Review #1: Hardware versus Software Speculation Mechanisms

- To speculate extensively, must be able to disambiguate memory references
  - Much easier in HW than in SW for code with pointers
- HW-based speculation works better when control flow is unpredictable, and when HW-based branch prediction is superior to SW-based branch prediction done at compile time
  - Mispredictions mean wasted speculation
- HW-based speculation maintains precise exception model even for speculated instructions
- HW-based speculation does not require compensation or bookkeeping code

Review #2: Hardware versus Software Speculation Mechanisms cont’d

- Compiler-based approaches may benefit from the ability to see further in the code sequence, resulting in better code scheduling
- HW-based speculation with dynamic scheduling does not require different code sequences to achieve good performance for different implementations of an architecture
  - may be the most important in the long run?

Review #3: Software Scheduling

- Instruction Level Parallelism (ILP) found either by compiler or hardware
- Loop level parallelism is easiest to see
  - SW dependencies/compiler sophistication determine if compiler can unroll loops
  - Memory dependencies hardest to determine => Memory disambiguation
    - Very sophisticated transformations available
- Trace Scheduling to Parallelize If statements
- Superscalar and VLIW: CPI < 1 (IPC > 1)
  - Dynamic issue vs. Static issue
  - More instructions issued at same time => larger hazard penalty
  - Limitation is often number of instructions that you can successfully fetch and decode per cycle

VLIW in Embedded Designs

- VLIW: greater parallelism under programmer, compiler control vs. hardware in superscalar
- Used in DSPs, Multimedia processors as well as IA-64
- What about code size?
- Effectiveness, Quality of compilers for these applications?

Example VLIW for multimedia: Philips Trimedia CPU

- Every instruction contains 5 operations
- Predicated with single register value; if 0 => all 5 operations are canceled
- 128 64-bit registers, which contain either integer or floating point data
- Partitioned ALU (SIMD) instructions to compute on multiple instances of narrow data
- Offers both saturating arithmetic (DSPs) and 2’s complement arithmetic (desktop)
- Delayed Branch with 3 branch slots
### Trimedia Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Examples</th>
<th>No. Ops</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load/Store</td>
<td>ld8, ld16, ld32, ld64, limm</td>
<td>36</td>
<td>SIMD, signed, unsigned, register indirect, indirect scaled addressing</td>
</tr>
<tr>
<td>Byte shift</td>
<td>shift 1-16, zero, sign</td>
<td>67</td>
<td>SIMD sign convert</td>
</tr>
<tr>
<td>Bit shifts</td>
<td>shift 8-15, zero, sign</td>
<td>44</td>
<td>SIMD sign convert</td>
</tr>
<tr>
<td>Integer</td>
<td>add, or, and, xor</td>
<td>100</td>
<td>SIMD, signed, unsigned, register indirect, indirect scaled addressing</td>
</tr>
<tr>
<td>Arithmetic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating point</td>
<td></td>
<td>59</td>
<td>scalar and SIMD</td>
</tr>
<tr>
<td>Logical</td>
<td></td>
<td>33</td>
<td>SIMD, cache, spacial, logical</td>
</tr>
<tr>
<td>Branch</td>
<td></td>
<td>10</td>
<td>unconditional, trap</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>410</td>
<td></td>
</tr>
</tbody>
</table>

- large number of ops because used set of retargetable compilers, multiple machine descriptions, and die size estimators to explore the space to find the best cost-performance design
- Verification, manufacturing test, design time?

