Energy-Aware Computing

Lecture 10: Power-aware caches
Outline

• Value compression in caches
• Reducing cache parallel activity in associative caches
  – Phased cache
  – Way prediction
  – Way selection (halting)
• Controlling cache idle capacity
  – Selective cache ways
• Saving snoop power
Value compression

Dynamic zero compression

• Compress zero bytes using a tag bit per byte (zero indicator bit)
• Detect zero when storing and only store the ZIB, the bitlines are not driven
• Byte word line disabled when ZIB = 1
• Special sense amps force output byte to 0 when ZIB=1
DZC key circuits

Villa, Zhang, Asanovic
Dynamic zero compression for cache energy reduction, MICRO-33

UoE/Informatics
Energy-aware computing
DZC evaluation

- Spice simulation from extracted layout of a 16KB cache in 0.25um
- Array of ZIBs adds 9% area
- Read
  - Latency overhead 2FO4 delays
  - D$ 26% reduction in power
  - I$ 10% reduction in power
- Write
  - No delay; zero detection overlaps with tag access
  - Write energy 1/5th for 0x0 compared to non-zero
Other compression methods

- Simple compression scheme required
- Frequent value locality
  - Small number of distinct values appear frequently in memory accesses
- Dictionary based approaches
  - Keep a table of frequent values
  - Store index to dictionary in place of value
- Pack more lines into a cache!
  - Increases the capacity, hit rate
Activity in associative caches

• Associativity is desirable, if it comes cheaply
  – It improves the hit rate
• The standard RAM-tagged implementation is not power efficient
  – We know that at most one data bank will contain the address, but all are accessed
• A number of activity reducing methods have been proposed
Phased caches

- Conventional 4 way associative
- Phased
  - Full tag access required
  - On a hit, only one of the data sub-blocks is accessed
  - Significant power savings at the expense of 1 additional cycle for hits
    - Can be pipelined
  - Similar to CAM tagged caches
Way prediction

- Conventional 4 way associative
- Way predicting
  - Extra prediction unit required
  - On correct prediction, only 1 way (tag, data) accessed
  - On incorrect prediction, energy is as with standard cache
    - But slower
Way prediction evaluation

• For I$, way prediction accuracy is 96% on average, over 90% for all benchmarks
• For D$, average accuracy is 86%
  – Half benchmarks over 90%
• Improved EDP by 60% (D$) to 70% (I$)
• Compared to phased cache
  – Similar power savings
  – Approx half average cache access time
Way prediction problems

• High associativity reduces miss rates
  – Power efficient
• But the success rate of way-prediction drops with increasing associativity
  – More options to choose from
  – Not a serious problem for L1 caches, but increasingly important for other levels
• Cache misses on way-predicting caches are very slow as all options are explored before a miss is declared
Way-selection

• Deterministically select the right way
  – No prediction is made
• Location cache
  – Hold way info for L2 $ next to L1
  – Update location $ for L1 replacements
• Way-halting
• Decaying Bloom filters
Way-halting

- Tag storage split btw RAM, CAM mems
- Quick partial look up of tags (using CAM)
- All mismatches halt tag and data array accesses
- 2% area overhead, no performance penalty, 45-60% energy reduction
Decaying Bloom Filters

• Use a leakage power saving method to save dynamic power
  – Decay (switch off) inactive cache lines to save leakage power
• No need to search decayed lines
• A Bloom filter tracks which ways *may* hold a line
• Parallel access to all such lines
  – rather than 1 with way-prediction
Selective cache ways

- Not all cache needed by all programs all the time
- In sub-banked caches, “ways” are implemented as independent banks
- Easiest (only?) way to adapt cache dynamically: disable one or more ways
  - No accesses to data array, no prechanging,…
  - Tags remain active
Selective cache ways

- Dirty data in disabled ways
  - Temporarily enable on hit or coherence access
  - Flushing to L2 would be expensive
- When to adapt the cache size
  - Software control
Saving snoop power

- Multiprocessor caches stay coherent by snooping all other accesses
  - Every miss in a cache, searches all other caches
- 54% of all snoops miss in the L2 tags 4-processor SMP SPLASH-2 benchmarks
  - Wasted power searching tags
Jetty – snoop filtering

• Filter out unneeded snooping
• Bloom filter – efficient structure for non-membership test
  – to determine if item not in set
• Three Jettys:
  – Exclude: “I’ve seen this before and I am sure it’s not locally cached”
  – Include: captures a superset of cached data
  – Hybrid: check both (eliminates 76%)
Recent work on caches

• Non-uniform cache access (NUCA)
  – Access to banks that are far away from the core are slower than to banks that are near
    • Either operate at slowest speed or expose the non-uniformity to the processor
  – Caching within the cache: keep frequently used data in nearest bank

• Sharing a number of banks between cores in CMPs
  – Dynamically adjust the size of each core’s cache
  – Used in L2 of Intel multicore chips
Summary

• Value compression
  – Zero value compression
• Cache activity reduction
  – Phased cache
  – Way prediction
  – Way selection (halting)
• Controlling cache size
  – Selective cache ways
• Saving snoop power