Review: Reed-Solomon Codes

- Galois field codes: code words consist of symbols
  - Rather than bits
- Reed-Solomon codes:
  - Based on polynomials in GF(2^k) (i.e. k-bit symbols)
  - Data as coefficients, code space as values of polynomial:
    \[ P(x) = a_0 + a_1x + \ldots + a_{k-1}x^{k-1} \]
  - Coded: \( P(0), P(1), P(2), \ldots, P(n-1) \)
  - Can recover polynomial as long as get any \( k \) of \( n \)
- Properties: can choose number of check symbols
  - Reed-Solomon codes are “maximum distance separable” (MDS)
  - Can add \( d \) symbols for distance \( d+1 \) code
  - Often used in “erasure code” mode: as long as no more than \( n-k \)
    coded symbols erased, can recover data
- Side note: Multiplication by constant in GF(2^k) can be represented
  by \( k \times k \) matrix: \( a \times x \)
  - Decompose unknown vector into \( k \) bits: \( x = x_0 + 2x_1 + \ldots + 2^{k-1}x_{k-1} \)
  - Each column is result of multiplying a by \( 2^i \)

Aside: Why erasure coding?
High Durability/overhead ratio!

- Exploit law of large numbers for durability!
- 6 month repair, FBLPY:
  - Replication: 0.03
  - Fragmentation: 10^{-35}

\[
\begin{bmatrix}
1^0 & 1^1 & 1^2 & 1^3 & 1^4 \\
2^0 & 2^1 & 2^2 & 2^3 & 2^4 \\
3^0 & 3^1 & 3^2 & 3^3 & 3^4 \\
4^0 & 4^1 & 4^2 & 4^3 & 4^4 \\
5^0 & 5^1 & 5^2 & 5^3 & 5^4 \\
6^0 & 6^1 & 6^2 & 6^3 & 6^4 \\
7^0 & 7^1 & 7^2 & 7^3 & 7^4 \\
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
a_3 \\
a_4 \\
\end{bmatrix}
\]
Archival Dissemination of Fragments (online Error Correction Codes)

Archival Storage Discussion
• Continuous Repair of Redundancy: Data transferred from physical medium to physical medium
  – No “tapes decaying in basement”
  – Information becomes fully Virtualized
  – Keep the raw bits safe
• Thermodynamic Analogy: Use of Energy (supplied by servers) to Suppress Entropy
  – 1000 year time frame?
• Format Obsolescence
  – Continuous format evolution
  – Saving of virtual interpretation environment
• Proof that storage servers are “doing their job”
  – Can use zero-knowledge proof techniques
  – Reputations
• Data deletion/scrubbing?
  – Harder with this model, but possible in principle

Statistical Advantage of Fragments

• Latency and standard deviation reduced:
  – Memory-less latency model
  – Rate ½ code with 32 total fragments

Review: Cache performance
• Miss-oriented Approach to Memory Access:
  \[ \text{CPUtime} = IC \times \left( CPI_{\text{Execution}} + \frac{\text{MemAccess}}{\text{Inst}} \times \text{MissRate} \times \text{MissPenalty} \right) \times \text{CycleTime} \]
• Separating out Memory component entirely
  \[ \text{AMAT} = \frac{\text{HitTime}}{\text{Hit}} + \text{MissRate} \times \text{MissPenalty} \]
  \[ \text{AMAT} = \frac{\text{Frac}_{\text{hit}} \times ((\text{HitTime}_\text{hit} + \text{MissRate}_\text{hit} \times \text{MissPenalty}_\text{hit})) + \text{Frac}_{\text{data}} \times ((\text{HitTime}_\text{data} + \text{MissRate}_\text{data} \times \text{MissPenalty}_\text{data}))}{1} \]
• AMAT for Second-Level Cache
  \[ \text{AMAT}_2 = \text{HitTime}_2 + \text{MissRate}_2 \times \text{MissPenalty}_2 \]
  \[ \text{AMAT}_2 = \text{HitTime}_2 + \text{MissRate}_2 \times \text{AMAT}_2 \]
  \[ \text{AMAT}_2 = \text{HitTime}_2 + \text{MissRate}_2 \times ((\text{HitTime}_2 + \text{MissRate}_2 \times \text{MissPenalty}_2)) \]
### Example: Impact of Cache on Performance

- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Miss Behavior:
  - 10% of memory operations get 50 cycle miss penalty
  - 1% of instructions get same miss penalty
- CPI = ideal CPI + average stalls per instruction
  \[
  1.1 \text{ (cycles/ins)} + \left[ 0.30 \text{ (DataMops/ins)} \times 0.10 \text{ (miss/DataMop)} \times 50 \text{ (cycle/miss)} \right] + \left[ 1 \text{ (InstMop/ins)} \times 0.01 \text{ (miss/InstMop)} \times 50 \text{ (cycle/miss)} \right] = (1.1 + 1.5 + .5) \text{ cycle/ins} = 3.1
  \]
- 58% of the time the proc is stalled waiting for memory!
- AMAT = \((1/1.3) \times \left[ 1+0.01 \times 50 \right] + (0.3/1.3) \times \left[ 1+0.1 \times 50 \right] = 2.54

