

Lecture Topics

- subtle hazards (ProfileMe)
- space vs. time: sequential version
 - AI for ECE190 marbles game
 - BFS vs. DFS

Administrivia

- Note #2 handout: conclusion of a commercial bug saga

- source file annotation
 - Oprofile provides useful annotation tool
 - rather than mapping to functions
 - map to source lines
 - or to assembly code
 - N.B.: assembly source is harder as it lacks debug info
 - create dynamically based on profiled dataset
 - select a specific function or functions
 - limit by threshold (% of total samples)
- example (data courtesy of John Kelm)
 - 1000 traversals of 100,000-element linked list (+ a bunch of NOPs)
 - sample every 500 data cache accesses
 - annotated assembly (partial shown) [keep assembly up for next slide]

```

for (i = 0; i < 1000; i++) {
    iter = &head;
    while (NULL != iter->next) {
        asm ( "nop" ); // 32 of these...
        ...
        iter = iter->next;
    }
}

```

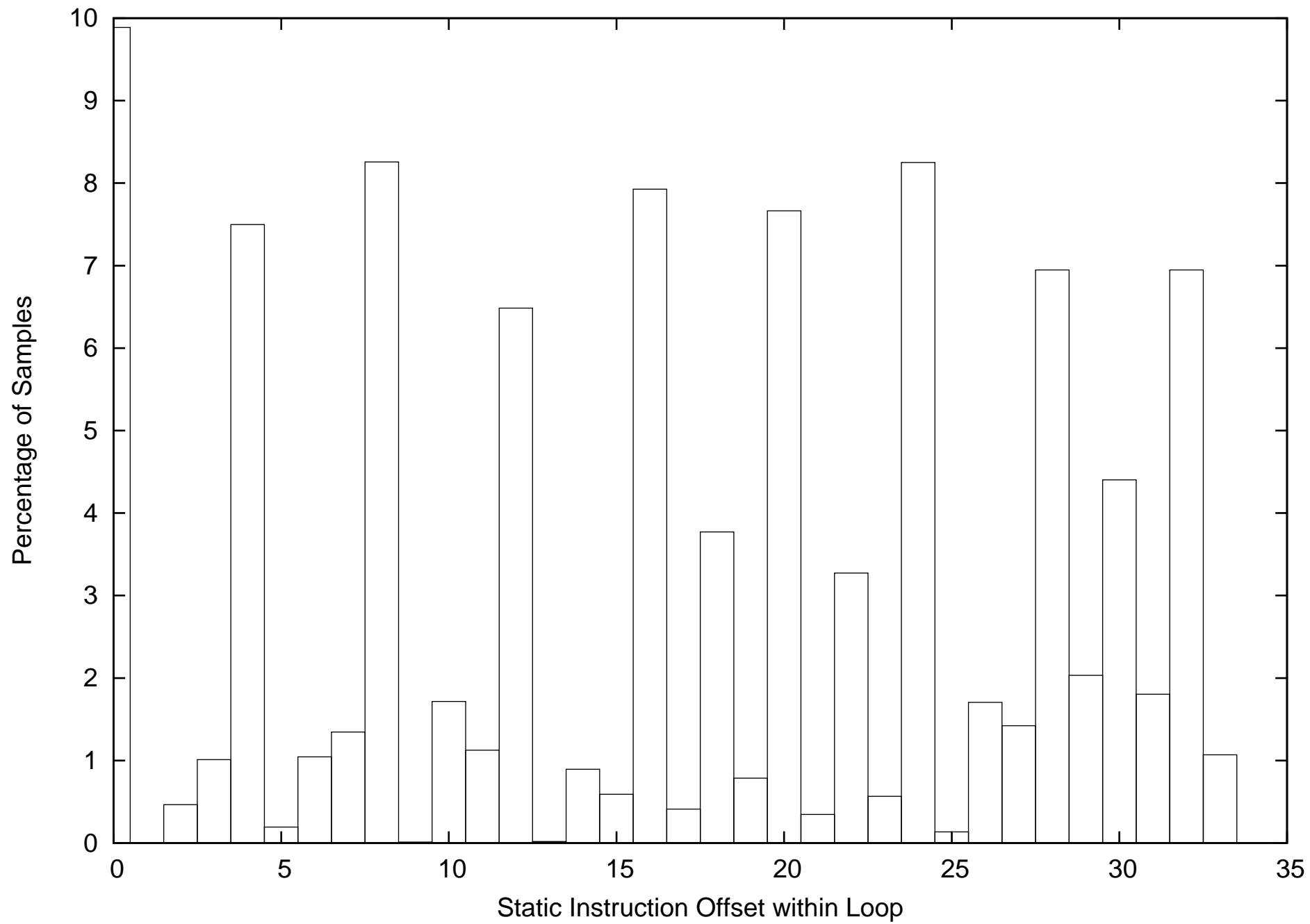
```

          : 40057d:          nop
407  0.3043 : 40057e:          nop    ; noise?
641  0.4792 : 40057f:          nop    ; more noise?
131703 98.4636 : 400580:          mov    (%rax),%rax
      10  0.0075 : 400583:          test   %rax,%rax
          : 400586:          jne    400560 <main+0x50>
          : 400588:          add    $1,%edx
      2  0.0015 : 40058b:          cmp    $1000,%edx
          : 400591:          jne    outer_loop

