ESE532: System-on-a-Chip Architecture

Day 17: October 30, 2017
Associative Maps, Hash Tables

Today

- Motivation/Reminder
- Hardware Associative Memory
- Software Maps
  - Hash Tables
- Hardware Maps
  - Associative Memories from BRAMs
  - Multi-hash Hash tables from BRAMs

Message

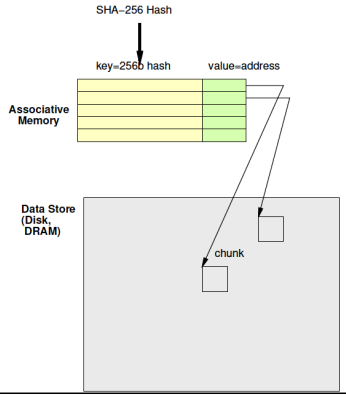
- Rich design space for Maps
- Hash tables are useful tools
- Multiple hashes are better than one

Reminder from Last Time

Deduplicate

- Compute chunk hash
- Use chunk hash to lookup known chunks
  - Data already have on disk
  - Data already sent to destination, so destination will know
- If lookup yields a chunk with same hash
  - Check if actually equal (maybe)
- If chunks equal
  - Send (or save) pointer to existing chunk

Deduplication Architecture



Associative Memory

- Maps from a key to a value
- Key not necessarily dense
  - Contrast simple RAM

Memory Tree Algorithm

```
curr – pointer into input chunk
// follow tree
y=0;
while(encode[x][input[curr]]!=NONE)
    x=encode[x][c]; y++;
If (y>0)
  send <x,y>
  send input[curr]
  encode[x][input[curr]]=curr
```

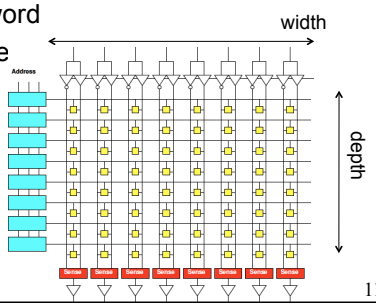

Compact Memory

- int encode[SIZE][256];
- Table is very sparse
- Store as associative memory
  - At most SIZE entries

Custom Hardware Associative Memory

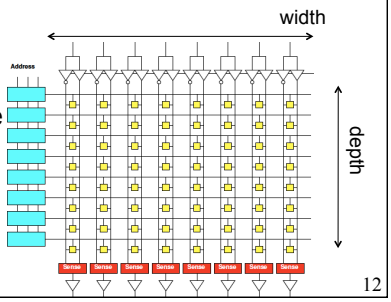
Memory Block Review

- Match on address
- Select wordline for a row
- Reads out a word
- Address dense and hardwired
- One row for each 2^{Abits} values

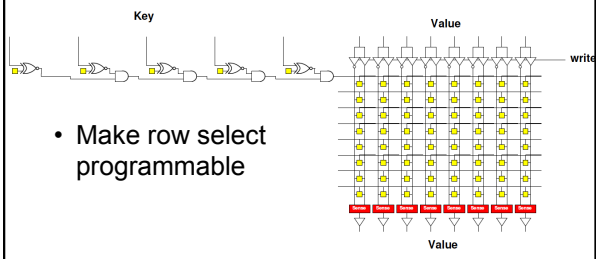


Memory Block Review

- Want address as key
- Word is value
- Key sparse
- Rows<2^{kbits}
- Key programmable

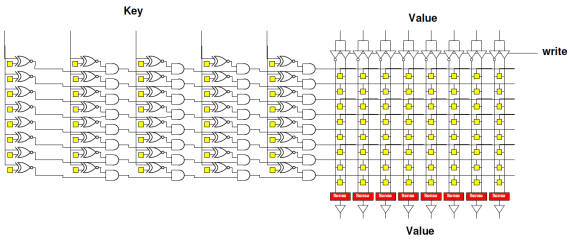


Programmable Key

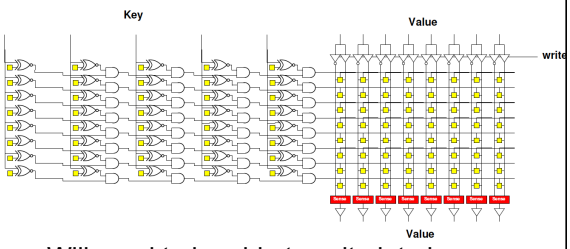


- Make row select programmable

Associative Memory Bank



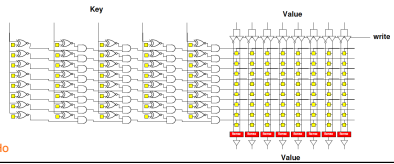
Associative Memory Bank



- Will need to be able to write into key
  - Another "fixed" decoder to generate key-word line for programming

