Lecture 9: Branch Prediction, Dependence Speculation, and Data Prediction

Prof. John Kubiatowicz
Computer Science 252
Fall 1998

Review: Vector Processing

- Vector Processing represents an alternative to complicated superscalar processors.
- Primitive operations on large vectors of data
- Load/store architecture:
  - Data loaded into vector registers; computation is register to register.
  - Memory system can take advantage of predictable access patterns:
    - » Unit stride, Non-unit stride, indexed
- Vector processors exploit large amounts of parallelism without data and control hazards:
  - Every element is handled independently and possibly in parallel
  - Same effect as scalar loop without the control hazards or complexity of tomasulo-style hardware
- Hardware parallelism can be varied across a wide range by changing number of *vector lanes* in each vector functional unit.

Review: Vector Processing

- If code is vectorizable, then simple hardware, more energy efficient than Out-of-order machines.
- Can decrease power by lowering frequency so that voltage can be lowered, then duplicating hardware to make up for slower clock:

$$\text{Power} \propto CV^2 f$$

$$\left\{\begin{array}{c} f = \frac{1}{n} f_0 \\ Lanes = n \times Lanes_0 \\ V = \delta V_0;\ \delta < 1 \end{array}\right\} \Rightarrow \left\{\begin{array}{c} \text{Performance Constant} \\ \text{Power Change}: \delta^2 < 1 \end{array}\right\}$$

- Note that V_o can be made as small as permissible within process constraints by simply increasing "n"

Prediction: Branches, Dependencies, Data New era in computing?

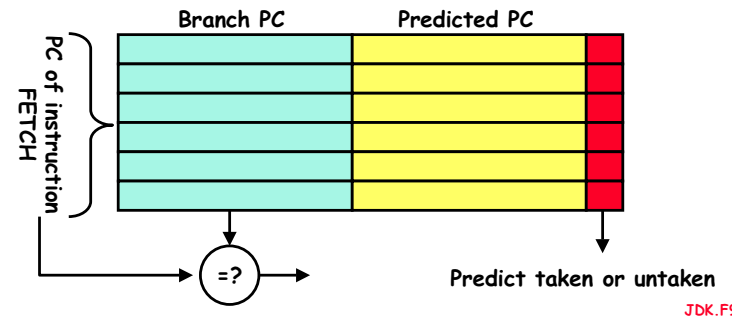
- Prediction has become essential to getting good performance from scalar instruction streams.
- We will discuss predicting branches, data dependencies, actual data, and results of groups of instructions:
  - At what point does computation become a probabilistic operation + verification?
  - We are pretty close with control hazards already…
- Why does prediction work?
  - Underlying algorithm has regularities.
  - Data that is being operated on has regularities.
  - Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems.
- Prediction $\Rightarrow$ Compressible information streams?

Dynamic Branch Prediction

- Is dynamic branch prediction better than static branch prediction?
  - Seems to be. Still some debate to this effect
  - Josh Fisher had good paper on "Predicting Conditional Branch Directions from Previous Runs of a Program." ASPLOS '92. In general, good results if allowed to run program for lots of data sets.
    - » How would this information be stored for later use?
    - » Still some difference between best possible static prediction (using a run to predict itself) and weighted average over many different data sets
  - Paper by Young et all, "A Comparative Analysis of Schemes for Correlated Branch Prediction" notices that there are a small number of important branches in programs which have dynamic behavior.

Need Address at Same Time as Prediction

- Branch Target Buffer (BTB): Address of branch index to get prediction AND branch address (if taken)
  - Note: must check for branch match now, since can't use wrong branch address (Figure 4.22, p. 273)



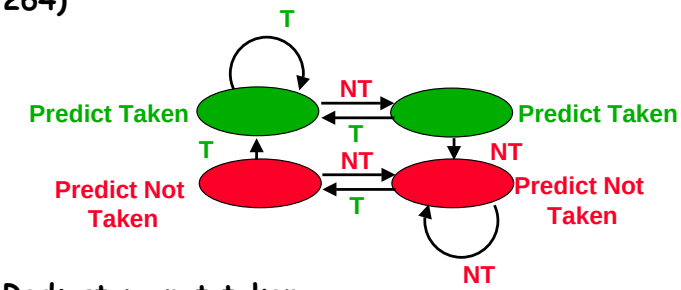
- Return instruction addresses predicted with stack

