EECS 252 Graduate Computer Architecture

Lec 16 – Papers, MP Future Directions, and Midterm Review

David Patterson
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~pattrsn
http://vlsi.cs.berkeley.edu/cs252-s06

Outline

- ILP
- Compiler techniques to increase ILP
- Loop Unrolling
- Static Branch Prediction
- Dynamic Branch Prediction
- Overcoming Data Hazards with Dynamic Scheduling
- (Start) Tomasulo Algorithm
- Conclusion

Amdahl's Law Paper

- Gene Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings, (30), pp. 483-485, 1967.
- How long is paper?
- How much of it is Amdahl's Law?
- What other comments about parallelism besides Amdahl's Law?

Parallel Programmer Productivity

- Lorin Hochstein *et al* "Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers." International Conference for High Performance Computing, Networking and Storage (SC'05). Nov. 2005
- What did they study?
- What is argument that novice parallel programmers are a good target for High Performance Computing?
- How can account for variability in talent between programmers?
- What programmers studied?
- What programming styles investigated?
- How big multiprocessor?
- How measure quality?
- How measure cost?

Parallel Programmer Productivity

- Lorin Hochstein *et al* "Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers." International Conference for High Performance Computing, Networking and Storage (SC'05). Nov. 2005
- What hypotheses investigated?
- What were results?
- Assuming these results of programming productivity reflect the real world, what should architectures of the future do (or not do)?
- How would you redesign the experiment they did?
- What other metrics would be important to capture?
- Role of Human Subject Experiments in Future of Computer Systems Evaluation?

CS 252 Administrivia

- Monday March 20 Quiz 5-8 PM 405 Soda
- Monday March 20 lecture – Q&A, problem sets with Archana
- Wednesday March 22 no class: project meetings in 635 Soda
- Spring Break March 27 – March 31
- Chapter 5 Advanced Memory Hierarchy
- Chapter 6 Storage
- Interconnect Appendix

High Level Message

- Everything is changing
- Old conventional wisdom is out
- We DESPERATELY need a new architectural solution for microprocessors based on parallelism
  - My focus is "All purpose" computers vs. "single purpose" computers
    $\Rightarrow$ Each company gets to design one
- Need to create a "watering hole" to bring everyone together to quickly find that solution
  - architects, language designers, application experts, numerical analysts, algorithm designers, programmers, …

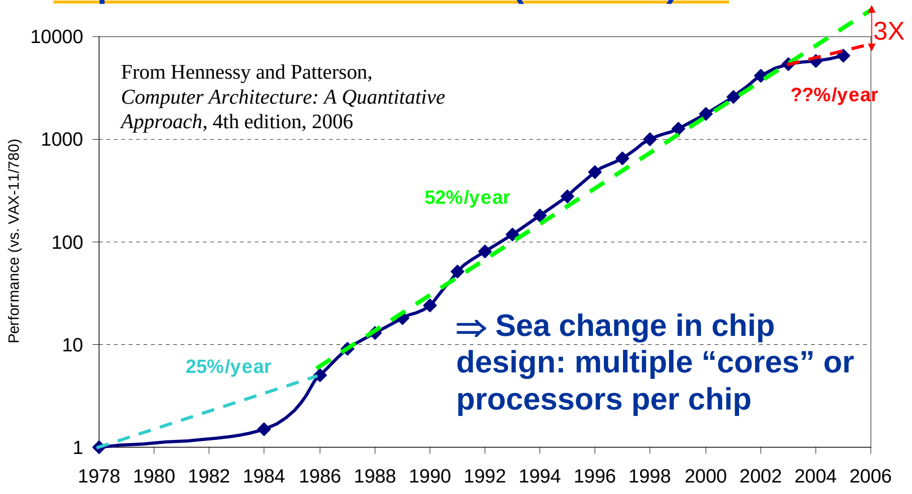
Outline

- Part I: A New Agenda for Computer Architecture
  - Old Conventional Wisdom vs. New Conventional Wisdom
  - New Metrics for Success
  - Innovating at HW/SW interface without compilers
  - New Classification for Architectures and Apps
- Part II: A "Watering Hole" for Parallel Systems
  - Research Accelerator for Multiple Processors
- Conclusion

Conventional Wisdom (CW) in Computer Architecture

- Old CW: Power is free, Transistors expensive
- New CW: "Power wall" Power expensive, Xtors free (Can put more on chip than can afford to turn on)
- Old: Multiplies are slow, Memory access is fast
- New: "Memory wall" Memory slow, multiplies fast (200 clocks to DRAM memory, 4 clocks for FP multiply)
- Old : Increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New CW: "ILP wall" diminishing returns on more ILP
- New: Power Wall + Memory Wall + ILP Wall = Brick Wall
  - Old CW: Uniprocessor performance 2X / 1.5 yrs
  - New CW: Uniprocessor performance only 2X / 5 yrs?