Associate Memory Cost

- More expensive than equal capacity SRAM memory bank
  - Memory cells in decoder
  - No sharing of AND-terms in decoder
  - Need to support write into key



Software Map

Software Map

- Map abstraction
  - void insert(key,value);
  - value lookup(key);
- Will typically have many different implementations

Map Implementations

- How could we implement Map in software?

Preclass 1

- For a capacity of 4096
- How many memory accesses needed
  - When lookup fail?
  - When lookup succeed (on average)?

Tree

- Build search tree
- Walk down tree
- For a capacity of 4096, assume balanced…
- How many tree nodes visited
  - When lookup fail?
  - When lookup succeed (on average)?

Tree Insert

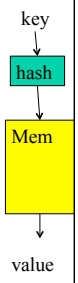
- Need to maintain balance
- Doable with O(log(N)) insert
  - Tricky
  - See Red-Black Tree

High Performance Map

- Would prefer not to search
- Want to do better than log(N) time
- Direct lookup of arrays, memory is good…

Hash Table

- Attempt to turn into direct lookup
- Compute some function of key
  - A hash
- Perform lookup at that point
- If hash maps a single entry
  - Great, got direct lookup

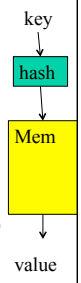


Preclass 3a

- Average number of entries per hash when N > HASH_CAPACITY?

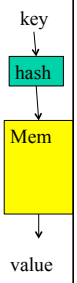
Hash Table

- Attempt to turn into direct lookup
- Compute some function of key
  - A hash
- Perform lookup at that point
- Typically, prepared for several keys to map to same hash → call it a bucket
  - Keep list or tree of things in each bucket



Hash Table

- Compute some function of key
  - A hash
- Perform lookup at that point
- Find bucket with small number of entries
  - Searching that bucket easier
  - …but challenge if bucket can get big



Preclass 3b

- Probability of conflict if N<<HASH_CAPACITY ?
- How can we reduce bucket sizes?

Preclass 4

N=1024

m→	0	1	2	3	4+
C=1024	0.37				
C=2048					
C=4096					

$$\binom{N}{m}\left(\frac{1}{C}\right)^{m}\left(1-\frac{1}{C}\right)^{N-m}$$

Hash

- Can tune hash parameters to control distribution
- Spend more memory → smaller buckets → less work finding things in buckets
  - Memory-Time tradeoff
- Still have possibility of large buckets

Perfect Hash

- **Perfect hash:** every key maps to a unique hash value
- Can create perfect hash if know keys in advance

Graph and Edge Model

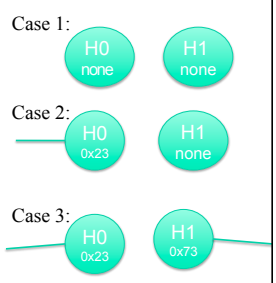
- Use 2 hashes
- Each hash value is node
- Each entry is edge
  - 2 hashes, connects 2 nodes
- Program output by filling in value for nodes
- add node values modulo capacity to get edge value

```
int G[capacity];
int value[capacity];
h0=hash0(key);
h1=hash1(key);
addr=(G[h0]+G[h1])
   %capacity;
res=value[addr];
```


Graph and Edge Model

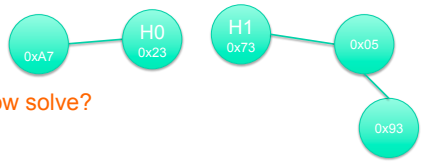
- Use 2 hashes
- Each hash value is node
- Each entry is edge
  - 2 hashes, connects 2 nodes
- Program output by filling in value for nodes
- add node values modulo capacity to get edge value

Consider insert 0xB0 in each case



Case 3 often Solvable

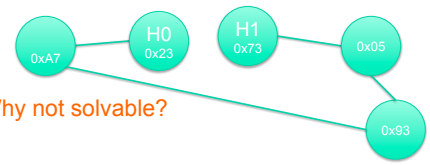
Consider insert 0xB0 in each case



- How solve?

