**Cache Prefetching and Offchip Bandwidth Partitioning/Scheduling**

Dr. A. Sahu  
CSE, IIT Guwahati

---

**Outline**

- Multiprocessor Cache Prefetching
- S/W Helper thread based prefetching
- Pre-fetch Aware memory scheduling

---

**Recall: Cache Policies**

- Placement: what gets placed where?
- Read: when? from where?
- Load: order of bytes/words?
- Fetch: when to fetch new block?
- Replacement: which one?
- Write: when? to where?

---

**Recall: Fetch Policies**

- Fetch on miss (demand fetching)
- Software prefetching
- Hardware Prefetching

---

**Recall: Fetch Policies**

- Demand fetching
  - fetch only when required (miss)
- **Hardware prefetching**
  - Automatically prefetch next block
- **Software prefetching**
  - programmer decides to prefetch
    questions:
    - how much ahead (prefetch distance)
    - how often

---

**Recall: Compiler Controlled Prefetching**

- Semantically invisible (no change in registers or cache contents)
- Makes sense if processor doesn’t stall while pre-fetching (non-blocking cache)
- Overhead of pre-fetch instruction should not exceed the benefit
**Software Pre-fetch: Means**

- Essentially get in to Cache by reading that memory address

\[ \text{Prefetch}(\&A[k]) \]

\[ \Rightarrow \text{Prefetch(int Add)} \]

\[ X=\text{Add} \]

\[ } \]

**SW Pre-fetch Example**

```
for(i=0; i<100; i++)
    A[i]=B[i]*C[i];

Prefetch(\&A[0]); Prefetch(\&B[0]); Prefetch(\&C[0]);
for (j=0; j<4; j++)
    A[j]=B[j]*C[j];
for(i=1; i<25; i++)
    k=i*4;
    Prefetch(\&A[k]); Prefetch(\&B[k]); Prefetch(\&C[k]);
for (j=0; j<4; j++)
    w=k+j;
    A[w]=B[w]*C[w];
} }
```

**H/W Data Pre-fetching : Array and Tree**

- Regular Access
- Stream Pre-fetcher

```
BFS Sequence: 1,2,3,5,4,7,6
```

- Irregular Access: But Determined Order and in node 1 information/Address of 2,3,5 are available

**CDP: Content Directed Pre-fetching**

- Pre-fetching can significantly reduce memory latency impact on performance
- Stream pre-fetching very useful but unable to reduce latency of many misses
- Access patterns that follow pointers in linked data structures (LDS) prevalent in many applications
- High-performance and bandwidth-efficient LDS prefetchers are needed

**Prefetching with Helper Threads for Loosely Coupled Multiprocessor Systems**


**Pre-fetcher : Stream + Content Directed**

- Technique to
  1) Enable low cost and bandwidth-efficient pre-fetching of linked data structure accesses
  2) Efficiently combine such pre-fetchers with commonly-employed stream based pre-fetchers
Basic Ideas

- Take any Institute
- Tier Architecture
  - Administrators (1 or 2)
  - Many Worker (50-100)
    - May be power full and intelligent
  - Many Helper (Peon/ Attendant ) (10-15)
    - May not be power full but efficient in doing manual work

Helper Pre-fetch

- It should pre-fetch and put in L2 before application access
- It should not pre-fetch much before
  - How much before
- General Idea:
  - Application is working on data i
  - Helper pre-fetch data i+L; L is distance : experimental
  - If helper is pre-fetching faster; wait to maintain L
- Synchronization:
  - Application & Helper to Make distance L
Demerits

- Require an processor/core to run helper thread
- Can be utilized with slower memory processor
  -- Required special SYNC with main core and memory processor

Adaptive Prefetch for shared cache
Based Chip multiprocessor

- M Kandemir et al, Adaptive Prefetch for
  shared cache Based Chip multiprocessor, DATE
  2009.

Shared L2 : Multiprocessor

- Shared L2 Based System
- Multiple Core complete for the shared on-chip
  cache & limited off chip BW
- May tends to degrade the performance

  Harmful Pre-fetches: which kicks out useful
  data from L2 whose usage is earlier then pre-
  fetch data

- Pressure on L2: require to increase Cache size

Required L2 Cache with Pre-fetching

- One Core : 2MB cache
- To maintain the same pre-fetching performance
  as in single core case

Performance Improvement

- Graph showing % improvement due to pre-fetching
  with respect to number of cores.

