Caches

Today's topics:

Basics
memory hierarchy
locality
cache models
associative options
calculating miss penalties
some fundamental optimization issues

The Problem

• Widening memory gap
  • DRAM latency CAGR = 7%
  • CPU performance CAGR
  > 25% prior to 1986, 52% 86-2005, 20% thereafter

The Solution

• Deepen memory hierarchy
  • make dependency on DRAM latency the rare case
    • If you can
      • multi-ported
      • single ported

Balancing Act

• As always
  • performance vs. cost, capacity vs. latency, ...
  • on-die SRAM is expensive per bit
    • latency depends on size and organization
    • area - 6T/bit
    • power - not so bad since only small portion active/cycle
  • DRAM
    • latency is awful in cycles w/ GHz clock speeds
    • area = 1T + 1C/bit - lots of details later
  • Dealing w/ cache size latency
    • deepen cache hierarchy
      • small separate IL16 and DL18: minimize mem structural stalls
      • unified L2 reduces fragmentation problems
      • multicore introduces global L3
      • may not continue to be a good idea
      • everything runs out of steam at some point
        • buy is what % of L1 misses hit in L2 (induction on Le & Lau1)
Locality

2 basic types
- temporal
  » if you used this block then you will use it again soon
- spatial
  » if you used this block you will use a nearby one soon

Exploiting locality
- match of application memory usage and cache design
- some codes are touch once
  » encrypt/decrypt, encode/decode, ...
  » here caches are a liability
  » why?

Exploiting locality
- match of application memory usage and cache design
  » if you match you win - simple as that
  » if you match you win - fall through misses
    » e.g. encrypt/decrypt, encode/decode, ... (media influence)
  » here caches are a liability
    » you have to swing to miss
      - try L1 - miss? - try L2 - miss? ...
      - a lot of extra time if you end up going to DRAM anyway
  » historical tidbit
    » Seymour Cray didn’t believe in caches or DRAM
      » or even 2’s complement initially

Performance Issues

Basics
CPUtime = IC * CPI * cycle_time
CPI = 1 / frequency
IPC = 1 / CPI
CPUtime = IC / (IPC * frequency)

Enter stalls
- IPC gets degraded by stalls
  » we've seen stalls due to hazards and pipeline issues
  » now the focus is on memory stalls
  » influence is based on load & store %
  » with good branch prediction most stalls are memory induced

Computing Memory Effect

Misses induce stalls
- ILP & TLP can overlap some of the penalty
  » but a miss is a surprise so some of the penalty won't be hidden
XEQtime = (CPUcycles + MEMstallcycles) * cycle_time
MEMstallcycles = num_misses * miss_penalty
MEMstallcycles = IC * misses * instruction
MEMstallcycles = IC * memory_accesses * miss_rate * miss_penalty
Separate
MEMstallcycles = \sum \text{READstalls,WRITEstalls}
READstalls = IC * reads * read_miss_rate * read_miss_penalty
WRITEstalls = IC * writes * write_miss_rate * write_miss_penalty
4 Questions = $ Organization

- **Q1:** Where can a block be placed? (examples shortly)
  - fully associative: answer = anywhere
  - direct mapped: answer = only one place
  - set-associative: in a small number of "ways"

- **Q2:** How is a block found?
  - 2 types of tags
    - status: valid and dirty (for now)
    - address: alias chance must be resolved
      - set size

- **Q3:** Which block is replaced on a miss?
  - LRU – best but expensive – matches temporal locality model
  - FIFO – good but expensive – LRU but on 1st touch not used
  - random – defeats temporal locality but simple to implement
  - approximation by epochs – add "use" status tag

- **Q4:** What happens on a write miss?
  - hit: write-through or write-back (requires dirty flag)
  - miss: write-replace or write-around (modern CPU's – allow both)

Cache Components

- **Cache block or line**
  - size varies – 64B is a common choice
    - no reason why the block size can't be larger the deeper you go
    - in the memory hierarchy
      - cache lines typically the same size – reduces some complexity
      - memory to cache transfer in line sized chunks
      - disk to memory transfer in page sized chunks

- **2 main structures & a bunch of logic**
  - data RAM – holds the cached data
  - tag RAM – holds tag information
    - same number of "entries" as data RAM
      - entry = line for direct mapped or fully associative
      - entry = set for set-associative
    - width for set associative a number of ways
      - for each set of address tags
        - status tags present as well

Block Identification

- **Address**
  - Block address
  - Index
  - Offset

- **tag = address tag**
  - held in tag ram
  - size is what's left over after index and block offset size

- **Index**
  - \( \log_2(\text{number of data RAM entries}) \)

- **block offset**
  - says which byte, word, or half-word is to be moved to the target register
  - silly is a way - word or double transferred to register
  - appropriate byte or half-word is then used for the op
  - size = \( \log_2(\text{line size}) \)

- **increase offset or Index reduces tag size**

Cache Access

- **Byte address**
  - 101000 (data)

- **Offset**
  - 8-byte words
  
- **8 words: 3 index bits**

- **Alias problem?**
  - Data array
  - Sets

Direct-mapped cache: each address maps to a unique address
**De-Alias by Matching Address Tag**

- Direct-mapped cache: each address maps to a unique address.