Case 3 not always solvable

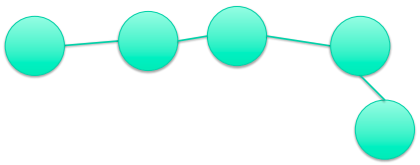
Consider insert 0xB0 in each case



- Why not solvable?

Perfect Hash Theory

- If graph is acyclic
  - Can always assign values to nodes to satisfy/produce any edge values
  - Walk graph in order from starting point



Perfect Hash Theory

- If graph is acyclic
  - Can always assign values to nodes to satisfy/produce any edge values
- Random graphs with a sufficiently low edge to node ratio are acyclic with high-probability
  - Good hash functions (random) makes graph random

Perfect Hash

- If do get cycle
  - Change hash
  - Change modulus
    - (add a bit more capacity)

Variants for Hardware

- Modulus can be expensive
  - Unless it is a power of 2
- Set capacity to next power of 2
  - …and modulus becomes simple
  - Can use xor instead of addition
    - No delay for carry, O(1) combining

Perfect Hash

- Can find tools to compute the perfect hash (e.g. gnuperf)

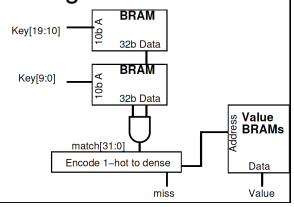
Hardware Map

FPGA

- Has BRAMs – normal memories, not associative
- 36Kb BRAM
  - 1024x36
- Can be 10b key → 36b value
  - Just using the memory sparsely

Assoc. Mem from BRAM

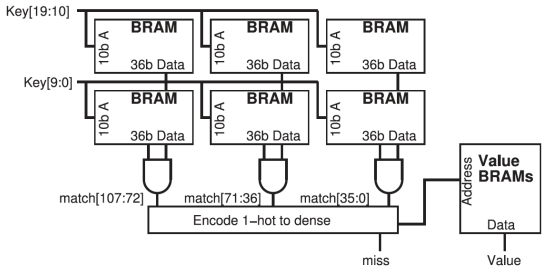
- For wider match
- Cover 10b of key with each BRAM
- Use 36 output bits to indicate if one of 36 entries match
- AND together corresponding entries
- Get 36 match bits
- Re-encode match bits to lookup value



BRAM Associative Memory

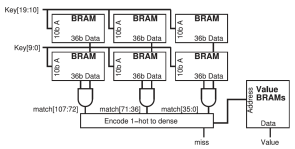
- How would we expand capacity?

BRAM Associative Memory



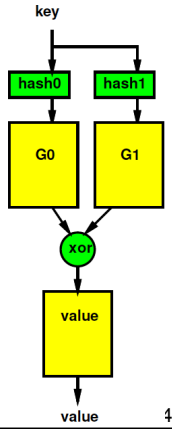
Associative Memory Cost

- Match unit
  - Requires 1 BRAM per 10b of key per 36 entries
  - (keylen/10b) * (capacity/36)



Perfect Hash

- Can do perfect hash very compactly
- Hash to log(capacity)
  - Two different hashes
- Use separate BRAM for each hash
- Xor together values
- Use as dense address to lookup values



Multi Hash

- Perfect hash hints that multiple hashes may be more useful
- E.g.
  - Setup several hashes and memories
  - Compute each hash
  - See if any are free and put there
    - (in software, put in smallest bucket found reduce maximum bucket size)

