Module: Memory Hierarchy

Reading for this Module

- Basic Cache Principles: Sections 5.1, 5.2, 5.3
- Cache optimizations
  - Reducing miss penalty (Section 5.4), reducing miss rate (Section 5.5), compiler based cache management (Section 5.6), and reducing the hit time (Section 5.7)
Moore’s Law

  - Complexity for minimum component costs increasing at 2X/yr

Technology Trends

- Density of DRAMs have been increasing between 40% - 60% each year quadrupling roughly every 3-4 years
  - Speed growth has been much slower
- CPUs have been doubling in speed every 2-3 years
- Disks recently have been quadrupling in density every 2 years following a 100%/year growth
- Visit the ITRS roadmap at http://public.itrs.net/
The Memory Wall

• We must find ways to hide this performance gap

CPU performance

Memory performance

Gap grows at 50% per year

Moore’s Law Effect

60%/yr

7%/yr

Time

Performance

100000

10000

1000

100

10

1

1

10

100

1000

10000

100000

How soon will applications become memory bound?
– Bandwidth is fully utilized

What can change/slow this trend
– New memory technologies?
– Move away from uniform access time implementations
The Memory Hierarchy

- Large, deep memory hierarchies can push back the effects of the memory wall
- Goal: The memory appears as if it is operating at the speed of the fastest memory at the cost of the slowest

Reference Locality

- Programs exhibit spatial and temporal reference locality in their memory accesses
- Spatial locality: references tend to be clustered in space
- Temporal locality: references to the same memory location tend to be clustered time
The Memory Hierarchy (cont.)

- Decreasing speed
- Decreasing cost/bit

Exploiting Reference Locality

- Note that smaller memories are faster!

- Use a small fast memory, a cache, to hold references that are likely to be used again
  - Subsequent accesses to a referenced item take less time

- Caches are originally conceived as transparent additions to the hardware datapath
Cache Operation

- Memory is organized and managed in units of a block/line
  - Line/block size, word size
- Hit time, hit rate, miss rate, miss penalty

Analysis

- Average memory access time (AMAT) =
  \[ \text{Hit time} + \text{MissRate} \times \text{MissPenalty} = t_c + m \times t_{\text{penalty}} = t_c + (1-h) \times t_{\text{penalty}} \]
  - Assumption: hit time is incurred regardless of hit or miss
- Consider the following assumptions:
  - Cache speed = processor speed
  - Perfect cache with only compulsory misses
  - With non-zero values of \( m \), cache and memory accesses continue to diverge
- When do applications become memory bound?
Cache Organization: Review

- **Design**
  - Placement policies
  - Replacement policies
  - Update policies
  - Allocation policy

- **Analyzing the behavior**
  - Characterization of misses

- **Optimizations**
  - What parameter is being optimized?

Placement Policies

- **Direct mapped caches**
  - Simplest address translation
  - Fastest

- **Fully Associative Caches**
  - Most flexible and expensive
  - Access often on the critical path

- **Set associative caches**
  - Combines the attributes of fully associative and direct mapped caches
Example: Direct Mapped Cache

Example: Fully Associative Cache
Example Set Associative Cache

- K-way associativity
- Extra multiplexer at the outputs

Replacement Policies

- Consider the reference string \( r_1, r_2, r_3, \ldots, r_k, \ldots, r_{n-1}, r_n \).
  - Optimal replacement policy
  - Least recently used (LRU)
  - Most recently used (MRU)
  - First-in-first-out
  - Random

- Policy ideally reflects reference probabilities
Update Policies: Write Through

- Ensures consistency between cache contents and main memory contents at all times
- Write traffic can dominate performance
- Write buffers are used to hide the latency of memory writes by overlapping writes with useful work

Update Policies: Write Back

- Write operations take place in the cache and modified cache lines are marked
  - Modified or “dirty” cache lines written back on replacement
- Locality of writes impacts memory traffic
- Writes occur at the speed of a cache
- Complexity of cache management is increased
- Cache may be inconsistent with main memory
Update Policies and Cache Allocation

- Concurrency in cache operation is limited
  - No updates can be performed until tag check is complete

- Allocation Policy #1: Do not allocate a cache line on a miss
  - Often used with write through designs

- Allocation Policy #2: Allocate a new cache line on a miss
  - Often used with write back designs

Analyzing Cache Misses

- Types of misses
  - Compulsory misses
  - Capacity misses
  - Conflict misses

- Optimizations are often geared to reduce one type of miss, possibly at the expense of another type

- Misses may simply migrate from one category to another
Measuring Performance

- These expressions themselves are an approximation
- Note the equivalence between the use of misses/instruction and misses/memory reference

Improving the Performance of the Cache Hierarchy

- Reductions in miss penalty
- Reductions in the miss rate
- Reductions in the hit time
- Compiler optimizations
Reducing Miss Penalty 1: Multilevel Caches

- Goal: balance (fast) hits vs. (slow) misses
  - Techniques for the former are distinct from those for the latter
- Goal: keep up with the processor vs. keep up with memory