### Philips Trimedia CPU

- Compiler responsible for including no-ops
  - both within an instruction-- when an operation field cannot be used-- and between dependent instructions
  - processor does not detect hazards, which if present will lead to incorrect execution
- Code size? compresses the code (~ Quiz #1)
  - decompresses after fetched from instruction cache

### Example

- MIPS code for loop
  
  Loop:  
  ```
  L1:  
  R11, R0 (R4)  # R11 = a[i]  
  L1:  
  R12, R0 (R5)  # R12 = b[i]  
  DADDU R17, R11, R12  # R17 = a[i] + b[i]  
  SD  
  R17, 0 (R6)  # c[i] = a[i] + b[i]  
  DADDIU R5, R5, 8  # R5 = a[i] + 8  
  DADDIU R6, R6, 8  # R6 = c[i] + 8  
  BNE R4, R7, Loop  # if not last go to Loop  
  ```

- Then unravel 4 times and schedule

- Using MIPS notation, look at code for
  
  ```
  void sum (int a[], int b[], int c[], int n)  
  {  
  int i;  
  for (i=0; i<n; i++)  
  c[i] = a[i] + b[i];  
  ```

### Trimedia Functional Units, Latency, Instruction Slots

- 23 functional units of 11 types, which of 5 slots can issue (and hence number of functional units)

<table>
<thead>
<tr>
<th>F.U.</th>
<th>Latency</th>
<th>Operation Slot</th>
<th>Typical operations performed by functional unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>0 X X X</td>
<td>1 2 3 4 5</td>
<td>Integer add/subtract/compare, logarithmic</td>
</tr>
<tr>
<td>DMem</td>
<td>2 X</td>
<td></td>
<td>Loads and stores</td>
</tr>
<tr>
<td>DMemclp</td>
<td>2 X</td>
<td></td>
<td>Cache invalid, prefetch, alloca</td>
</tr>
<tr>
<td>Status</td>
<td>0 X X</td>
<td></td>
<td>Stacks and registers</td>
</tr>
<tr>
<td>DSPALU</td>
<td>X X</td>
<td></td>
<td>Simple DSP arithmetic ops</td>
</tr>
<tr>
<td>DSPALU</td>
<td>X X</td>
<td></td>
<td>DSP ops with multiplication</td>
</tr>
<tr>
<td>Branch</td>
<td>3 X X</td>
<td></td>
<td>Branches and jumps</td>
</tr>
<tr>
<td>FALU</td>
<td>2 X X</td>
<td></td>
<td>FP add, subtract</td>
</tr>
<tr>
<td>FMUL</td>
<td>2 X X</td>
<td></td>
<td>Integer and FP multiply</td>
</tr>
<tr>
<td>FComp</td>
<td>0 X</td>
<td></td>
<td>FP compare</td>
</tr>
<tr>
<td>FToogh</td>
<td>16 X</td>
<td></td>
<td>FP divide, square root</td>
</tr>
</tbody>
</table>

### Trimedia Version

- 24/40 slots used (60%) in this example
Clock cycles to execute 2D iDCT

- 3rd project meetings 4/11: good progress!
  - Meet with some on Friday
- 4/18 Wed Quiz #2 310 Soda at 5:30
- Pizza at La Val's at 8:30
- What’s left:
  - 4/20 Fri, “How to Have a Bad Academic Career”
    (Career/Talk Advice): signup for talks
  - 4/25 Wed, Oral Presentations (BAM to 2 PM) 611 Soda (no lecture)
  - 4/27 Fri (no lecture)
- 5/2 Wed Poster session (noon - 2): end of course

Transmeta Crusoe MPU

- 80x86 instruction set compatibility through a software system that translates from the x86 instruction set to VLIW instruction set implemented by Crusoe
- VLIW processor designed for the low-power marketplace

Crusoe processor: Basics

- VLIW with in-order execution
- 64 Integer registers
- 32 floating point registers
- Simple in-order, 6-stage integer pipeline:
  - 2 fetch stages, 1 decode, 1 register read, 1 execution, and 1 register write-back
- 10-stage pipeline for floating point, which has 4 extra execute stages
- Instructions in 2 sizes: 64 bits (2 ops) and 128 bits (4 ops)