Uniprocessor Performance (SPECint)

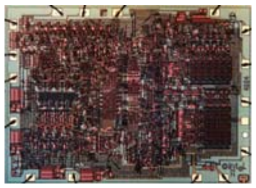


⇒ Sea change in chip design: multiple "cores" or processors per chip

- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

Sea Change in Chip Design

- Intel 4004 (1971): 4-bit processor, 2312 transistors, 0.4 MHz, 10 micron PMOS, 11 mm^2 chip



- RISC II (1983): 32-bit, 5 stage pipeline, 40,760 transistors, 3 MHz, 3 micron NMOS, 60 mm^2 chip

- 125 mm^2 chip, 0.065 micron CMOS = 2312 RISC II+FPU+Icache+Dcache
  - RISC II shrinks to ≈ 0.02 mm^2 at 65 nm
  - Caches via DRAM or 1 transistor SRAM (www.t-ram.com) ?
  - Proximity Communication via capacitive coupling at > 1 TB/s ? (Ivan Sutherland @ Sun / Berkeley)

- **Processor is the new transistor?**

Déjà vu all over again?

"… today's processors … are nearing an impasse as technologies approach the speed of light.."

David Mitchell, *The Transputer: The Time Is Now* (1989)

- Transputer had bad timing (Uniprocessor performance↑) ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"

Paul Otellini, President, Intel (2005)

- All microprocessor companies switch to MP (2X CPUs / 2 yrs) ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs

Manufacturer/Year	AMD/'05	Intel/'06	IBM/'04	Sun/'05
Processors/chip	2	2	2	8
Threads/Processor	1	2	2	4
Threads/chip	2	4	4	32

21st Century Computer Architecture

- Old CW: Since cannot know future programs, find set of old programs to evaluate designs of computers for the future
  - E.g., SPEC2006
- What about parallel codes?
  - Few available, tied to old models, languages, architectures, …
- New approach: Design computers of future for numerical methods important in future
- Claim: key methods for next decade are 7 dwarves (+ a few), so design for them!
  - Representative codes may vary over time, but these numerical methods will be important for > 10 years

Phillip Colella's "Seven dwarfs"

High-end simulation in the physical sciences = 7 numerical methods:

1. Structured Grids (including locally structured grids, e.g. Adaptive Mesh Refinement)
2. Unstructured Grids
3. Fast Fourier Transform
4. Dense Linear Algebra
5. Sparse Linear Algebra
6. Particles
7. Monte Carlo

- If add 4 for embedded, covers all 41 EEMBC benchmarks
  8. Search/Sort
  9. Filter
  10. Combinational logic
  11. Finite State Machine
- Note: Data sizes (8 bit to 32 bit) and types (integer, character) differ, but algorithms the same

Well-defined targets from algorithmic, software, and architecture standpoint

Slide from "Defining Software Requirements for Scientific Computing", Phillip Colella, 2004

6/11 Dwarves Covers 24/30 SPEC

- SPECfp
  - 8 Structured grid
    - » 3 using Adaptive Mesh Refinement
  - 2 Sparse linear algebra
  - 2 Particle methods
  - 5 TBD: Ray tracer, Speech Recognition, Quantum Chemistry, Lattice Quantum Chromodynamics (many kernels inside each benchmark?)
- SPECint
  - 8 Finite State Machine
  - 2 Sorting/Searching
  - 2 Dense linear algebra (data type differs from dwarf)
  - 1 TBD: 1 C compiler (many kernels?)