```

Subtle Hazards

- two main problems
 - sampling rate may alias with program's natural event frequency
 - events and instructions are not necessarily correlated (interrupts do not break most instructions)
 - floating point division
 - long-latency events (e.g., TLB misses)
 - still true today for some event types
- graph handed out
 - Intel CPU clock sampling (100,000 cycles)
 - note periodic behavior; superscalar issue/retirement?
- other data points based on AMD processor
 - retired micro ops
 - 99.6% on the JNE
 - NOP = 0 micro ops, and TEST is ... bypassed?
 - retired branches
 - 97% on the TEST
 - usually waiting on MOV cache miss, so test collects samples?
 - try again for micro op hypothesis?
 - instruction fetch
 - 87.5% on TEST
 - smaller peaks on unevenly-spaced NOPs
 - dispatch stalls
 - 70% spread over TEST, JNE, and first NOP
 - remainder distributed across other NOPs
 - usually waiting for cache miss; sometimes limited by integer pipeline? not if NOP = 0 micro ops...



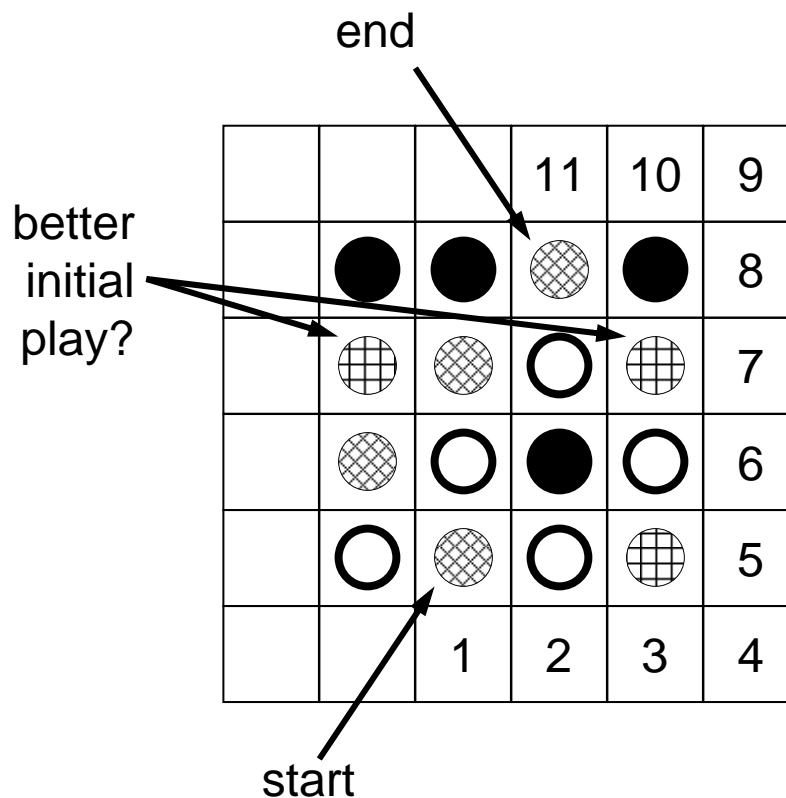
- ProfileMe
 - Dean et al., Proceedings of MICRO-30, pp. 292-302, 1997.
 - hardware support for collecting data on individual instructions
 - marked on fetch, tracked in detail through pipeline
 - randomize sampling period
 - track PC, cycles per pipeline stage, cache misses, effective addresses, branch target, retire/abort (e.g., mispredicted path)
 - also support sample pairs
 - use much smaller inter-fetch delay
 - obtain information on correlations in execution
 - estimate co-occupation of pipeline
- sample pair use example: wasted issue slots
 - given instruction X
 - estimate number of instructions that usefully issue with X
 - issue between X's issue and ready to retire
 - not squashed later due to branch misprediction or exception
 - estimate issue slots available
 - average latency of X
 - multiply by issue slots per cycle
 - subtract to find number of wasted issue slots when X executes
- cheap path profiles
 - use global branch history
 - sequence of taken/not-taken bits
 - easy and inexpensive to obtain
 - control flow convergence may lead to ambiguity
 - augment with paired instructions
 - 1-50 fetched instructions apart
 - was predecessor on a path being considered (if not, discard path)
 - around 75% accurate in their measurements