Dynamic Branch Prediction

- Performance = *f*(accuracy, cost of misprediction)
- Branch History Table: Lower bits of PC address index table of 1-bit values
  - Says whether or not branch taken last time
  - No address check
- Problem: in a loop, 1-bit BHT will cause two mispredictions (avg is 9 iteratios before exit):
  - End of loop case, when it exits instead of looping as before
  - First time through loop on *next* time through code, when it predicts exit instead of looping

Dynamic Branch Prediction
(Jim Smith, 1981)

- Solution: 2-bit scheme where change prediction only if get misprediction *twice:* (Figure 4.13, p. 264)



- Red: stop, not taken
- Green: go, taken
- Adds *hysteresis* to decision making process

BHT Accuracy

- Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch when index the table
- 4096 entry table programs vary from 1% misprediction (nasa7, tomcatv) to 18% (eqntott), with spice at 9% and gcc at 12%
- 4096 about as good as infinite table (in Alpha 211164)

Correlating Branches

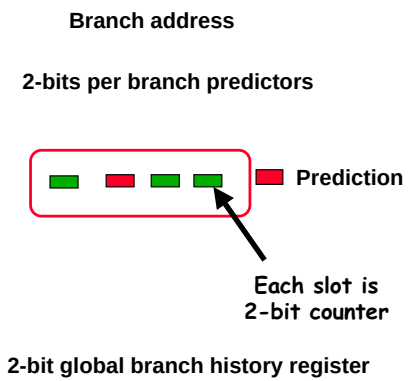
- Hypothesis: recent branches are correlated; that is, behavior of recently executed branches affects prediction of current branch
- Two possibilities; Current branch depends on:
  - Last m most recently executed branches anywhere in program Produces a "GA" (for "global address") in the Yeh and Patt classification (e.g. GAg)
  - Last m most recent outcomes of same branch. Produces a "PA" (for "per address") in same classification (e.g. PAg)
- Idea: record m most recently executed branches as taken or not taken, and use that pattern to select the proper branch history table entry
  - A single history table shared by all branches (appends a "g" at end), indexed by history value.
  - Address is used along with history to select table entry (appends a "p" at end of classification)
  - If only portion of address used, often appends an "s" to indicate "set-indexed" tables (I.e. GAs)

Correlating Branches

- For instance, consider global history, set-indexed BHT. That gives us a GAs history table.

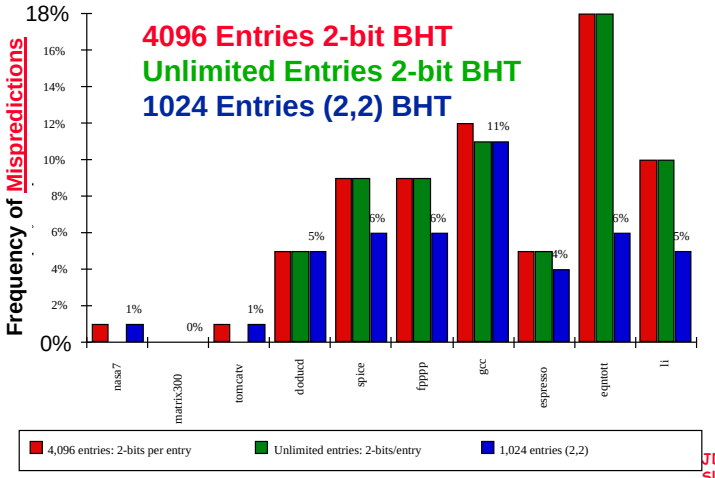
(2,2) GAs predictor

- First 2 means that we keep two bits of history
- Second means that we have 2 bit counters in each slot.
- Then behavior of recent branches selects between, say, four predictions of next branch, updating just that prediction
- Note that the original two-bit counter solution would be a (0,2) GAs predictor
- Note also that aliasing is possible here...



Accuracy of Different Schemes

(Figure 4.21, p. 272)



Re-evaluating Correlation

- Several of the SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:

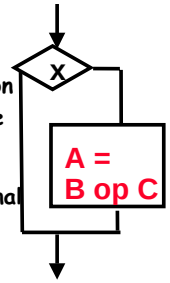
program	branch %	static	# = 90%
compress	14%	236	13
eqntott	25%	494	5
gcc	15%	9531	2020
mpeg	10%	5598	532
real gcc	13%	17361	3214

- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?