Analysis

\[ \text{AMAT} = \text{Hit\_time}_{L1} + \text{miss\_rate}_{L1} \ast \text{miss\_penalty}_{L1} \]

\[ \text{Miss\_penalty}_{L1} = \text{hit\_time}_{L2} + \text{miss\_rate}_{L2} \ast \text{miss\_penalty}_{L2} \]

\[ \text{AMAT} = \text{Hit\_time}_{L1} + \text{miss\_rate}_{L1} \]
\[ (\text{hit\_time}_{L2} + \text{miss\_rate}_{L2} \ast \text{miss\_penalty}_{L2}) \]

- Local miss rate
  - Defined with respect to the cache
- Global miss rate
  - Defined with respect to the total number of memory references
  - Model the L1/L2 as a single cache
Performance

- Note L2 hit time not that important, why?
- Miss rate behavior of large L2 is indistinguishable from a single cache
  - Global miss rate is a good indicator of performance

Design Issues

- Requirements on Speed
  - L1 coupled with CPU
  - L2 coupled with L1 miss penalty

- L2 design goal
  - Reduce miss rate to main memory
    - Associativity → reduction in conflict misses
    - Size → reduction in capacity misses
  - Match main memory design
Design Issues

- Handling writes
  - Using a write through design with write buffers

- Multi-level inclusion (exclusion)
  - L1 data is always contained in the L2
  - Maintenance requires L1 invalidations by L2
  - Size (cost) constrained designs may use exclusion

Multilevel Inclusion/Exclusion

- Simplifies coherence maintenance
- Increase in miss rate but reduced cost
2. Critical Word First/Early Restart

- Fetch referenced word first
  - Fetching of the rest of the line is performed in the background
- Fetch the cache line but referenced word is forwarded to the CPU when it is fetched
- Gains improve for larger line size
- Complexity of multiple, successive references to the same block

3. Priority of Reads over Writes

- Give reads priority over writes to main memory
- Check for RAW hazards in the write buffer and stall on conflict
- Use of write buffers for write-through design as well as write-back design
  - Overlap writeback with CPU operation
4. Merging Write Buffer (write combining)

- Improving the efficiency of write buffers
- Combine sequential writes into a burst transaction to memory
- Amortize bus transfer startup overhead

Performance of Write Combining

- Close to 90% PCI bus bandwidth utilization on a Pentium Pro
5. Victim Cache

- Reduce the conflict misses of a direct mapped cache
  - Effective backup
- Capture a slightly larger cache footprint
  - Capture recent discards

Reducing the Miss Rate

- Focus on reducing
  - Compulsory misses \(\rightarrow\) e.g., larger block size
  - Capacity misses \(\rightarrow\) e.g., larger cache
  - Conflict misses \(\rightarrow\) e.g., higher associativity

- Trade-off miss rate with
  - Hit time – e.g., higher associativity can increase hit time
  - Miss penalty – e.g., larger block size can increase miss penalty
1: Larger Block Size

- Larger block size increases spatial locality at the (eventual) expense of temporal locality (compare Figures 5.17 & 5.18)
- Reduces compulsory misses but (eventually) increases conflict misses
- Reductions in miss rate are accompanied by increase in miss penalty

2. Increase Associativity

- 2:1 rule
- Trade-off with increased hit time
3. Way Prediction and Pseudo Associative Caches

- **Way Prediction**
  - Use set-associative caches but predict the line in the set
  - Multiplexer is set early
  - Makes the common case fast
  - We can see a natural affinity with I-cache behavior
    - 2-way set associative 21264 I-cache with predictor bit
    - 1 cycle hit vs. 3 cycle hit
  - Activity management for power management

- **Variant is pseudo associative cache**
  - Each set has a fast hit line and a slow hit line (fixed!)
  - Maintenance of a fast hit block requires transfers from the slow hit block
  - Performance degradation due to too many slow hits

4. Compiler Techniques

- **Memory hierarchy exposed to the compiler**
  - We can schedule for execution performance, why not schedule for miss rate or miss penalty?

- **Examples**
  - Re-ordering instructions to improve locality
  - Re-ordering data accesses to improve locality
  - Reduce conflict misses by re-mapping of instructions or data in memory
4.1 Loop Interchange

```
for ( j=0; j<100; j++ )
  for ( i=0; i<5000; i++ )
    x[i][j] = 2 * x[i][j];
for ( i=0; i<5000; i++ )
  for ( j=0; j<100; j++ )
    x[i][j] = 2 * x[i][j];
```

- Improve spatial locality by matching order of traversal with order of storage
  - Principle: maximize the use of data in line before it is discarded
- This optimization does not affect the dynamic instruction count

