Memory Access Challenges

Performance Impact of Caches

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
Recall: Cache Hierarchies and Performance

- Cache is **fast** memory, typically on chip
  - DRAM is off-chip
- It has to be **small** to be fast
- It is also more **expensive** than DRAM on per-byte basis
- Idea: bring frequently accessed data in the cache
Why and How Does a Cache Help?

- **Temporal and spatial locality**
  - Programs tend to access the same and/or nearby data repeatedly

- Size of cache

- Multiple levels of cache

- Performance impact of caches
  - Designing programs for good sequential performance
Cache: Parameters and Metrics

• Cache line: spatial locality
  • 64 bytes or 128 bytes, consecutive memory addresses
  • Make it too short: what happens?
  • Make it too long: what happens?

• Hit ratio
  • Fraction of times a data item is found in cache
  • Best possible: 1
  • May be higher than that given by reuse ratio because of spatial locality
Cache: Parameters and Metrics

- Cache line: spatial locality
  - 64 bytes or 128 bytes, consecutive memory addresses
  - Make it too short: what happens? Too many cache misses, ..
  - Make it too long: what happens? Potential waste if not that much spatial reuse

- Hit ratio
  - Fraction of times a data item is found in cache
  - Best possible: 1
  - May be higher than that given by reuse ratio because of spatial locality
Cache Mapping and Associativity

• Say you have 64 KB of cache, 64 byte cache lines
  • So, 1024 cache blocks in all
• On a read miss, you have to fetch a cache line from memory
  • (Or from next level cache)
  • Where in the cache should it go?
    • I.e., which of the 1024 blocks should be used to store it?
    • A fixed place based on the address: direct-mapped cache
    • Anywhere: fully-associative cache
      • Expensive (hardware and time!)

Solution: Set associative cache
Set Associative Cache

Program’s memory: a sequence of cache lines

Where in the cache can the shaded one go
(if it needs to be brought into the cache)?

Direct mapped

Set associative: 4 sets of 2 cache lines each

Fully associative
Cache Replacement Policy

• Which cache line to evict?
• For direct mapped cache, there is only one candidate
• For associative caches, multiple choices exist, so a policy is needed
  • Random
  • LRU: least recently used
  • What do you think will be a good policy?
• But no significant impact on performance across applications is observed...
Categorizing Cache Misses

• Compulsory misses:
  • “Cold” cache accesses, i.e., first time you access it

• Capacity misses:
  • There is no space in cache

• Conflict misses:
  • There is space, but it is not in the right set

• The idea of a “working set”
  • A set of addresses that the program accesses over a time window

• Question: Which misses cannot happen with fully associative cache? Conflict misses
for (i = 0; i < n; i += M)   // M is the cache-line size in words
A[i] = B[i] + C[i];

• What happens with:
  • Direct map cache?
  • Fully associative cache?
  • 4-way associative cache?

• How many cache misses?
• What if the loop repeats?
  • I.e., there is an outer loop
  • Working set: the data in A, B, and C
for (i = 0; i < n; i++). sum = sum + A[i];

• How many cache misses?
• What kind of misses are those?
• Can we reduce the cost of those misses? How?
• Answers:
  • One for every cache line... N/k, where k is the size of a cache line in words
    • So, 64 byte cache line, single precision words of 4 bytes: k = 16

• These are all compulsory misses
  • First time we access the data
Imagine a sequential program running using a large array, A
Values in the indirection array are all between 0…size-1
How long should the program take, if each addition is \( a \) ns?
What is the performance difference you expect, depending on the value of size?

```c
for (i = 0; i < repetitions/size; i++)
    for (count1 =0; count1 < size ; count1++)
```
Other Architectural Innovations

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
Prefetch

• Look for pattern in memory accesses:
  • That you are accessing successive memory locations in A
  • If I make the hardware detect this pattern, what can I do?

• Two techniques:
  • Bring multiple cache lines in on a miss
  • Prefetch even more lines... asynchronous load instructions
Out of Order Execution

• Keep many instructions “in flight”
• If there are data dependencies, buffer the instruction
• Important techniques: Register renaming, scoreboard, etc., to satisfy dependencies and ensure correctness
• Also, try to issue multiple instructions per cycle!
Superscalar Processors

• Can we execute multiple instructions per cycle?
  • FMAD was a single instruction

• Idea: Fetch multiple instructions in each cycle...
  • And execute as many of them in parallel as is possible without violating data dependencies
  • Need more hardware on chip
  • More registers for buffering results
  • Multiple load units, etc.

• IPC: instruction-level parallelism
  • How many can you execute in each cycle?
  • Typically 2-3 is the max
Impact on programming

• Usually, we should let the compiler handle this

• But its useful to know whether our program is exploiting the superscalar processors well
  • IPC : instructions per cycle --- this is a useful metric to track

• Try to write code such that consecutive instructions can be executed concurrently
  • Avoid dependences.. But these may be unavoidable
  • ILP (Instruction level parallelism)

• Avoid pipeline bubbles
Performance and Complexity
Additional Issues: Paging, Compilers, ...

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
Multiprogramming and Virtual Memory

• A computer runs many programs “simultaneously”
  • It gives each program a time slice of some milliseconds (e.g., 10ms)
• How can it keep their memories from mixing up?
• Answer: virtual memory
  • Each program thinks it has the whole processor to itself, and can address memory from 0...X gigabytes
  • This “virtual” address is mapped to a physical address by the hardware and operating system via “page tables”
  • Each page has (say) 4 kilobytes of data... It may or may not be in physical memory
  • If it is not in memory, it is kept on disk (in “swap” space)
  • The mapping itself is stored in memory!
    • But most commonly/recently used entries of this map are stored in hardware: TLB or Translation Lookaside Buffer
Load Instruction

(Morris, 1998)
What Is a Good Page Size?

• In most OSs it is set at 4 KB
• That’s fine for multiprogrammed machines
• For a dedicated parallel computer?
  • HugePages option: 2 MB pages, 4 MB pages
  • On BlueWaters, simply load the 2 MB page module before compilation and execution
  • Loading a module redirects mallocs to use larger pages
Compilers

• Capabilities
• Black-box nature
• Optimization levels: -O1, -O2, -O3…
  • Higher level of optimization is better for performance, but occasionally (rarely) may compile wrong!
  • Lower levels make debugging easier
• Flags
  • More specialized, nuanced, control over what kinds of optimizations the compiler may try
  • Debug information retained
Bottom Line?

• The speed increase has come at the cost of complexity
• This leads to high performance variability that programmers have to deal with
• It takes a lot to write an efficient program!