CS252
Graduate Computer Architecture
Lecture 22

Caching Optimizations
April 19th, 2010

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

Brief discussion of Transactional Memory

- LogTM: Log-based Transactional Memory
  - Kevin Moore, Jayaram Bobba, Michelle Moravan, Mark Hill & David Wood
  - Use of Cache Coherence protocol to detect transaction conflicts
- Transactional Interface:
  - `begin_transaction()`: Request that subsequent statements for a transaction
  - `commit_transaction()`: Ends successful transaction begun by matching `begin_transaction()`. Discards any transaction state saved for potential abort
  - `abort_transaction()`: Transfers control to a previously register conflict handler which should undo and discard work since last `begin_transaction()`

```
for(i=0; i<10000; ++i){
   begin_transaction();
     new_total = total.count + 1;
     private_data[id].count++;
     total.count = new_total;
   commit_transaction();
   think();
}
```

Figure 4. Shared-counter microbenchmark (main loop)

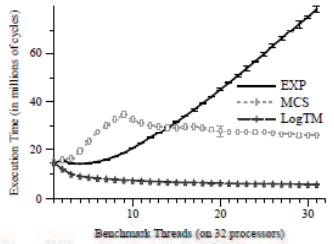
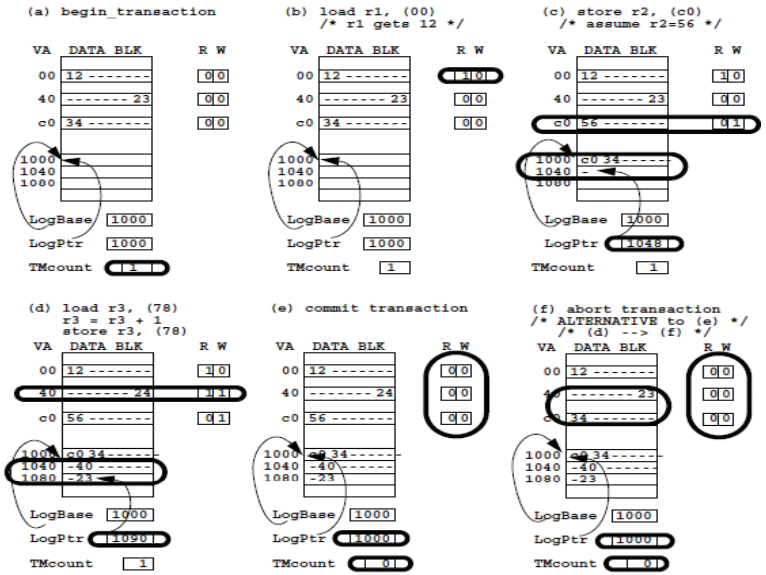


Figure 5. Execution time for LogTM transactions, test-and-test-and-set locks with exponential backoff (EXP) and MCS locks (MCS).

Specific Logging Mechanism



Review: Cache performance

- Miss-oriented Approach to Memory Access:

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

- Separating out Memory component entirely
  - AMAT = Average Memory Access Time

$$CPUtime = IC \times \left(CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$\begin{aligned} AMAT &= HitTime + MissRate \times MissPenalty \\ &= Frac_{Inst} \times (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) + \\ &\quad Frac_{Data} \times (HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data}) \end{aligned}$$

- AMAT for Second-Level Cache

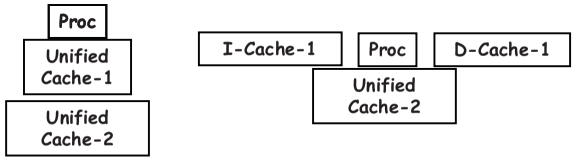
$$\begin{aligned} AMAT_{1st} &= HitTime_{1st} + MissRate_{1st} \times MissPenalty_{1st} \\ &= HitTime_{1st} + MissRate_{1st} \times AMAT_{2nd} \\ &= HitTime_{1st} + MissRate_{1st} \times (HitTime_{2st} + MissRate_{2st} \times MissPenalty_{2st}) \end{aligned}$$

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(cycles/ins) +
  [0.30 (DataMops/ins)
  x 0.10 (miss/DataMop) x 50 (cycle/miss)] +
  [1 (InstMop/ins)
  x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1
- 58% of the time the proc is stalled waiting for memory!
- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=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)

$AMAT_{Harvard}$=75%x(1+0.64%x50)+25%x(1+6.47%x50) = 2.05
$AMAT_{Unified}$=75%x(1+1.99%x50)+25%x(1+1+1.99%x50)= 2.24