Crusoe processor: Operations

- 5 different types of operation slots:
  - ALU operations: typical RISC ALU operations
  - Compute: this slot may specify any integer ALU operation (2 integer ALUs), a floating point operation, or a multimedia operation
  - Memory: a load or store operation
  - Branch: a branch instruction
  - Immediate: a 32-bit immediate used by another operation in this instruction
- For 128-bit instr: 1st 3 are Memory, Compute, ALU; last field either Branch or Immediate

80x86 Compatability

- Initially, and for lowest latency to start execution, the x86 code can be interpreted on an instruction by instruction basis
- If a code segment is executed several times, translated into an equivalent Crusoe code sequence, and the translation is cached
  - The unit of translation is at least a basic block, since we know that if any instruction is executed in the block, they will all be executed
  - Translating an entire block both improves the translated code quality and reduces the translation overhead, since the translator need only be called once per basic block
- Assumes 16MB of main memory for cache
Exception Behavior during Speculation

- Crusoe support for speculative reordering consists of 4 major parts:
  1. shadowed register file
     - Shadow discarded only when x86 instruction has no exception
  2. program-controlled store buffer
     - Only store when no exception; keep until OK to store
  3. memory alias detection hardware with speculative loads
  4. conditional move instruction (called select) that is used to do if-conversion on x86 code sequences

Crusoe Performance?

- Crusoe depends on realistic behavior to tune the code translation process, it will not perform in a predictive manner when benchmarked using simple, but unrealistic scripts
  - Needs idle time to translate
  - Profiling to find hot spots
- To remedy this factor, Transmeta has proposed a new set of benchmark scripts
  - Unfortunately, these scripts have not been released and endorsed by either a group of vendors or an independent entity

Real Time, so comparison is Energy

<table>
<thead>
<tr>
<th>Workload Description</th>
<th>Energy Consumption for the Workload (W/Hr.)</th>
<th>Relative Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mobile Pentium III 1.5V @ 500 MHz</td>
<td>TM 3200</td>
</tr>
<tr>
<td>MP3 playback</td>
<td>0.672</td>
<td>0.214 0.32</td>
</tr>
<tr>
<td>DVD playback</td>
<td>1.13</td>
<td>0.479 0.42</td>
</tr>
</tbody>
</table>

Crusoe Applications?

- Notebook: Sony, others
- Compact Servers: RLX technologies

VLIW Readings

- What are characteristics of VLIW?
- Is ELI-512 the first VLIW?
  - How many bits in instruction of ELI-512?
- What is breakthrough?
- What expected speedup over RISC?
- What is wrong with vector?
- What benchmark results on code size, speedup?
- What limited speedups to 5X to 10X?
- What other problems faced ELI-512?
- In retrospect, what wished changed?
- In retrospect, what naïve about?

Review of Course

- Review and Goodbye to Computer Architecture, topic by topic + follow-on courses
- Future Directions for Computer Architecture?
Chapter 1: Performance and Cost

- Amdahl's Law:
  \[
  \text{Speedup}_{\text{overall}} = \frac{1}{\text{ExTime}_{\text{old}} \times (1 - \text{Fraction}_{\text{enhanced}}) + \text{Fraction}_{\text{enhanced}} \times \text{Speedup}_{\text{enhanced}}}
  \]

- CPI Law:

<table>
<thead>
<tr>
<th>CPU time</th>
<th>Seconds</th>
<th>Instructions</th>
<th>Cycles</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Program</td>
<td>Instruction</td>
<td>Cycle</td>
<td></td>
</tr>
</tbody>
</table>

Goodbye to Performance and Cost

- Will sustain 2X every 1.5 years?
  - Can integrated circuits improve below 1.8 micron in speed as well as capacity?
  - 5-6 yrs to PhD => 16X CPU speed, 10X DRAM capacity, 25X Disk capacity?
    (10 GHz CPU, 1GB DRAM, 2TB disk?)

