ECE472-Assignment 3

Mohannad Al Arifi

November 20, 2015

- Brief architecture overview
- Optimizing for (code and) data cache
  1. General suggestions
  2. Data structures
- Aliasing

0.1 cache review

- 32/64 bytes cache lines
- N-way set-associative: logical cache line → N physical lines. Helps minimize cache line thrashing.
- Foes:
  1. Compulsory misses: Data read for first time.
  2. Capacity misses: small cache space for all active data. Too much data accessed inbetween successive use.
  3. Conflict misses: cache thrashing because data mapping to same cache lines.
- Friends:
  1. Rearrange (code, data): to increase spatial locality.
  2. Reduce (size,# cache lines read): smarter smaller formats, compression.
  3. Reuse (cache lines): increase temporal and spatial locality.
- pay attention to:
  1. Profile
  2. study the generated code
  3. locality
  4. size

0.2 data cache optimization

- prefetching/preloading data
  1. Software prefetching
     (a) Not-too-Early
     (b) Not-too-Late
     (c) Greedy
  2. Hardware prefetching
     (a) Hit-under-miss
- Cache-conscious structure layout
  1. Field reordering: e.g. Likely accessed together so store them together.
  2. Hot/cold splitting

3. Compiler padding: watchout and store in decreasing order. *Good one.

- Cache performance analysis:
  1. Usage patterns:
     (a) Activity: hot/cold field
     (b) Correlation: for field reordering
  2. Logging tool
- Tree data structures
  1. Rearrange nodes: increase spatial locality
  2. Reduce size: pointer elimination, compression.
  3. Breadth-first order: requires storage for complete tree of height H
  4. Depth-first order: stores existing nodes.
- Linearization caching
  1. linear data: best spatial locality, easily prefetchable.
  2. So linearize at runtime.
- Memory allocation
  1. Allocate from pools not heap. *interesting.
  2. Free ASAP, reuse immediately.
- Aliasing/anti-aliasing
  1. Aliasing is multiple references to the same storage location.
  2. Free ASAP, reuse immediately.
- restrict-qualified pointer
  1. Restricting can help perform parallel operations not possible otherwise.