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

- Reducing Miss Penalty

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

- **Increasing cache bandwidth**

4. Pipelined caches
5. Multibanked caches
6. Nonblocking caches

- **Reducing Miss Penalty**

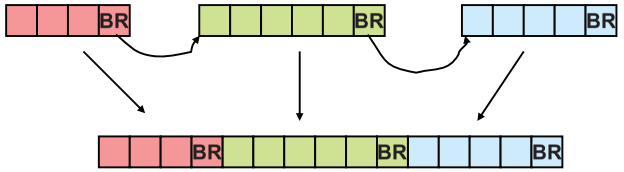
7. Critical word first
8. Merging write buffers

- **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

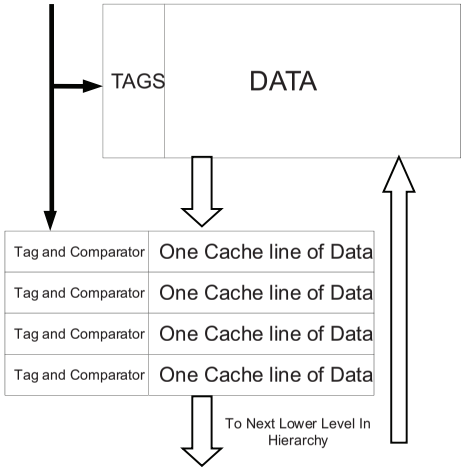
\+ 1. ⇒ better utilize long blocks (don't exit in middle of block, don't enter at label in middle of block)

\- 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

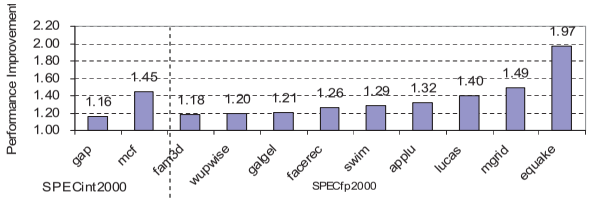
9. Reducing Misses: a "Victim Cache"

- **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



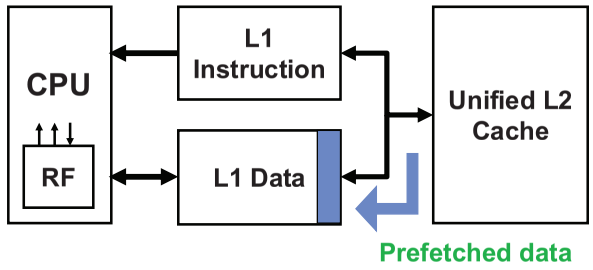
10. Reducing Misses by Hardware Prefetching of Instructions & Data

- 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

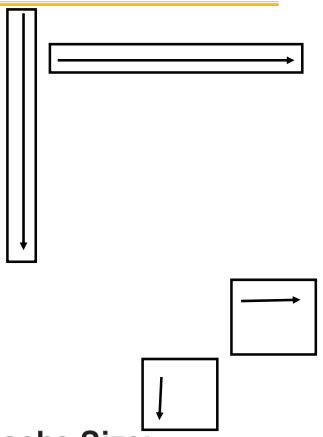
- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings 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;
     };
```



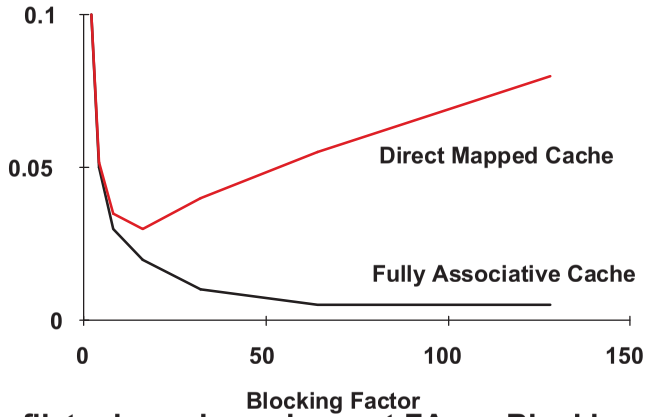
- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits

Blocking Example