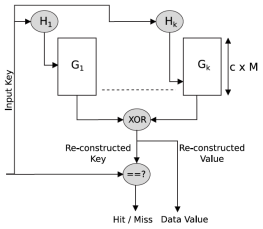
Multihash

- K hashes
- Probability of conflict in all K tables

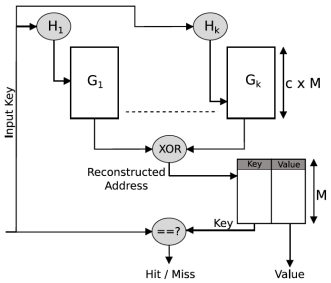
$$\left(\frac{N}{C}\right)^{K}$$

dMHC

- Dynamic Multi-Hash Cache

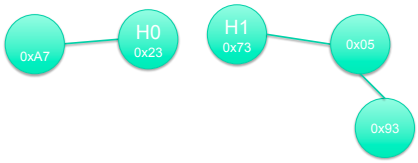


Two-Level dMHC



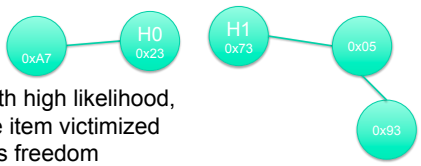
Perfect Hash: Case 3 often Solvable

Consider insert 0xB0 in each case



Conflict: victimize and reinsert

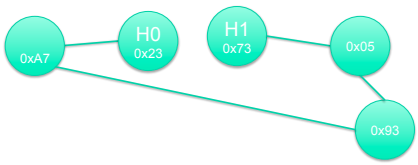
Consider insert 0xB0 in each case



- With high likelihood, the item victimized has freedom

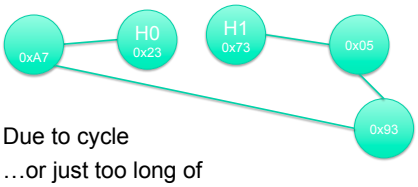
Perfect Hash: Case 3 not always solvable

Consider insert 0xB0 in each case



May not be able to solve

Consider insert 0xB0 in each case



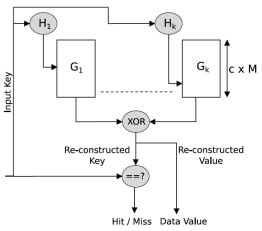
- Due to cycle
- …or just too long of a chain to repair in limited time

On Conflict Reinsert

- Nearly associative – occasionally need to drop an entry
- Makes less than perfect
  - Approximates an associative memory
- Potentially acceptable in some scenarios

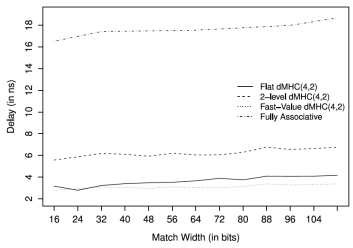
dMHC Lookup Fast

- Hash (few levels of xor)
- BRAM read
  - 2-3ns
- Xor (one level)
- Can run 1/cycle @200MHz +



dMHC Lookups Fast

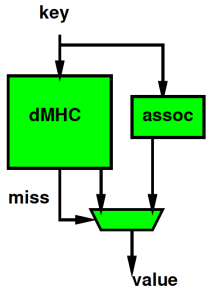
- Virtex 6 (one generation older than Zynq)



Hybrid

- If the number of conflicts (things we might drop) are small
  - Maybe just save those in a true associative memory.
- Use dMHC for bulk
- When get (unresolvable) conflict
  - Stick in small associative memory
    - Use BRAM associative memory
- Lookup in both in parallel

Hybrid dMHC+Assoc.



Design Space

- Area-Time tradeoffs
  - Minimal with sequential search
  - Hash capacity (more memory, less time)
  - Build fully associative – spend area, instead of time
- Area-Quality tradeoffs
  - dMHC tuning c, k
- Pre-computation/specialization opportunities when data known early
  - Perfect hash

Big Ideas

- Sparse, near O(1) Map access → Hash Table
- If know set of keys in advance, can build perfect hash
- More hashes, reduce conflicts
- Rich design space for engineering associative map solutions

Admin

- No class Wednesday
- Go to Horowitz Talk
  - *Life of Breaking and Making: What Happens When You Mix a Driving Curiosity and Episodic Overconfidence*
  - Wu & Chen at 3pm (classtime)
- Olukotun Thursday
  - Scaling Machine Learning performance…
  - Wu & Chen at 3pm
- First project milestone due Friday
  - Including teaming