Motivation

- Two major factors affecting the amount of ILP exposed
  - Branch instructions
  - Memory access instructions (loads & stores)
- Branches
  - Insert bubbles into the pipeline while waiting for the branch to be resolved
  - Have greater impact on performance as pipelines get deeper
  - Very important for Speculative Execution
- Memory Accesses
  - Grind the processor to a halt on a miss
  - Are satisfied by a hierarchy of caches
  - Have to tolerate multiple outstanding misses as machines get wider

Outline

- Static Branch Prediction
  - Motivation
  - State of the Art
  - Methodology
  - Results
- Data Prefetching
  - Motivation
  - State of the Art
  - Methodology
  - Results
- Related Work
- Conclusion

Static Branch Prediction - Motivation

- ILP compilers perform aggressive optimizations including control speculation
  - Modify the instruction schedule based on predictability to maximize ILP
- Profiling
  - Vital to the success of these optimizations
  - Not always feasible
- Can substitute Static Branch Prediction for profiling
  - Apply heuristics on source level information

Static Branch Prediction - State of the Art

- Several heuristics based on branch attributes such as:
  - Direction
  - Opcode and operand types
  - Operand values
  - Code contained in taken and fall-through paths
- Contributions
  - Additional heuristics
  - Use of static branch prediction in the compiler
Static Branch Prediction - Methodology

- **Call Heuristic**
  - “If the successor contains a subroutine call and does not postdominate, predict that the successor is not taken.”
  - Function types that are predicted strongly not taken:
    - I/O Buffering
    - Exiting
    - Error Processing
    - Memory Allocation
    - Printing
    - Functions calling the ones mentioned above

- **Loop Header Heuristic**
  - “When a branch is derived from a conditional loop expression, predict that the successor is taken when it is a loop header or a loop preheader that does not postdominate the branch.”
  - Different cases
    - variable init and loop-bound
    - constant init and variable loop-bound
    - constant init and constant init

- **Pointer Heuristic**
  - “If a branch compares a pointer to NULL or compares two pointers, predict that pointers are not equal as long as these pointers are not part of an array.”

- **Return Heuristic**
  - “Predict that the successor containing a return is not taken.”

- **Character Comparison Heuristic**
  - “If a branch is an integer comparison against a character constant, predict that the comparison fails.”

- **Restricted opcode AND Heuristic**
  - “If a comparison is made against one bit in a bit field, predict that the bit is not set.”

- **Opcode Heuristic**
  - Predict the following comparisons fail:
    - $< 0$ or $<= 0$ for integers
    - $a == b$ for floating point values
  - Predict the following comparisons succeed:
    - $> 0$ or $>= 0$ for integers

Static Branch Prediction - Results

- Compare with perfect predictor
  - Always predict the most frequently taken branch direction

- Show
  - Percentage of branches that the heuristic is applicable to
  - Misprediction rate of the perfect predictor
  - Misprediction rate of the individual heuristic

Data Prefetching - Motivation

- Processor/Memory Performance Gap
  - Processors increase in speed by 1.5x every two years
  - Memory gets denser at each generation, but only marginally faster
  - In real applications, a great deal of the execution time is spent waiting for memory requests to be satisfied

- Prefetching
  - One way to hide the time spent waiting on caches
  - Makes use of compiler analysis or runtime information to make requests for data before it is needed
  - Overlaps useful computation with memory stalls. However, to get good overlap, data must be requested far before its’ use
Data Prefetching - Motivation

- **Pointer Chasing**
  - To prefetch d nodes ahead we have to dereference d pointers.

Data Prefetching - State of The Art

- **Software and Hardware techniques**
  - Both hardware and software techniques are well studied.
  - Software can make use of compiler level program analysis of arrays, but it pollutes the cache and incurs execution overhead.
  - Hardware is dynamic and adaptable, but has access to less information than a compiler.
  - This talk will focus on software prefetching targeting Recursive Data Structures (RDS).

Data Prefetching - Terminology

- n: Current RDS node.
- d: The minimum node distance that completely hides the access latency.
- Jump pointer: Pointer to node n+d.

Data Prefetching - Methodology (1 of 3)

- **Greedy Prefetching**
  - In a k-ary RDS, each node has k pointers to other nodes.
  - Immediate control can flow to only one of these.
  - Greedy prefetching uses the other k-1 pointers as natural jump pointers.
  - There is no need for additional storage space.
  - There is no control over prefetching distance i.e. d.

Data Prefetching - Methodology (2 of 3)

- **History Prefetching**
  - Artificially insert jump pointer at node n to node n+d.
  - Requires compiler synthesis to determine optimal d.
  - Uses a FIFO queue of length d.
  - Cost:
    - Execution overhead to create and maintain jump pointers.
    - Space overhead to store jump pointers.

Data Prefetching - Methodology (3 of 3)

- **Data-Linearization Prefetching**
  - Prefetches w/o any pointer dereferences.
  - Maps heap-allocated objects into contiguous memory locations.
  - The prefetch address at node n is &n + d x sizeof(n).
  - Preferably performed at creation to minimize overhead.
  - Improves spatial locality.
  - For a binary tree:
    - left child of node i: 2i
    - right child of node i: 2i+1

Data Prefetching - Experimental Framework

- **Implementation**
  - Automated: Greedy.
    - Type declaration information to keep track of RDSs.
    - Control flow information to recognize loops and recursive calls with recurrent pointer updates.
  - By hand: History and Data-linearization.

- Cycle accurate MIPS R10000 simulator.
- Olden benchmarks (pointer-intensive toy benchmarks).
- Additional MMU to satisfy increased memory pressure.
Data Prefetching - Greedy Results
- Half of the programs show degradation up to 2%
- Other half show improvements up to 45%

Data Prefetching - Other Results
- History Prefetching
  - Works only for health
- Data linearization
  - Not applicable to other programs

Data Prefetching - Filtering
- Misfetches occupy bus bandwidth and replace useful data in the data cache
- Use memory feedback to prevent loads that hit above a certain threshold from being prefetched

Related Work
- Branch Prediction
  - Multithreading (e.g., SMT, SSMT)
    - D. Tullsen, UCSD
  - Predicated Execution
    - S. Mahlke, Hewlett Packard
  - Hardware Branch Predictors
    - S. McFarling, Compaq Western Research Laboratory
- Data Prefetching
  - Multithreading
    - Hardware prefetching more effective
      - Context Predictors (e.g., Markov predictor)
        - D. Grunwald, University of Colorado
      - Streaming Buffers
        - N. Jouppi, Compaq Western Research Laboratory
  - Notable software prefetching studies
    - Jump pointer chaining
      - A. Roth, G. Sohi, University of Wisconsin

Conclusions
- Branch and memory stalls are the main performance bottlenecks in microprocessors
- Two effective techniques aiming to hide these latencies:
  - Branch Prediction
  - Prefetching