```
/* 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;
     };
```

- B called *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

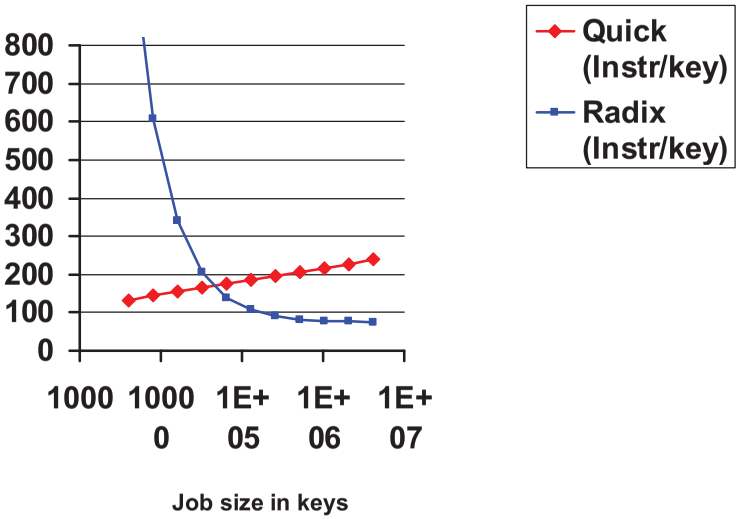
Administrivia

- Exam: Next Exam: Tentatively Monday 5/3
  - Could do it during day on Tuesday
  - Material: Everything up to last lecture
  - Closed Book, but 2 page hand-written notes (both sides)
- We have been talking about Chapter 5 (memory)
  - You should take a look, since might show up in test

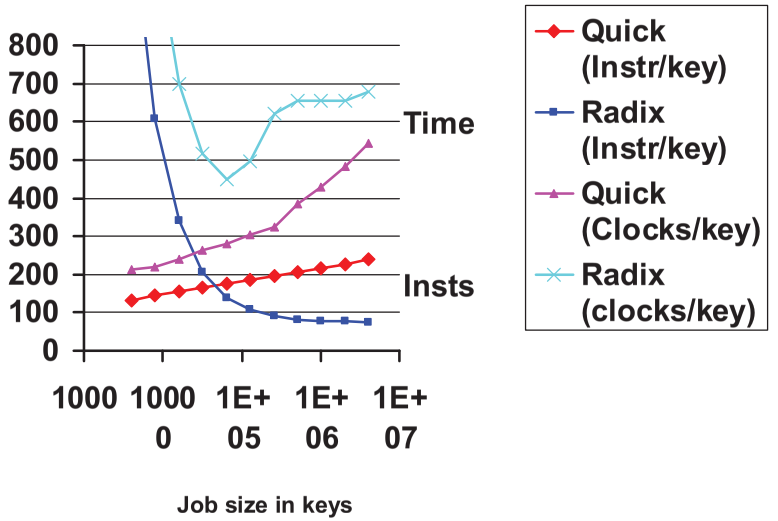
Impact of Hierarchy on Algorithms

- 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
- "The Influence of Caches on the Performance of Sorting" by A. LaMarca and R.E. Ladner. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, January, 1997, 370-379.
  - For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000

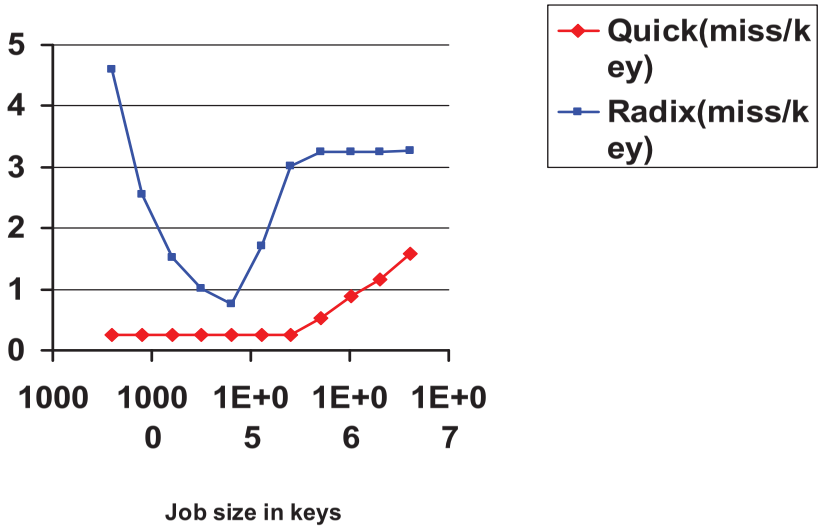
Quicksort vs. Radix: Instructions



Quicksort vs. Radix Inst & Time

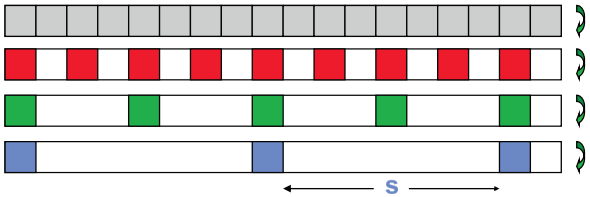


Quicksort vs. Radix: Cache misses



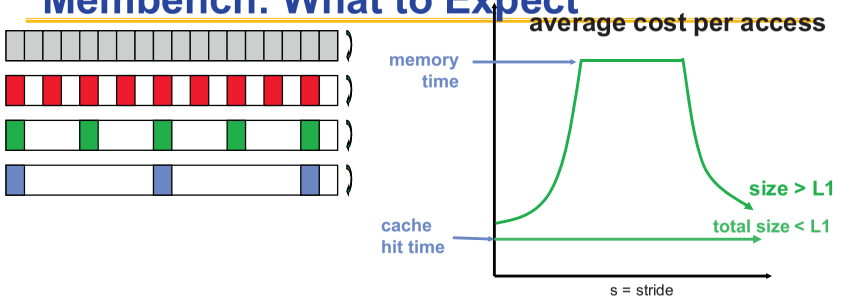
Experimental Study (Membench)

- Microbenchmark for memory system performance



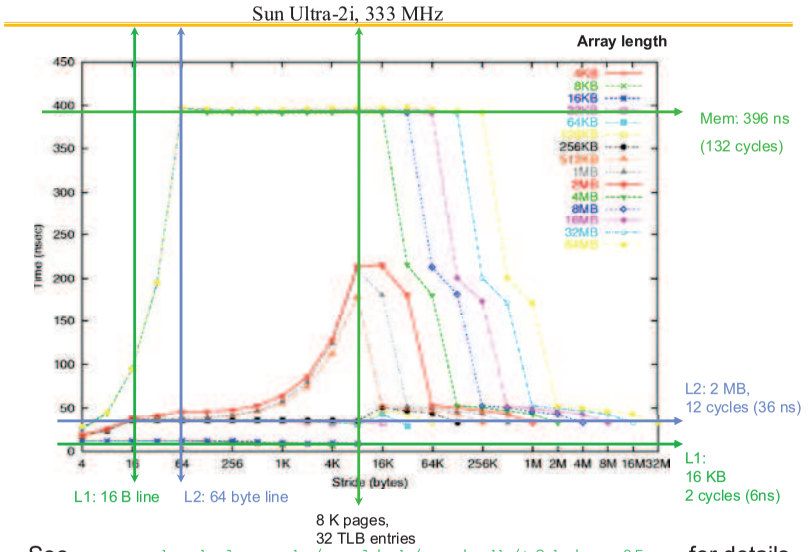
- for array A of length L from 4KB to 8MB by 2x