**Set Associative Cache**

- Set associativity → fewer conflicts; wasted power because multiple data and tags are read.

**Miss Types**

- 3 C's (for now – a 4th will show up later)
  - **Compulsory**
    - 1st access to a block will always miss
    - fix: prefetch if you can
  - **Capacity**
    - If line that was previously in the cache is evicted and then reloaded
    - Indication that working set size of app is bigger than the cache
    - fix – bigger cache or prefetch
  - **Conflict**
    - e.g. only need 2 lines but they victimize each other
    - how can you tell the difference between capacity and conflict misses?
Increasing Associativity

- Data and Tag RAM #_entries the same
  - will depend on capacity
- Logic involved in compares
  - for n ways → n parallel compares
    - If one of the succeeds then hit in the associated way
    - If none succeed then miss
    - If >1 succeed somebody made a big mistake
    - If n is large then problems → way prediction
- fully associative
  - huge number of parallel compares (m line capacity → m compares)
    - power hungry - limits use to smallish caches
    - or save power but increase hit-time
    - n compares where n << m
    - walk tag array in n sized chunks
    - step when you find a hit but miss_time is VERY long
    - variable miss_time but in essence this is always true anyway

Organization Effects

Increasing associativity and capacity helps but there are trade-offs
Increasing either increases both power and delay
Need to find the sweet spot

Optimizations to Reduce Miss Rate

- Increase block size
  - +: reduces tag size, compulsory misses, and miss rate if there is spatial locality
  - -: miss must fetch larger block, no spatial locality → waste, increases conflict misses since for same capacity there will be less blocks in the cache
- Increase cache size
  - +: reduces conflict and capacity misses
  - -: larger caches are slower – increased hit time
- Increased associativity
  - issues already discussed
  - rule of thumb 2-way associativity w/ capacity N/2 has the same miss rate as 1-way size N cache
- Way prediction
  - can use methods similar to branch prediction
    - get it right and reduce power consumption since
    - 1 way SA = direct mapped

Hiding/Tolerating Miss Penalty

- OOO execution
  - combo of ILP and TLP techniques
    - do as much as you can in between a load and consumer
- Non-blocking caches
  - first miss doesn’t block subsequent actions
  - cache controller keeps track of multiple outstanding misses
    - MSHR’s – miss status handling registers & dynamic issue req’d
- Write buffers
  - prioritize reads – handle writes when memory is otherwise idle
    - opposite of Itanium ALAT concept
    - reads must check write buffer to get latest result
- Prefetching (even if you get it right)
  - too aggressive → increased cache pressure
  - possible to increase conflict/capacity misses
The Problem w/ Addresses

- Program vs. Memory
  - program – virtual addresses
  - main memory and I/O land – physical addresses
  - what does the cache use?

- Physically indexed physically tagged cache
  - result – increased hit time
    - not a good idea according to Amdahl if you hit most of the time
  - why?
    - virtual address needs to be translated to a physical address
      before the cache access can even begin
    - TLB is a cache of these translations
    - miss in the TLB goes to the page table
      - which may be in main memory or even on the disk
      - UGHly
  - improve by doing address translation in parallel with data access
    - a bit speculative but if high hit rate then it's the right choice

Virtually Indexed Physically Tagged $

Virtual address space can be > physical addr. space

Issues and Options

- VI-PT $
  - page size inherently limits size of the L1$
    - higher associativity can mitigate this BUT
    - delay and/or power may go up

- VI-VT $'s
  - just do everything virtual
    - note these caches do exist so it's not hypothetical hokum
  - removes page size cache capacity problem
  - problems?
Issues and Options

- VI-PT
  - page size inherently limits size of the L1
    - higher associativity can mitigate this BUT
      - delay and/or power may go up
- VI-VT
  - just do everything virtual
    - note these caches do exist so it’s not hypothetical hokum
      - removes page size cache capacity problem
  - problems
    - multiple processes concurrently running
      - OS guarantees address space privacy
  - fix
    - assign process to an address space ID
      - address space part of tag RAM – space ID provided w. virtual address
    - processes are not interleaved like threads so process ID is known
      - on context switch OS must flush cache if AS-ID isn’t active

Concluding Remarks

- This lecture – somewhat remedial
  - but essential to understand what follows
    - if you don’t
      - read the book more thoroughly or go back to the cs3810 text
      - ask questions, confer w. Dogan, ...
    - mid-term questions will be conceptual
    - subsequent homework will be more substantive
- Applies to reality as well
  - lots of cores complicate cache design
  - foundation however is the same