- Digital Continuous Profiling Infrastructure
 - Anderson et al., ACM Trans. Computer Sys., 15(4):1-14, 1997.
 - software package designed around hardware extensions
 - still an HP product as of 2004 (DEC to Compaq to HP)

Space vs. Time: Sequential Version

- AI for a game developed as an ECE190 project
- the playing board
 - $N \times N$ square of marbles
 - ring of open spaces around the marbles (a larger square)
 - each marble randomly chosen to be one of four colors
- scoring
 - based on shortest path between two marbles
 - move vertically/horizontally (not diagonally)
 - do not move through other marbles
- play
 - alternate turns until no play is possible
 - the turn
 - pick two marbles of same color
 - score shortest path between them
 - remove them from the board
- Clearly the first player has a major advantage...right?
- solution: play the same board with each player taking a turn at being first

- a sample 4×4 board
 - best opening play
 - diagonally opposite non-corner pair for 12 points
 - in this case, exposes internal pair worth 14 points!
 - better choice?
 - remove horizontally opposite pair for 11 points
 - if opponent removes 12-point pair, 14-point pair is reduced, too



- What's the best strategy?
 - analyze the whole game play?
 - theoretically possible, but lots of alternative scenarios...
 - one approach: min-max tree evaluation to fixed depth

- mix-max evaluation
 - assumptions
 - zero-sum game (two players, opposing goals)
 - “optimal” play by all players
 - approach
 - build tree of all possible moves for several future turns
 - use some heuristic to assign values to leaf states
 - work upwards from bottom of tree
 - opponent picks move that gives minimum value
 - player picks move that gives maximum value

- complexity? one simple approach...
 - number of possible moves could be N^4
 - (much smaller in most practical cases)
 - score changes? use all-pairs shortest paths
 - Floyd-Warshall is simple and fast: N^6
 - could do slightly better: $N^4 \lg N$
 - so...
 - N^6 for ONE move (current player only)
 - N^{10} for TWO moves (both players once)
 - N^{18} for FOUR moves (both players twice)
 - $N^{(8T+2)}$ for T moves by both players
 - nice to have $T=3$ for good play
 - and $N=8$ for a fun game
 - 2^{78} ...Good luck!

- again, number of moves is usually not so bad
- although it doesn't hurt to trim them down a bit
- we'll focus on the N^6 part

- Why Floyd-Warshall?
 - really simple algorithm
 - works by “relaxation” of distances
 - for all nodes M (middle)
 - for all nodes S (start)
 - for all nodes D (destination)
 - » if S can reach D more quickly by passing through M, reduce the distance and update the optimal direction
 - starting with an existing set of distances and directions
 - update for removing marble is ONE relaxation step
 - N^4 instead of $N^4 \lg N$
- easy as I walk DOWN my tree of moves
 - but what about UP?
 - relaxation is hard to undo...
- some options
 - “delta” data structure
 - record only those pairs that relax
 - use pointers
 - give up random access and pay time and code complexity
 - just copy it!
 - make a new copy and relax that one
 - throw it away to move UP the tree
 - use extra space, but avoid recomputation and complexity!
- code’s online
 - move reduction: ignore any move that is worth $< 3/4$ of the best (you can almost never recover such a bad move)
 - not meant to be read (sorry)
 - has some other AI gunk I was toying with (e.g., anti-greedy strategy)

- What's the big picture here?
 - incremental relaxation
 - applicable to output of ANY all-pairs-shortest-paths algo.
 - happens to be part of Floyd-Warshall
 - why write two algorithms?
 - need to invert/reproduce results
 - copying might be a good option, as with marbles game
 - “delta” data structures
 - pointer-based tree of changes
 - look up answer by checking each node back to original
 - slow look up, cheap copies
 - could be better for big data sets
- implicit assumed DFS rather than BFS...why? [brainstorm]
 - BFS [breadth-first search] good when...
seeking one answer at minimal depth or depth expected to be small
 - DFS [depth-first search] good when...
ALL OTHER INSTANCES!
- comparison...
 - elements ready for exploration
 - proportional to tree height
 - BFS is proportional to tree SIZE
 - data copying
 - one copy of dataset with modification/unmodification process
 - or a copy per element ready for exploration/being explored
 - good cache behavior around leaves
- DFS = stack/add to front; BFS = queue/add to back (careful with STL!)