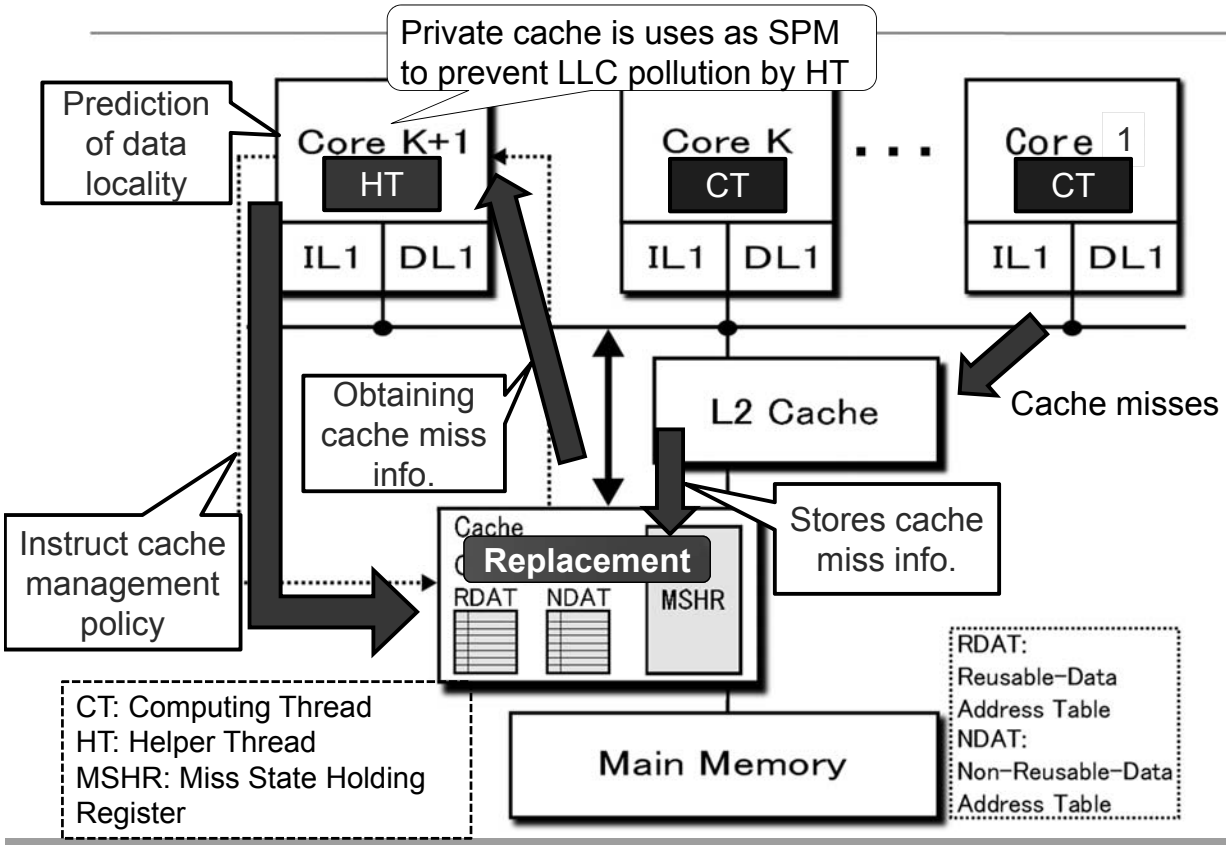
- Alleviating the effect of conflict in shared LLC
  - ✓ Lines with high-locality → keep them in cache
  - ✓ Lines with low-locality → evict them from cache ASAP
- Managing cache lines by helper threading
  - ✓ Flexible cache line management by software
    - Locality prediction and replacement control for cache lines
  - ✓ Making good use of idle cores with helper threading
    - Some cores tend to be idle in manycore SoCs due to lack of parallelism or smaller number of tasks than cores

Helper Thread Assisted Cache Replacement

- Required steps
  - ✓ Helper thread (HT) obtains cache miss info. from cache
    - Access **MSHR: Miss State Holding Register** via Memory-mapped I/O
  - ✓ HT predicts data locality for each missed line
    - Flexibly implemented by software → demonstrate 3 prediction methods
  - ✓ HT stores predicted result and HW manages replacement based on it
    - RDAT: Reusable Data Address Table
      - ✓Keeps addresses (pages) with high data locality
        - → Insert lines to MRU position for next line-fill
    - NDAT: Non-Reusable Data Address Table
      - ✓Keeps addresses (pages) with no data locality
        - → Insert lines to LRU position for next line-fill

