Program Cache Behavior, Miss Categorization and Average Latency

Outline
- Hit/Miss, IPC
- Program Cache Hit/Miss
- Access time

Hashing Vs Caching
- Simple Hashing: Direct Map Cache
  - Example: Array
  - int A[10], each can store one element
  - Data stored in Addr%10 location
- Array of List
  - Int LA[10], each can store a list of element
  - Data stored in List of (Addr%10)th location
  - List size is limited in Set Associative Cache
- List of Element
  - Full Associative Cache
  - All data stored in one list

Cache: Placement
- Direct Mapped
  - Only one tag matching, only index
- Set Associative
  - Both Tag and index matching
- Full Associative
  - Only Tag matching, No index (CAM:Contents Add Mem)

Program Cache Behavior: Hit/Miss

Cache model
- Direct mapped 8 word per line
**Program**

```
int A[128];
for (i=0; i<128; i++) {
    A[i] = i;
}
```

- Assume &A=000000, Behavior of only Data
- Scalar variable [i] mapped to register
- Data have to moved from cache/memory

**Cache perf.: Data Size <= Cache Size**

```
int A[128];
for (i=0; i<128; i++) {
    A[i] = i;
}
```

Scalar mapped to register
Vector mapped to memory


14
16:112 15 A[127]

**Cache perf.: Data Size > Cache Size**

```
int A[512];
for (i=0; i<512; i++) {
    A[i] = i;
}
```

Scalar mapped to register
Vector mapped to memory


14
16:112 15 A[127]

**Cache perf.: Data Size <= Cache Size**

```
int A[512];
for (i=0; i<512; i++) {
    A[i] = i;
}
```

Scalar mapped to register
Vector mapped to memory

32:224 = 64m:448h


14
32:224 15 A[255]

**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];
        A[i][j] = S;
    }
}
```

**Data Size > Cache Size**

- (64+64+64) > 128 words
- When we get into cache it can take benefit
  for (k=0; k<8; k++)
  S=S+B[i][k]*C[k][j];
- Inner loop execute for 64 times
  - We have to get B[i][j] once will have 1miss/7hit
  - C[k] have to bring every time 8miss
  - Total = 7h+9m
- 2nd loop A have one miss in 8 access (1miss/7hit)
  - Total for A= 8m+56h
- Total program: 64*(7h+9m)+8m+56h=504h+584m
- Miss Probability = 584/(504+584)=0.5367
Cache Miss Characterization

Cache Miss Type: 3 C
- Cold Miss/Compulsory misses
  - Those misses caused by the first reference to a datum.
- Capacity misses
  - Those misses that occur regardless of associativity or block size, solely due to the finite size of the cache.
  - There is no useful notion of a cache being "full" or "empty".
- Conflict misses
  - Those misses that could have been avoided, had the cache not evicted an entry earlier.
- Mapping misses: Misses unavoidable given a particular amount of associativity
- Replacement misses: Misses are due to the particular victim choice of the replacement policy.

Average Memory Access Time

- Processor give 1000 reference, 90% hit, 10 miss in cache (miss probability m=0.1)
- Hit time 1 cycle, Miss time = 10 cycle
- AMAT = TimeHit + RateMiss · TimeMiss
  \[ \text{AMAT} = 1 + \text{miss \\_probability} \times \text{miss \\_penalty} \]
  \[ = 1 + 0.1 \times 10 = 2 \]
- Total Mem access time : AMAT · Totalref = 2 · 1000 = 2000 cycle

Multi Level Hierarchy

Memory Hierarchy Analysis

Access time \( t_i \): \[ t_1 + t_2 + \ldots + t_i \] (\( t_i \) at level \( i \))
\[ t_1 < t_2 < \ldots < t_n \]

Hit ratios \( h_i(s_i) \):
\[ h_1 < h_2 < \ldots < h_n = 1 \]

Effective time \( T_{\text{eff}} \):
\[ \sum_i m_i \cdot h_i \cdot t_i = \sum_i m_i \cdot t_i \]

Cache Performance

CPU Exec. Time = (CPU cycles + Mem stall cycles) x cycle time

Mem stall cycles = Number of misses x Miss penalty

Number of misses = IC x Memory accesses x Miss rate

Average memory access time:
- an indirect measure of cache performance
- better than miss rate alone
- not as good as execution time
**Multi-level Caches**

AMAT = Hit time\_L1 \times Miss rate\_L1 \times Miss rate\_L2 \times Miss penalty\_L2

Local miss rate – miss rate of individual caches considering their individual requests:

**Local miss rate**

- \text{L1 Cache: } Miss rate\_L1
- \text{L2 Cache: } Miss rate\_L2

Global miss rate – overall miss rate of caches, considering all requests:

**Global miss rate**

- \text{L1 Cache: } Miss rate\_L1
- \text{L2 Cache: } Miss rate\_L1 \times Miss rate\_L2

Avg. mem. stalls per instr. = Misses per instr\_L1 \times hit time\_L2 + Misses per instr\_L2 \times miss penalty\_L2

**Cache Types**

Instruction | Data | Unified | Split

Split vs. Unified:
- Split allows specializing each part
- Unified allows best use of the capacity

On-chip | Off-chip
- on-chip: fast but small
- off-chip: large but slow

Single level | Multi level

**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

A Sahu