Chapter 5: Memory Hierarchy

- Processor-DRAM Performance gap
- Alpha 21264: 108 clock to memory
  \[648\] instruction issues during miss
- 1/3 to 2/3 die area for caches, TLB
- Alpha 21264: 108 clock to memory
  \(648\) instruction issues during miss
- 3 Cs: Compulsory, Capacity, Conflict
- 4 Questions: where, who, which, write
- Applied recursively to create multilevel caches
- Performance = \(f(\text{hit time}, \text{miss rate}, \text{miss penalty})\)
  - danger of concentrating on just one when evaluating performance

Cache Optimization Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>Miss rate</th>
<th>MB</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>HW Prefetching of Inst/Data</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ctrl</td>
<td>Priority to Read Misses</td>
<td>+</td>
<td>+</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Subblock Placement</td>
<td>+</td>
<td>+</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Early Restart &amp; Critical Word 1st</td>
<td>+</td>
<td>+</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Non-Blocking Caches</td>
<td>+</td>
<td>+</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Second Level Caches</td>
<td>+</td>
<td>+</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Small &amp; Simple Caches</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Avoiding Address Translation</td>
<td>+</td>
<td>+</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Pipelining Writes</td>
<td>+</td>
<td>+</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

memory hierarchy art: taste in selecting between alternatives to find combination that fits well together
Chapter 6: Storage I/O

• Disk BW 40%/yr, areal density 60%/yr, $/MB faster?
• Little’s Law: \[ \text{Length}_{\text{system}} = \text{rate} \times \text{Time}_{\text{system}} \]
  (Mean number customers = arrival rate x mean service time)
• Benchmarks: scaling, cost, auditing, response time limits
  RAID: performance and reliability
  Queuing theory? IEOR 161, 267, 268
  SW storage systems? CS 286
  “Implementation of Data Base Systems”

Goodbye to Storage I/O

• Disks attached directly to networks, avoiding the file server? (“Network Attached Storage Devices”)
• Disks:
  - Extraordinary advance in capacity/drive, $/GB
  - Currently 17 Gbit/sq. in. vs. can continue past 100 Gbit/sq. in?
  - Bandwidth, seek time not keeping up: 3.5 inch form factor makes sense? 2.5 inch form factor in near future? 1.0 inch form factor in long term?
• Tapes
  - No investment, must be backwards compatible
  - Are they already dead?
  - What is a tapeless backup system?

Summary: I/O Benchmarks

• Scaling to track technological change
• TPC: price performance as normalizing configuration feature
• Auditing to ensure no foul play
• Throughput with restricted response time is normal measure
• Benchmarks to measure Availability, Maintainability?

Chapter 7: Networks

Sender

<table>
<thead>
<tr>
<th>Sender Overhead</th>
<th>Transmission time (size + bandwidth)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(processor busy)</td>
<td></td>
</tr>
</tbody>
</table>

Receiver

<table>
<thead>
<tr>
<th>Time of Flight</th>
<th>Transmission time (size + bandwidth)</th>
<th>Receiver Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>(processor busy)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total Latency = Sender Overhead + Time of Flight + Message Size + BW + Receiver Overhead

High BW networks + high overheads violate of Amdahl’s Law

Chapter 7: Networks

• Similarities of SANs, LANs, WANs
• Integrated circuit revolutionizing networks as well as processors
• Switch is a specialized computer
• Protocols allow heterogeneous networking, handle normal and abnormal events
• Interested in learning more on networks?
  EE 122 “Introduction to Computer Networks” (Stoika)
  CS 268 “Computer Networks” (Stoika)
Review: Networking

- Clusters +: fault isolation and repair, scaling, cost
- Clusters -: maintenance, network interface performance, memory efficiency
- Google as cluster example:
  - fault isolation (2 failures per day yet available)
  - repair (replace failures weekly/repair offline)
- Maintenance: 8 people for 6000 PCs
- Cell phone as portable network device
  - # Handsets >> # PCs
  - Universal mobile interface?
- Is future services built on Google-like clusters delivered to gadgets like cell phone handset?