Predicated Execution

- Avoid branch prediction by turning branches into conditionally executed instructions:

  if (x) then A = B op C else NOP
  - If false, then neither store result nor cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr.
  - IA-64: 64 1-bit condition fields selected so conditional execution of any instruction
  - This transformation is called "if-conversion"



- Drawbacks to conditional instructions
  - Still takes a clock even if "annulled"
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline

Dynamic Branch Prediction Summary

- Prediction becoming important part of scalar execution.
  - Prediction is exploiting "information compressibility" in execution
- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch.
  - Either different branches (GA)
  - Or different executions of same branches (PA).
- Branch Target Buffer: include branch address & prediction
- Predicated Execution can reduce number of branches, number of mispredicted branches

CS252 Administrivia

- Select Project by next Friday (we will look at some options later in the lecture)
  - Need to have a partner for this. News group/email list?
  - Web site (as we shall see) has a number of suggestions
  - I am certainly open to other suggestions
    - » make one project fit two classes?
    - » Something close to your research?
- Reading for next Wednesday:
  - Avinash Sodani and Guri Sohi, "Dynamic Instruction Reuse"
  - Yiannakis Sazeides and James Smith, "Modeling Program Predictability"
  - Dirk Grundwald and Artur Klauser, "Confidence Estimation for Speculation Control"
- One paragraph for first two papers, one paragraph for last.

Discussion of Young/Smith paper

Discussion of Store Sets

Design problem: improve answer

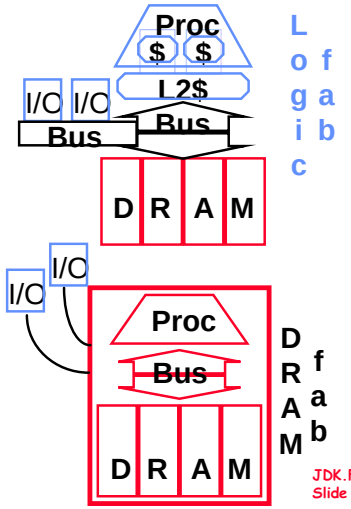
CS252 Projects

- IRAM project related
- BRASS project related
- DynaCOMP related (or Introspective Computing)
- Industry suggested/MISC

IRAM Vision Statement

Microprocessor & DRAM on a single chip:

- on-chip memory latency 5-10X, bandwidth 50-100X
- improve energy efficiency 2X-4X (no off-chip bus)
- serial I/O 5-10X v. buses
- smaller board area/volume
- adjustable memory size/width



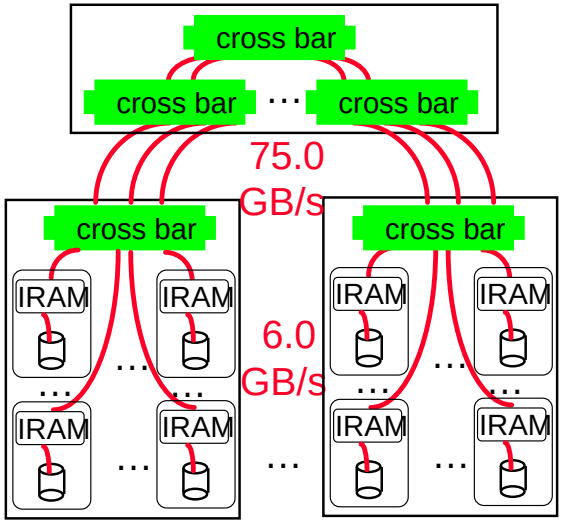
App #1: Intelligent PDA (2003?)