A Sahu
What to do with harmful pre-fetch

- Pre-fetch Suppressing:
  - Similar to Stopping Interfering Gundas by Police
  - If pre-fetches from core, is continually harmful and (harmfulness > Threshold) for time frame S
  - Suppress pre-fetches form Core, in time frame S+1

- Data Pining:
  - Similar to Providing Quota to Suffered Guys by Govt
  - Select data block brought to L2 marked as non-removable for Certain period of Time
  - Other will not be able to remove

Correcting Approach: Source Side and Serving Side

- Source Side (Originating side/L2 Cache)
  - Pre-fetch Suppressing
  - Data Pining

- Serving Side (Memory side/Scheduling BW)
  - Bandwidth sharing
  - Access scheduling
  - Batching

Fairness in using

- Multi program workload running on multi-core

- Fairness: How less Unfairness
  - Individual Slow down (IS) = \( T_{\text{shared}} / T_{\text{alone}} \)
  - Max Slowdown = MAX( IS_0, IS_1, ..., IS_{N-1} )
  - Min Slowdown = MIN( IS_0, IS_1, ..., IS_{N-1} )
  - Unfairness = Max Slowdown / Min Slowdown

Prefetch-Aware Shared-Resource Management for Multi-Core Systems

Eiman Ebrahimi, Chang JooLee, Onur Mutlu (CMU), Yale N. Patt (UTA)
ISCA 2011
**Network Fair Queuing (NFQ)**

- Multi program workload running on multi-core
- Network Fair Queuing:
  - Every core is assigned a memory bandwidth
  - Virtual finish time (VFT) of a request $R_i$ from core $c$ is finish time w.r.t. Bandwidth of Core $c$
  - All the memory request are served with Earliest VFT

**Parallelism-Aware Batch Scheduling (PARBS)**

- Principle 1: Parallelism-awareness
  - Schedules requests from each thread to different banks back to back
  - Preserves each thread's bank parallelism
- Principle 2: Request Batching
  - Marksa et al. fixed number of oldest requests from each thread to form a “batch”
  - Eliminates starvation & provides fairness

**Impact of Prefetching on Parallelism-Aware Batch Scheduling**

- Policy (a): Include prefetches and demands alike when generating a batch
  - Pros: Accurate prefetches will be more timely
  - Cons: Inaccurate prefetches from one thread can unfairly delay demands and accurate prefetches of others
- Policy (b): Prefetches are not included alongside demands when generating a batch
  - Pros: Inaccurate prefetch can not unfairly delay demands of other cores
  - Cons: Accurate prefetches will be less timely
  - Less performance benefit from prefetching
Prefetch-Aware Shared Resource Management

- Three key ideas:
  - *Fair memory controllers:* Extend underlying prioritization policies to distinguish between prefetches based on prefetch accuracy
  - *Fairness via source-throttling technique:* Coordinate core and prefetcher throttling decisions
  - *Demand boosting* for memory non-intensive applications

Fairness via Source Throttling (FST)

- Coordinate core and prefetcher throttling decisions
- Application slowest:
  - Core experiencing largest slow down
- Application Interfering:
  - Core creating most interference for \( \text{App}_{\text{slowest}} \)
- Calculate unfairness value
- If unfairness > Threshold:
  - *Throttle Up:* Increase BW share for \( \text{App}_{\text{slowest}} \)
  - *Throttle Down:* Decrease BW share for \( \text{App}_{\text{interfering}} \)
  - How much? Depend on level of interference
  - Threshold specified by System

Still Problem of Performance

- Both Network Nair Queuing (NQF) and Fairness via Source Throttling (FST)
  - Both treat Miss and Pre-fetch as same demand
  - Reduce the performance
- Solution:
  - Pre-fetch should be treated as demand only when they are useful (Prioritized Pre-fetch)
  - But by doing this, delay demand request of non memory intensive application (NMIA)
  - Boost the demand of NMIA

References

- Onur Mutlu (cmu), Yale Patt (u texas), et al, *Prefetch aware Shared Resource management for Multi-core System*, ISCA 2011