### What is impact of Harvard Architecture?

- Unified vs Separate I&D (Harvard)
  - Statistics (given in H&P):
    - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
    - 32KB unified: Aggregate miss rate=1.99%
- Which is better (ignore L2 cache)?
  - Assume 33% data ops ⇒ 75% accesses from instructions (1.0/1.33)
  - hit time=1, miss time=50
  - Note that data hit has 1 stall for unified cache (only one port)
  \[
  \begin{align*}
  \text{AMAT}_{\text{Harvard}} &= 75\% \times (1+0.64\% \times 50) + 25\% \times (1+6.47\% \times 50) = 2.05 \\
  \text{AMAT}_{\text{Unified}} &= 75\% \times (1+1.99\% \times 50) + 25\% \times (1+1+1.99\% \times 50) = 2.24
  \end{align*}
  \]

### Recall: Reducing Misses

- Classifying Misses: 3 Cs
  - **Compulsory**—The first access to a block is not in the cache, so the block must be brought into the cache. Also called cold start misses or first reference misses. (Misses in even an Infinite Cache)
  - **Capacity**—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. (Misses in Fully Associative Size X Cache)
  - **Conflict**—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called collision misses or interference misses. (Misses in N-way Associative, Size X Cache)
- More recent, 4th “C”:
  - **Coherence** - Misses caused by cache coherence.

### Review: 6 Basic Cache Optimizations

- Reducing hit time
  1. Avoiding Address Translation during Cache Indexing
     - E.g., Overlap TLB and cache access, Virtual Addressed Caches
  2. Giving Reads Priority over Writes
     - E.g., Read complete before earlier writes in write buffer
  3. Multilevel Caches
- Reducing Miss Rate
  4. Larger Block size (Compulsory misses)
  5. Larger Cache size (Capacity misses)
  6. Higher Associativity (Conflict misses)
12 Advanced Cache Optimizations

- Reducing hit time
  1. Small and simple caches
  2. Way prediction
  3. Trace caches

- Reducing Miss Penalty
  7. Critical word first
  8. Merging write buffers

- Increasing cache bandwidth
  4. Pipelined caches
  5. Multibanked caches
  6. Nonblocking caches

- Reducing Miss Rate
  9. Victim Cache
  10. Hardware prefetching
  11. Compiler prefetching
  12. Compiler Optimizations

3. Fast (Instruction Cache) Hit times via Trace Cache

Key Idea: Pack multiple non-contiguous basic blocks into one contiguous trace cache line

- Single fetch brings in multiple basic blocks
- Trace cache indexed by start address and next n branch predictions

3. Fast Hit times via Trace Cache (Pentium 4 only; and last time?)

- Find more instruction level parallelism?
  How avoid translation from x86 to microops?
- Trace cache in Pentium 4
  1. Dynamic traces of the executed instructions vs. static sequences of instructions as determined by layout in memory
     - Built-in branch predictor
  2. Cache the micro-ops vs. x86 instructions
     - Decode/translate from x86 to micro-ops on trace cache miss

+ 1. ⇒ better utilize long blocks (don’t exit in middle of block, don’t enter at label in middle of block)
- 1. ⇒ complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size
- 1. ⇒ instructions may appear multiple times in multiple dynamic traces due to different branch outcomes

9. Reducing Misses: a “Victim Cache”

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines
**Administrivia**

- Exam: Finally finished grading!
  - AVG: 64.4, Std: 21.4
  - Sorry for the delay!
  - Please look at my grading to make sure that I didn’t mess up
  - Solutions are up – please go through them
- We are talking about Chapter 5 (memory)
- Final dates:
  - Next Wednesday 4/27: Talk about GPUs and Manycore OS
    » Also, please come for HKN review
  - Wednesday 5/4: Quantum Computing (and DNA computing?)
  - Thursday 5/5: Oral Presentations: 10-11:30
  - Monday 5/9: Final papers due
    » 10-pages, double-column, conference format

---

**10. Reducing Misses by Hardware Prefetching of Instructions & Data**

- Prefetching relies on having extra memory bandwidth that can be used without penalty
- Instruction Prefetching
  - Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
  - Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer
- Data Prefetching
  - Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
  - Prefetching invoked if 2 successive L2 cache misses to a page, if distance between those cache blocks is < 256 bytes

---

**Issues in Prefetching**

- Usefulness – should produce hits
- Timeliness – not late and not too early
- Cache and bandwidth pollution

---

**Hardware Data Prefetching**

- Prefetch-on-miss:
  - Prefetch \( b + 1 \) upon miss on \( b \)

- One Block Lookahead (OBL) scheme
  - Initiate prefetch for block \( b + 1 \) when block \( b \) is accessed
  - *Why is this different from doubling block size?*
  - Can extend to \( N \) block lookahead

- Strided prefetch
  - If observe sequence of accesses to block \( b \), \( b+N \), \( b+2N \), then prefetch \( b+3N \) etc.