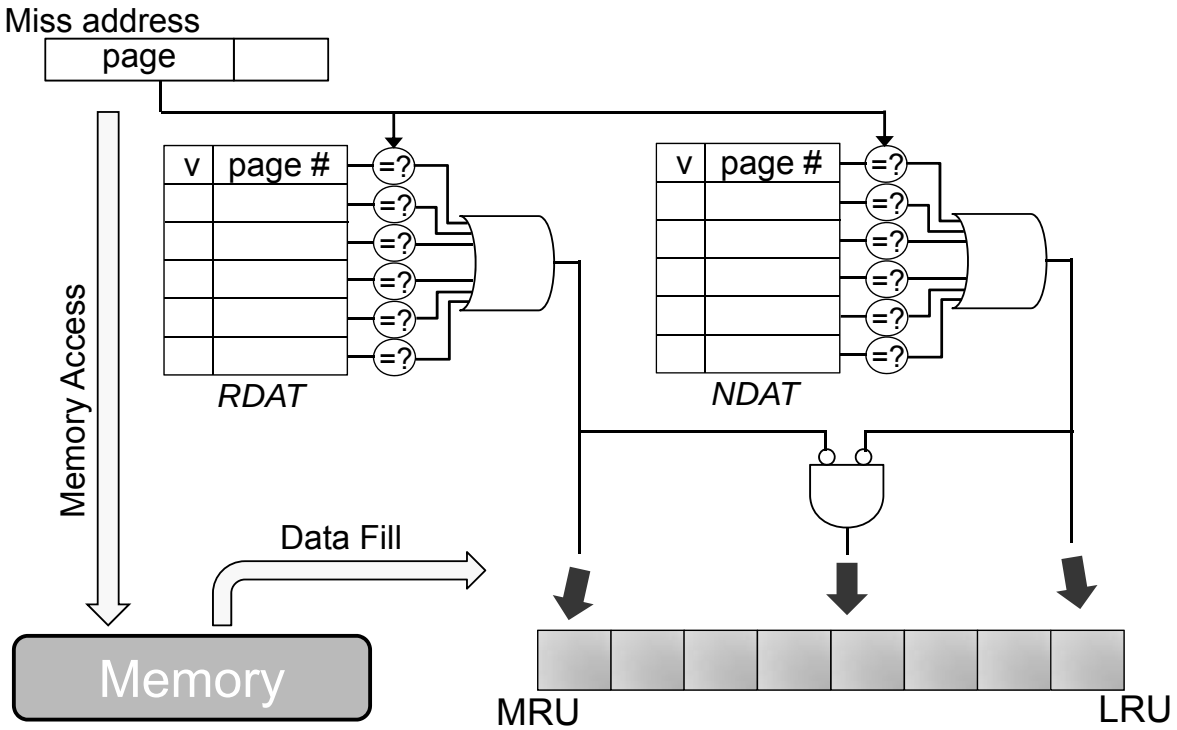


Baseline Architecture and Control Flow



Replacement Control

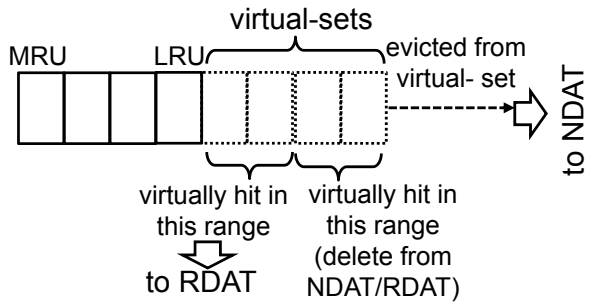
- Selecting line insert position based on RDAT and NDAT



Locality Prediction Algorithms

- Profile-based method
  - ✓ Data-locality is profiled in advance
  - ✓ Stores only non-reusable data addresses to NDAT
- Stream-based method
  - ✓ Detect stream accesses dynamically by access history
  - ✓ Stores addresses of stream data to NDAT (data with stream accesses is not likely to have locality)
- Virtual-set monitoring method
  - ✓ Chasing reusability of evicted lines using virtual-set
  - ✓ If an evicted line is accessed shortly, stores its address to RDAT
  - ✓ If evicted even from virtual set, stores its address to NDAT

Virtual-Set Monitoring Method




```
1.  // allocating virtual-set structure in the memory
2.  VirtualSet = Create_VirtualSet(SIZE);
3.  while (computing-threads-running) {
4.    // obtain cache miss info. from MSHR
5.    curMSHR = Read_current_MSHR();
6.    if !(curMSHR)
7.        continue;
8.    // check if it hits in the virtual set
9.    wayNum = isHit(curMSHR, VirtualSet);
10.   // check its locality and set page num. to NDAT/RDAT
11.   if (!wayNum)
12.       set_NDAT_entry(get_page(curMSHR));
13.   else if isReusableWayNo(wayNo)
14.       set_RDAT_entry(get_page(curMSHR));
15.   else
16.       delete_NDAT_RDAT(get_page(curMSHR));
17. }
```