- Pilot PDA (todo,calendar, calculator, addresses,...)
- + Gameboy (Tetris, ...)
- + Nikon Coolpix (camera)
- + Cell Phone, Pager, GPS, tape recorder, TV remote, am/fm radio, garage door opener, ...
- + Wireless data (WWW)
- + Speech, vision recog.
- + Speech output for conversations

– Speech control of all devices

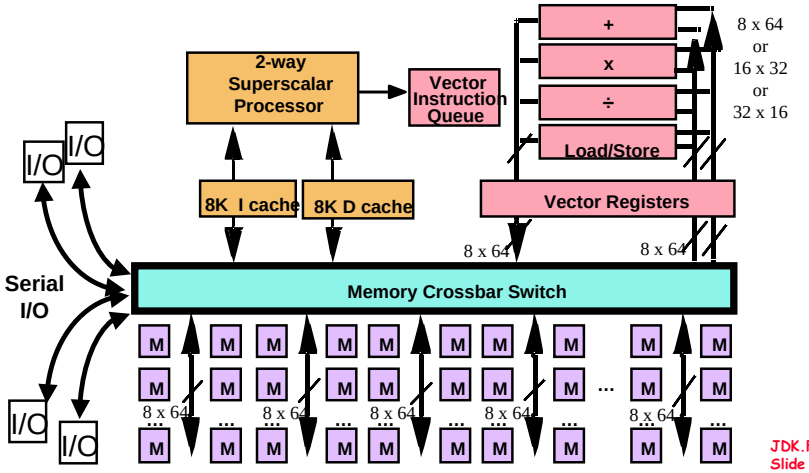
– Vision to see surroundings, scan documents, read bar codes, measure room

App #2: "Intelligent Disk"(IDISK): Scaleable Decision Support?

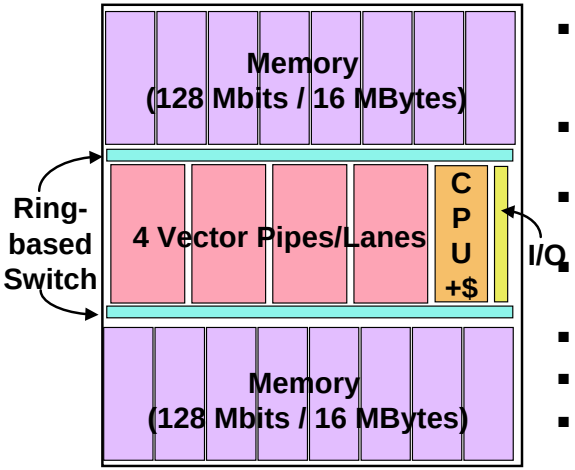


- 1 IRAM/disk + xbar + fast serial link v. conventional SMP
- Move function to data v. data to CPU (scan, sort, join,...)
- Network latency = f(SW overhead), not link distance
- Avoid I/O bus bottleneck of SMP
- Cheaper, faster, more scalable (-1/3 $, 3X perf)

V-IRAM-2: 0.13 µm, Fast Logic, 1GHz 16 GFLOPS(64b)/64 GOPS(16b)/128MB



Tentative VIRAM-1 Floorplan



- 0.18 µm DRAM 32 MB in 16 banks x 256b, 128 subbanks
- 0.25 µm, 5 Metal Logic
- - 200 MHz MIPS, 16K I$, 16K D$
- - 4 200 MHz FP/int. vector units
- die: - 16x16 mm
- xtors: - 270M
- power: -2 Watts

Brass Vision Statement

- The emergence of high capacity reconfigurable devices is igniting a revolution in general-purpose processing. It is now becoming possible to tailor and dedicate functional units and interconnect to take advantage of application dependent dataflow. Early research in this area of reconfigurable computing has shown encouraging results in a number of spot areas including cryptography, signal processing, and searching --- achieving 10-100x computational density and reduced latency over more conventional processor solutions.
- BRASS: Microprocessor & FPGA on single chip:
  - use some of millions of transitors to customize HW dynamically to application

DynaCOMP Vision Statement

- Modern microprocessors gather profile information in hardware in order to generate predictions for all sorts of things: Branch prediction, dependence speculation, and Value prediction.
- Also, processors such as the Pentium-II employ a primitive form of "compilation" to translate x86 operations into internal RISC-like micro-ops.
- So, why not do all of this in software? Make use of a combination of explicit monitoring, dynamic compilation technology, and genetic algorithms to:
  - Simplify hardware, possibly using large on-chip multiprocessors built from simple processors.
  - Improve performance through feedback-driven optimization. Continuous: *Execution, Monitoring, Analysis, Recompilation*
  - Generate design complexity automatically so that designers are not required to. Use of explicit proof verification techniques to verify that code generation is correct.
- This is aptly called *Introspective Computing*