  for stride s from 4 Bytes (1 word) to L/2 by 2x 1 experiment
  time the following loop
  (repeat many times and average)
  for i from 0 to L by s
  load A[i] from memory (4 Bytes)

Membench: What to Expect



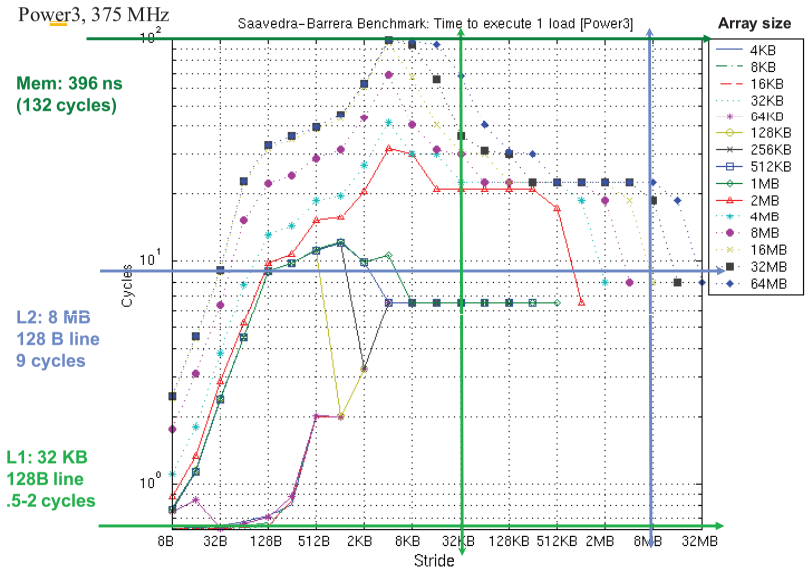
- Consider the average cost per load
  - Plot one line for each array length, time vs. stride
  - Small stride is best: if cache line holds 4 words, at most ¼ miss
  - If array is smaller than a given cache, all those accesses will hit (after the first run, which is negligible for large enough runs)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

Memory Hierarchy on a Sun Ultra-2i



See www.cs.berkeley.edu/~yelick/arvindk/t3d-isca95.ps for details

Memory Hierarchy on a Power3

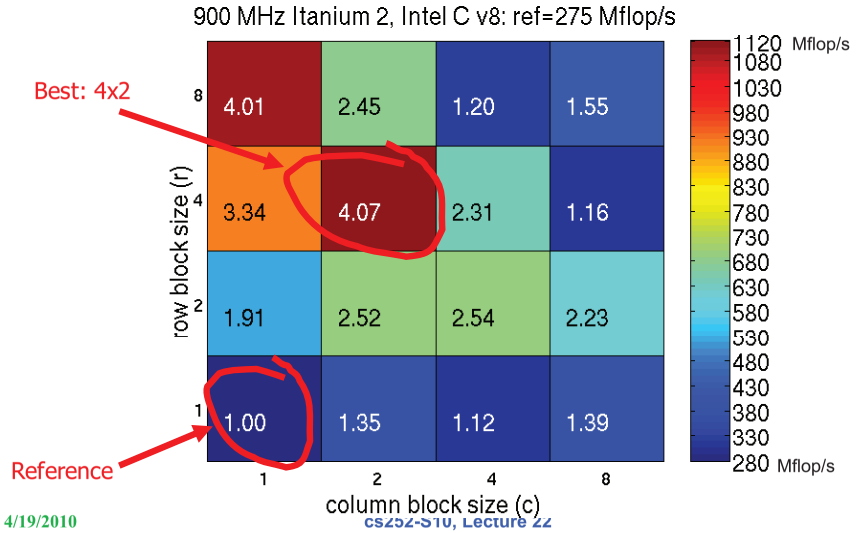


Compiler Optimization vs. Memory Hierarchy Search

- 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]



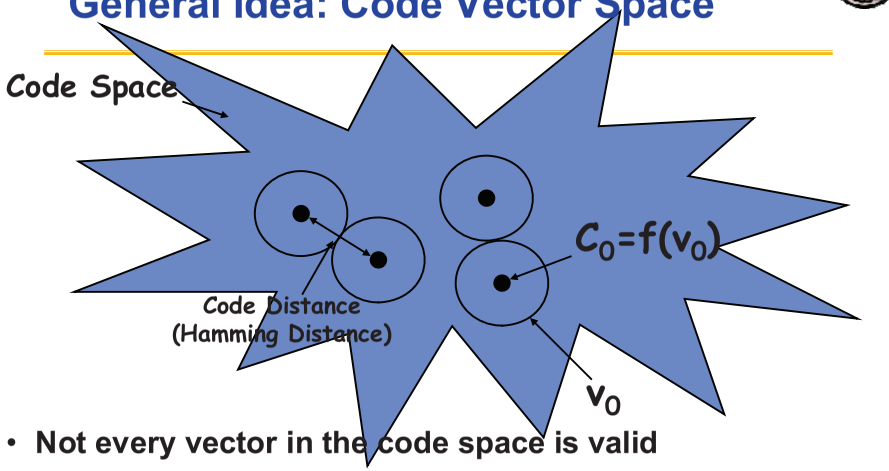
Setup for Error Correction Codes (ECC)

- Memory systems generate errors (accidentally flipped-bits)
  - DRAMs store very little charge per bit
  - "Soft" errors occur occasionally when cells are struck by alpha particles or other environmental upsets.
  - Less frequently, "hard" errors can occur when chips permanently fail.
  - Problem gets worse as memories get denser and larger
- Where is "perfect" memory required?
  - servers, spacecraft/military computers, ebay, …
- Memories are protected against failures with ECCs
- Extra bits are added to each data-word
  - used to detect and/or correct faults in the memory system
  - in general, each possible data word value is mapped to a unique "code word". A fault changes a valid code word to an invalid one - which can be detected.

ECC Approach

- Approach: Redundancy
  - Add extra information so that we can recover from errors
  - Simple technique: duplicate
- Block Codes: Data Coded in blocks
  - k data bits coded into n encoded bits
  - Measure of overhead: Rate of Code: K/N
  - Often called an (n,k) code
  - Consider data as vectors in GF(2) [i.e. vectors of bits]
- Code Space is set of all 2^n vectors, Data space set of 2^k vectors
  - Encoding function: *C=f(d)*
  - Decoding function: *d=f(C')*
  - Not all possible code vectors, *C, are valid!*

General Idea: Code Vector Space



- Not every vector in the code space is valid
- Hamming Distance (d):
  - Minimum number of bit flips to turn one code word into another
- Number of errors that we can detect: (d-1)
- Number of errors that we can fix: ½(d-1)

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?