Example: IBM Power 5 [2003] supports eight independent streams of strided prefetch per processor, prefetching 12 lines ahead of current access
11. Reducing Misses by Software Prefetching Data

- **Data Prefetch**
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache
    (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution

- **Issuing Prefetch Instructions takes time**
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

12. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software

- **Instructions**
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts (using tools they developed)

- **Data**
  - **Merging Arrays**: improve spatial locality by single array of compound elements vs. 2 arrays
  - **Loop Interchange**: change nesting of loops to access data in order stored in memory
  - **Loop Fusion**: Combine 2 independent loops that have same looping and some variables overlap
  - **Blocking**: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

**Blocking Example**

/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    {r = 0;
     for (k = 0; k < N; k = k+1)
       r = r + y[i][k]*z[k][j];
    x[i][j] = r;
    }

- **Two Inner Loops:**
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]

- **Capacity Misses a function of N & Cache Size:**
  - $2N^3 + N^2 \Rightarrow$ (assuming no conflict; otherwise …)
- **Idea:** compute on BxB submatrix that fits

/* After */
for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
    for (i = 0; i < N; i = i+1)
      for (j = jj; j < min(jj+B-1,N); j = j+1)
        {r = 0;
         for (k = kk; k < min(kk+B-1,N); k = k+1)
           r = r + y[i][k]*z[k][j];
         x[i][j] = x[i][j] + r;
        }

- **Blocking Factor**
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- Conflict Misses Too?
Reducing Conflict Misses by Blocking

- Conflict misses in caches not FA vs. Blocking size
  - Lam et al. [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

Impact of Hierarchy on Algorithms

- Today CPU time is a function of (ops, cache misses)
- What does this mean to Compilers, Data structures, Algorithms?
  - Quicksort: fastest comparison based sorting algorithm when keys fit in memory
  - Radix sort: also called “linear time” sort
    For keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys
    - For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000

Quicksort vs. Radix: Instructions

- Quick vs. Radix Inst & Time
  - Quick (Instr/key)
  - Radix (Instr/key)

Quicksort vs. Radix Inst & Time

- Quick vs. Radix Inst & Time
  - Quick (Instr/key)
  - Radix (Instr/key)
**Quicksort vs. Radix: Cache misses**

![Graph showing cache misses for Quicksort and Radix]

**Experimental Study (Membench)**

- Microbenchmark for memory system performance

```
for array A of length L from 4KB to 8MB by 2x
for stride s from 4 Bytes (1 word) to L/2 by 2x
time the following loop (repeat many times and average)
  for i from 0 to L by s
    load A[i] from memory (4 Bytes)
```

**Membench: What to Expect**

- Consider the average cost per load
  - Plot one line for each array length, time vs. stride
  - Small stride is best: if cache line holds 4 words, at most ¼ miss
  - If array is smaller than a given cache, all those accesses will hit (after the first run, which is negligible for large enough runs)
  - Picture assumes only one level of cache
  - Values have gotten more difficult to measure on modern procs

**Memory Hierarchy on a Sun Ultra-2i**

- L1: 16 KB, 2 cycles (6 ns)
- L2: 2 MB, 12 cycles (36 ns)
- Mem: 396 ns (132 cycles)

See [www.cs.berkeley.edu/~yelick/arvindk/t3d-isca95.ps](http://www.cs.berkeley.edu/~yelick/arvindk/t3d-isca95.ps) for details
Memory Hierarchy on a Power3

Power3, 375 MHz
L2: 8 MB 128 B line 9 cycles
L1: 32 KB 128B line .5-2 cycles
Mem: 396 ns (132 cycles)

Compiler Optimization vs. Memory Hierarchy Search

• Compiler tries to figure out memory hierarchy optimizations
• New approach: “Auto-tuners” 1st run variations of program on computer to find best combinations of optimizations (blocking, padding, ...) and algorithms, then produce C code to be compiled for that computer
• “Auto-tuner” targeted to numerical method
  – E.g., PHIPAC (BLAS), Atlas (BLAS), Sparsity (Sparse linear algebra), Spiral (DSP), FFT-W

Sparse Matrix – Search for Blocking
for finite element problem [Im, Yelick, Vuduc, 2005]
900 MHz Itanium 2, Intel C v8: ref=275 Mflop/s
Best: 4x2

Conclusion

• Memory wall inspires optimizations since much performance lost
  – Reducing hit time: Small and simple caches, Way prediction, Trace caches
  – Increasing cache bandwidth: Pipelined caches, Multibanked caches, Nonblocking caches
  – Reducing Miss Penalty: Critical word first, Merging write buffers
  – Reducing Miss Rate: Compiler optimizations
  – Reducing miss penalty or miss rate via parallelism: Hardware prefetching, Compiler prefetching
• Performance of programs can be complicated functions of architecture
  – To write fast programs, need to consider architecture
    » True on sequential or parallel processor
  – We would like simple models to help us design efficient algorithms
• Will “Auto-tuners” replace compilation to optimize performance?