4.2 Blocking

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    {r=0;
     for (k=0; k<n; k++)
       r = r + y[i][k] * z[k][j];
     x[i][j] = r;
    }
for (jj=0; jj<N; jj = jj+B)
  for (kk=0; kk<N; kk = kk+B)
    for (i=0; i<N; i++)
      for (j=jj; j<min(jj+B,N); j++)
        {r=0;
         for (k=kk; j<min(kk+B,N); k++)
           r = r + y[i][k] * z[k][j];
         x[i][j] = x[i][j] + r;
        }
```

- Restructure the loops to improve
  - Fit in the cache
  - Improve temporal locality
- Solutions now become machine dependent
4.2 Blocking (cont.)

- What is the miss behavior?
- Decompose the computation to operate on BxB blocks such that three blocks fit in the cache
- Reduce the overall number of worst case misses by a factor of B

4.3 Loop Fusion

```c
/* 3 sequential arrays of same size*/
int a[SIZE];
int b[SIZE];
int c[SIZE];

for (i=0; i<SIZE; i++)
    a[i] = b[i];
for (i=0; i<SIZE; i++)
    c[i] = a[i] + K;
```

- Reduce conflicts between the two arrays
- Exploit temporal locality across arrays
- Not always obvious whether loops can be fused
- What about merging the arrays into a single structure?
4.4 Compiler-Controlled Pre-fetching

- Register or cache pre-fetch instructions
- Behavior of the pre-fetch instructions: faulting vs. non-faulting
- Investment should be worthwhile
- Optimizations (hints) to allocate-on-write but not load!
- What about pre-fetching in the presence of pointers?

```c
for (i=0; i<3; i++)
    for (j=0; j<100; j++)
        a[i][j] = b[i][0] * b[i+1][0];
```

```c
for (j=0; j<100; j++)
    prefetch(b[j+7][0])
prefetch(a[0][j+7])
a[0][j] = b[j][0] * b[j+1][0];
```

```c
for (i=1; i<3; i++)
    for (j=0; j<100; j++)
        prefetch(a[i][j+7])
a[i][j] = b[i][0] * b[i+1][0];
```

5. Non-blocking caches

- Continue operation through a miss
- “hit under miss” or “miss under miss”
- Hide miss penalty effectively reducing impact on CPI → latency hiding
  - Implies the memory system can service multiple misses concurrently
- Handling multiple misses to the same line
- Complexity of the cache controller and memory system interface increases

![Graph showing percentage of benchmarks with hits under misses and latencies]

© 2003 Elsevier Science (USA). All rights reserved.
6. Hardware Pre-fetching

- Hardware engines to pre-fetch into stream buffers resident outside the cache
  - i+1 pre-fetch
  - Profile driven pre-fetch
  - Run-time statistics to drive pre-fetch mechanisms

Reducing the Hit Time: 1. Small, Simple Caches

- Components of a hit
  - Decode time → related to cache size
  - Tag match
  - Line access and word extraction
  - Cache size → can it fit on chip?
- For L2 caches keep tags on chip?
Hit Time Analysis

- Direct mapped caches tend to 1.2 - 1.5 times faster than two-way set associative caches
- Note trends in L1 cache size
  - Speed not keeping up with processor pipelines
  - Size growing slowly or staying constant to keep hit times low

Pipelined Caches

- Pipeline the cache to multiple levels
- Slower hits but increased instruction bandwidth
- Impact
  - Penalties due to hazards increases
  - Exception handling can be complicated
Summary

- Optimizations focused in three main cache attributes
  - Miss penalty
  - Miss rate
  - Hit time

- General strategies include
  - Latency tolerance
    - Hit latencies by overlapping misses with useful work
  - Concurrency of operation
  - Focus on each of the parameters of the expression for cache access time
  - Compiler strategies vs. hardware strategies

Study Guide

- Given a cache design produce the breakdown of the address bits used to address the cache

- Given a cache directory state, modify this directory state given a sequence of addresses and the corresponding read/write operation including the implementation of
  - Replacement policies
  - Update policies
  - Allocation policies

- Computation of average memory access time for various designs and policies

- Computation of impact on CPI of cache misses
Study Guide

• Given a complete memory system design understand the design
  – Depending on the data given, compute the number of sets, associativity, address breakdown at each level of the cache hierarchy

• Given a sequence of memory addresses and a specific optimization, such as a victim cache, be able to update the contents of the cache directory (tags and state such as dirty bit, valid/invalid bit) and any associated data structures

• Assess the impact of the memory hierarchy on the CPI
  – Single and multilevel caches
  – Be able to translate given miss data from misses/reference to/from misses/instruction. Similar ability for stall cycles

• Assess the impact of specific optimizations (given the relevant data) on the CPI, average memory access time, and stall cycles

Study Guide

• Given a reference pattern update the tags in a multilevel cache hierarchy
  – To preserve multilevel inclusion/exclusion

• Given a reference pattern be able to choose amongst a set of optimizations
  – That will minimize miss rate

• Understand each and every optimization
  – Why it works
  – How it reduces memory stall cycles
  – Assumptions about program memory reference behavior (if any)

• Compute the impact of compiler optimizations, e.g., placement of pre-fetch optimizations
  – On miss rate
  – CPI impact