Google to Networks

- Will network interfaces follow example of graphics interfaces and become first class citizens in microprocessors, thereby avoiding the I/O bus?
- Will Ethernet standard keep winning the LAN wars? e.g., 1 Gbit/sec, 10 Gbit/sec, wireless (802.11B)...

Chapter 8: Multiprocessors

- Layers:
  - Programming Model
  - Communication Abstraction
  - Interconnection SW/OS
  - Interconnection HW

- Programming Model
  - Multiprogramming: lots of jobs, no communication
  - Shared address space: communicate via memory
  - Message passing: send and receive messages
  - Data Parallel: several agents operate on several data sets simultaneously and exchange information globally and simultaneously (shared or message passing)

- Communication Abstraction:
  - Shared address space: e.g., load, store, atomic swap
  - Message passing: e.g., send, receive library calls
  - Debate over this topic (ease of programming, scaling)

- Many hardware designs 1:1 programming model

- Interested in learning more on multiprocessors:
  - CS 258 "Parallel Computer Architecture"
  - E 267 "Programming Parallel Computers"

Goodbye to Multiprocessors

- Successful today for file servers, time sharing, databases, graphics; will parallel programming become standard for production programs?
  - If so, what enabled it: new programming languages, new data structures, new hardware, new courses, ...
- Which won large scale number crunching, databases: Clusters of independent computers connected via switched LAN vs. large shared NUMA machines?
  - Why?

Chapter 2: Instruction Set Architecture

- What ISA looks like to pipeline?
  - Cray: load/store machine; registers; simple instr. format

- RISC: Making an ISA that supports pipelined execution

- VLIW/EPIC: compiler controls Instruction Level Parallelism (ILP)

- Interested in learning more on compilers and ISA?
  - CS 264/5 "Advanced Programming Language Design and Optimization"

Goodbye to Instruction Set Architecture

- What did IA-64/EPIC do well besides floating point programs?
  - Was the only difference the 64-bit address v. 32-bit address?
  - What happened to the AMD 64-bit address 80x86 proposal?
  - What happened on EPIC code size vs. x86?
  - Did Intel Oregon increase x86 performance so as to make Intel Santa Clara EPIC performance similar?
Goodbye to Dynamic Execution
- Did Transmeta-like compiler-oriented translation survive vs. hardware translation into more efficient internal instruction set?
- Did ILP limits really restrict practical machines to 4-issue, 4-commit?
- Did we ever really get CPI below 1.0?
- Did value prediction become practical?
- Branch prediction: How accurate did it become?
  - For real programs, how much better than 2 bit table?
- Did Simultaneous Multithreading (SMT) exploit underutilized Dynamic Execution HW to get higher throughput at low extra cost?
  - For multiprogrammed workload (servers) or for parallelized single program?

Goodbye to Static, Embedded
- Did VLIW become popular in embedded?
  What happened on code size?
- Did vector become popular for media applications, or simply evolve SIMD?
- Did DSP and general purpose microprocessors remain separate cultures, or did ISAs and cultures merge?
  - Compiler oriented?
  - Benchmark oriented?
  - Library oriented?
  - Saturation + 2's complement

Goodbye to Computer Architecture
- Did emphasis switch from cost-performance to cost-performance-availability?
- What support for improving software reliability? Security?

- 1985-2000: 1000X performance
  - Moore's Law transistors/chip vs. Moore's Law for Performance/MPU
  - Hennessy: industry been following a roadmap of ideas known in 1985 to exploit Instruction Level Parallelism to get 1.55X/year
  - Caches, Pipelining, Superscalar, Branch Prediction, Out-of-order execution, ...
- ILP limits: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler, HW?
- Did Moore's Law in transistors stop predicting microprocessor performance? Did it drop to old rate of 1.3X per year?
  - Less because of processor-memory performance gap?