Cache Policy and Performance

Cache Access Time: Through Lens

- $AMAT = 1 + \text{Prob}_{\text{miss}} \cdot \text{Penalty}_{\text{miss}}$
- Miss Penalty include Swap in + Swap out Time
- When to swap out if it is dirty/written (Not for I-cache block)
- $AMAT = 1 + \text{Prob}_{\text{miss}} \cdot T(2 - \text{P}_c)$
  \(\text{P}_c\) is Probability of block is clean

Cache Access Time: Through Lens

- $AMAT = 1 + \text{Prob}_{\text{miss}} \cdot T(2 - \text{P}_c)$
- Miss Penalty include Swap in + Swap out Time
- When to swap out if it is dirty/written (Not for I-cache block)
- Again Through Lens
  - Processor can start working after it got the block in the cache
  - Swap out block can be delayed a bit, but have to do
  - Processor Idle time: $\text{Prob}_{\text{miss}} \cdot T$
  - Bus busy time: $\text{Prob}_{\text{miss}} \cdot T(2 - \text{P}_c)$

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?

Read policies

- Sequential or concurrent
  - initiate memory access only after detecting a miss
  - initiate memory access along with cache access in anticipation of a miss
- With or without forwarding
  - give data to CPU after filling the missing block in cache
  - forward data to CPU as it gets filled in cache
Read Policies

Sequential Simple:
- Cache
  - Time
  - TM = \[1 - pm\] + \[pm \cdot (T + 2)\]
- Memory

Concurrent Simple:
- Cache
  - Time
  - TM = \[1 - pm\] + \[pm \cdot (T + 3)\]
- Memory

Sequential Forward:
- Cache
  - Time
  - TM = \[1 - pm\] + \[pm \cdot (T + 1)\]
- Memory

Concurrent Forward:
- Cache
  - Time
  - TM = \[1 - pm\] + \[pm \cdot (T + T)\]
- Memory

Load Policies

Sequential: TM = \[1 - pm\] + \[pm \cdot (T + 2)\]
Concurrent: TM = \[1 - pm\] + \[pm \cdot (T + 3)\]
Sequential Forward: TM = \[1 - pm\] + \[pm \cdot (T + 1)\]
Concurrent Forward: TM = \[1 - pm\] + \[pm \cdot (T + T)\]

Fetch Policies

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

Fetch Policies

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

Replacement Policies: Only for Associative cache

- Least Recently Used (LRU)
  - Add time stamp to each access
- Least Frequently Used (LFU)
  - Newly added line have frequency 0
  - Add Frequency Ctr for each line access
- First In First Out (FIFO)
  - Choose the in order
- Random
  - Randomly choose

Write Policies

Write Through (WT)
- Write into memory now
- Consistence maintained
- Reduced block traffic
- Word traffic introduced
- Better for smaller cache/higher miss rate

Write Back (WB)
- Write into memory later
- No consistence
- Higher block traffic
- No word traffic
- Better for larger cache/low miss rate
Matrix mult.c
int A[8][8], B[8][8], C[8][8];
for(i=0;i<8;i++){
    for(j=0;j<8;j++){
        S=0;
        for(k=0;k<8;k++)
            S=S+B[i][k]*C[k][j];
        if(j==0)A[i][j]=S;//Write in WT Cache
    }
}

Reducing Cache Hit Time

Reducing Hit Time

- Small and simple caches
- Pipelined cache access
- Trace caches
- Avoid time loss in address translation
  - Virtually indexed, physically tagged cache
    - simple and effective approach
    - possible only if cache is not too large
  - Virtually addressed cache
    - protection?, multiple processes?, aliasing?, I/O?

Small and Simple Caches

- Small size => faster access
- Small size => fit on the chip, lower delay
- Simple (direct mapped) => lower delay
- Second level – tags may be kept on chip

Eleven Advanced Optimization for Cache Performance

- Reducing hit time
- Reducing miss penalty
- Reducing miss rate
- Reducing miss penalty * miss rate

**Pipelined Cache Access**

- Multi-cycle cache access but pipelined
- reduces cycle time but hit time is more than one cycle
- Pentium 4 takes 4 cycles
- Greater penalty on branch misprediction
- More clock cycles between issue of load and use of data
  - IF IF IF in pipeline

**Trace Caches: Pre-decoded**

- What maps to a cache block?
  - not statically determined
  - decided by the dynamic sequence of instructions, including predicted branches
- Used in Pentium 4 (NetBurst architecture)
- starting addresses not \( \text{word size} \times \text{powers of 2} \)
- Better utilization of cache space
- downside – same instruction may be stored multiple times