Computer Systems Organization

Courant Institute
TA session 3
Mahla Rahiminejad
Outline

• Cache
• cache replacement policies
• Valgrind
• cache simulator
• Matrix Traverse
• Cachelab
Cache

• What is a cache?
  • a block of memory for temporary storage of data
  • small and fast
  • used to reduce access time of CPU

• What makes them useful?
  • Temporal locality: If a is used at time a, it's so probable to be used in near future.
  • Spacial locality: if a is used, it's so probable that the items near a being used.
Cache (cont'd)

• Accessing a data in cache:
  • the item is already in cache:
    • Cache Hit
  • the item is not in cache and cache is not full:
    • Cache Miss, bring it from memory into cache
  • the is not in cache and cache is full (worst case):
    • Miss eviction, discard an element and replace it with new value

• We have 3 kinds of caches:
  • direct-mapped cache
  • Fully associative cache
  • set-associative cache
Cache (cont'd)

• Direct-Mapped Cache:
  • Highest speed CPU
  • like a hash table with simple hash function
  • array of slots
  • can access each slot with an address
• Drawbacks:
  • high risk of collisions
  • may have high miss-rate
Cache (cont'd)

- Set-Associative cache:
  - hybrid between direct-mapped and fully-associative cache
  - same as direct-mapped: array of slots
  - difference: each slot groups into sets
    - extra bits in address line for defining set number
  - like a matrix:
    - rows: sets
    - columns: blocks
  - Use replacement policies for eliminating collisions
Cache (cont'd)

• Fully-associative cache:
  • Brings a block of memory of size $2^M$ into cache
Cache replacement policies

• When cache is full, we should evict one cell
  • which one should we evict?

• Methods:
  • LRU: Least Recent Used:
    • Discards the least recently used item
  • MRU: Most Recent Used
    • Discards the most recently used item
  • RU: Random Replacement
    • discards an item randomly and replaced it
Cache simulator

• A tool (piece of code) for simulating a cache behavior
• Input: cache characteristics including:
  • number of sets
  • size of blocks
  • a memory trace file
• Output: cache behavior:
  • num of misses
  • num of hits
  • num of miss evictions
Matrix transpose

• An algorithm that changes the rows and columns in a matrix
• For a given matrix A with M rows and N columns:
  • B is transpose matrix of A if:
    • for i from 1 to M:
      • for j from 1 to N:
        • A[i][j] = B[j][i]
• How is a matrix stored in memory?
  • an array of pointers with size N, points to N arrays of size M

• Main Question:
  • In a matrix transpose function, how can we reduce number of misses when transferring elements of A into cache?
  • What should be the characteristics of the cache? (# of sets, blocks, ...)

A
\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}
\]

A^T
\[
\begin{array}{ccc}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 9 \\
\end{array}
\]

A
\[
\begin{array}{ccc}
1 & 4 & 3 \\
8 & 2 & 6 \\
7 & 8 & 3 \\
4 & 9 & 6 \\
7 & 8 & 1 \\
\end{array}
\]

A^T
\[
\begin{array}{ccc}
1 & 4 & 3 \\
8 & 2 & 6 \\
7 & 8 & 3 \\
4 & 9 & 6 \\
7 & 8 & 1 \\
\end{array}
\]
Valgrind

- A tool for tracing data accesses to memory
- Dynamic analysis tool
- Good for memory management and thread tracing
- Three kind of memory access:
  - L: data load (load data from memory and move it into cache)
  - M: data modify (change data in cache, write through/back it into memory)
  - S: data store (store data from cache into memory)
  - I: instruction load (load instruction from memory)
Valgrind (cont'd)

• valgrind ubuntu command:
  • valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l

• sample valgrind output:
  • L 10,4 (read 4-bytes of data from address 10)
    • if it's in cache: hit; otherwise, miss or miss eviction, bring it from memory and put it in cache
  • S 18,4 (store 4-byte data in address 18 of memory )
  • M 20,4 (Modify 4-byte data in address 20)
  • I 11, 8 (load 8-byte instruction in address 11 of memory)
Project steps

• Download valgrind in ubuntu:
  • download command:
    • sudo apt-get download valgrind

• Get Project file from following link:
  • http://cs.nyu.edu/courses/fall14/CSCI-UA.0201-003/cachelab-handout.tar

• Extract project file in ubuntu
  • extract command:
    • tar -xvf cachelab-handout.tar
  • **Attention**: Do not extract it with unzip, etc.
Project steps (cont'd)

• **Step1:** design a cache simulator:
  - input: num of sets, blocks, a memory trace file from valgrind
  - output: num of hits and misses
  - cache replacement policy: LRU
    - Consider a time for each cell of cache (it can be a simple counter)
    - When cache is full:
      - Find min of counters
      - Evict that cell

• **Step2:** write an optimized matrix transpose function
  - input Matrix A[M][N]
  - output Matrix B[N][M]

• **Step3:** evaluate your matrix transpose function in cache simulator
  - run valgrind and trace memory when running your matrix transpose
  - run cache simulator on the trace file
  - Is your matrix transpose efficient? (miss rate < 500?)
Project steps (cont'd)

• Project files:
  • cachelab.c: you should write your cache simulator code in it.
  • cachelab.h: don't change it.
  • traces folder: sample trace files from valgrind for testing your cache simulator
  • test-trans: for testing matrix transpose function
    • ./test-trans -M 32 -N 32
  • csim-ref: running cache simulator on a trace file generated from valgrind
    • ./csim-ref -v -s 5 -E 1 -b 5 -t trace.f0
References

• http://cs.nyu.edu/courses/fall14/CSCI-UA.0201-003/cacheland.pdf
• http://en.wikipedia.org/wiki/Cache_(computing)
• http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Memory/set.html
• http://en.wikipedia.org/wiki/Valgrind
• http://people.inf.elte.hu/karuaai/szamalap/Fully%20Associative%20Cache.jpg
Good Luck! :)