- Monitors evicted lines from LLC
  - ✓ For only a few sample sets
  - ✓ HT emulates virtually extended LRU stacks for the sample sets
  - ✓ Check the reusability of lines by the virtual-set
- Virtual hit at the middle of LRU stack:
  → high data-locality
  - ✓ Register entire-page to RDAT
- Virtual hit at the end of LRU stack:
  → middle data-locality
  - ✓ Delete corresponding page from NDAT and RDAT
- Evicted from virtual-set:
  → low data-locality
  - ✓ Register entire-page to NDAT

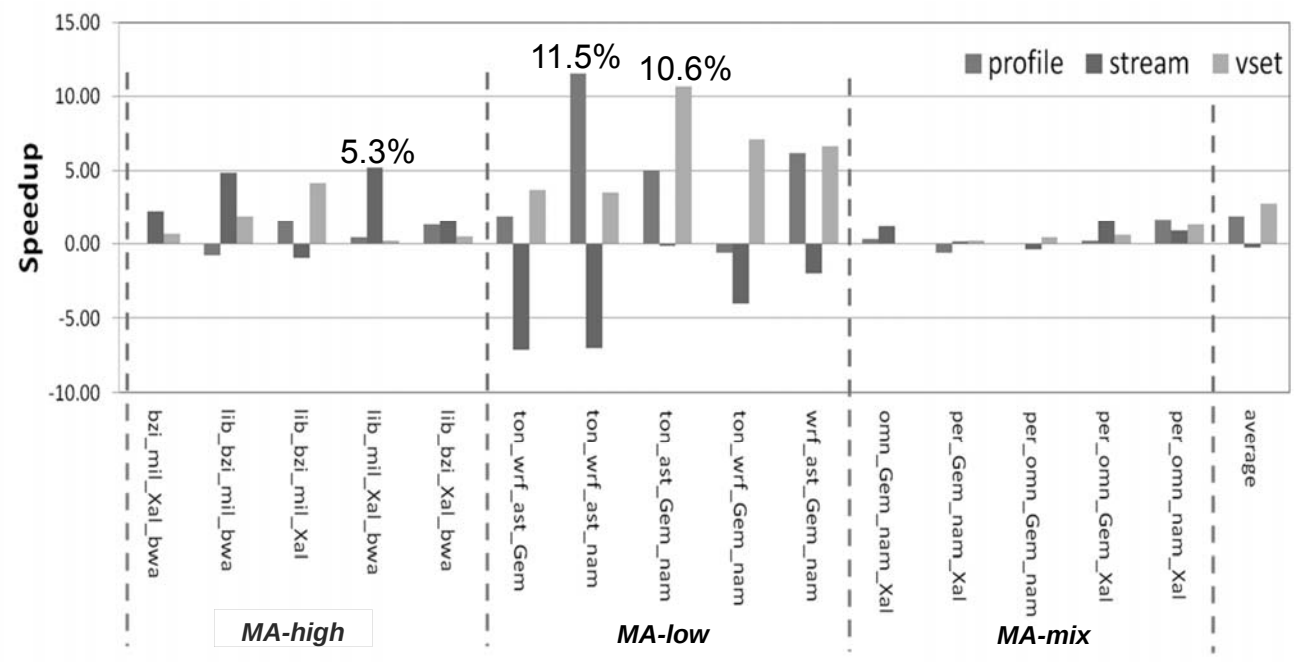
Evaluation Environment

- MARSSx86 multicore simulator
- Job mixes with SPEC CPU2006 (1-program/core)
  - ✓ MA-high: job mix of memory intensive applications
  - ✓ MA-low: job mix of CPU intensive applications
  - ✓ MA-mix: job mix of both memory and CPU intensive applications

Configuration

Parameter	Value
L1 D/I-Cache	32K, 8-way, 64Bline, 2-cycle latency
L2 Cache	1MB,8-way, 64Bline, 9-cycle latency
Main Memory	200-cycle latency
Executing instructions	100M inst. for all threads (FastForward:1B inst.)
number of cores	5 (4-application threads + 1-helper thread)

Result (4-Compute + 1-Helper Threads)



Summary

- Cache data management by helper threading
  - ✓ HT predicts data-locality and set hints to cache controller
  - ✓ Very flexible since we can implement multiple management algorithms by software modification
  - ✓ Performance (weighted speedup) improvement up to 11.5%

- Future work
  - ✓ Develop more accurate locality prediction algorithm
  - ✓ Select appropriate algorithm depending on a job mix
  - ✓ Compare with purely hardware-based counterpart