21st Century Measures of Success

- Old CW: Don't waste resources on accuracy, reliability
  - Speed kills competition
  - Blame Microsoft for crashes
- New CW: SPUR is critical for future of IT
  - Security
  - Privacy
  - Usability (cost of ownership)
  - Reliability
- Success not limited to performance/cost

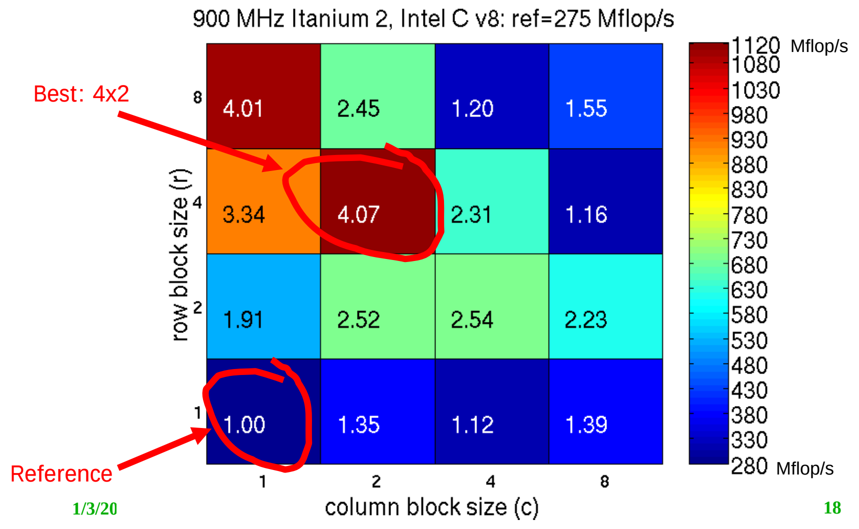
"20th century vs. 21st century C&C: the SPUR manifesto," *Communications of the ACM* , 48:3, 2005.

21st Century Code Generation

- Old CW: Takes a decade for compilers to introduce an architecture innovation
- 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
  - E.g., PHiPAC (BLAS), Atlas (BLAS), Sparsity (Sparse linear algebra), Spiral (DSP), FFT-W
  - Can achieve 10X over conventional compiler
- One Auto-tuner per dwarf?
  - Exist for Dense Linear Algebra, Sparse Linear Algebra, Spectral

Sparse Matrix – Search for Blocking

for finite element problem [Im, Yelick, Vuduc, 2005]



Best Sparse Blocking for 8 Computers

row block size (r)	1	2	4	8
8		Intel Pentium M		Sun Ultra 2, Sun Ultra 3, AMD Opteron
4	IBM Power 4, Intel/HP Itanium	Intel/HP Itanium 2	IBM Power 3	
2				
1				

column block size (c)

- All possible column block sizes selected for 8 computers; How could compiler know?

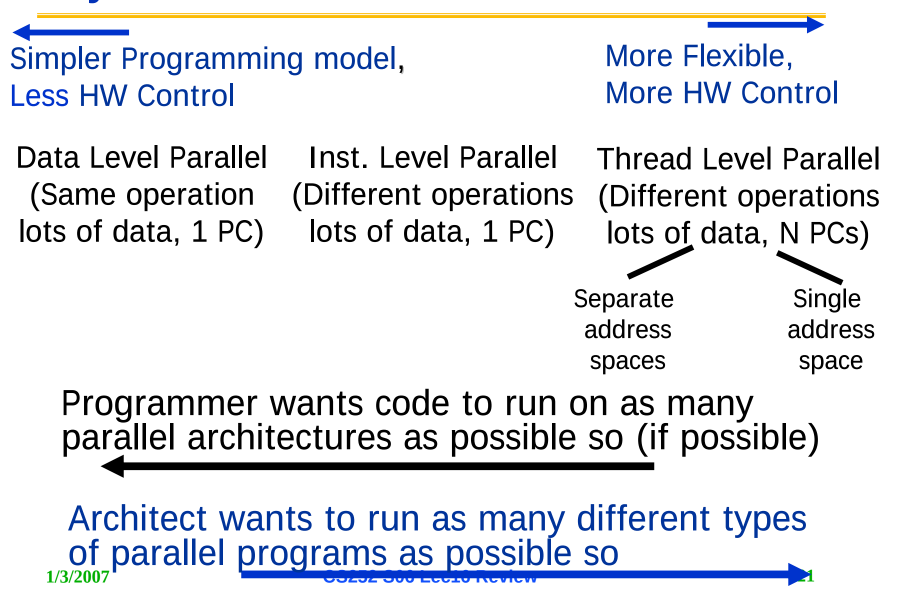
Operand Size and Type

Programmer should be able to specify data size, type independent of algorithm

- 1 bit (Boolean*)
- 8 bits (Integer, ASCII)
- 16 bits (Integer, DSP fixed pt, Unicode*)
- 32 bits (Integer, SP Fl. Pt., Unicode*)
- 64 bits (Integer, DP Fl. Pt.)
- 128 bits (Integer*, Quad Precision Fl. Pt.*)
- 1024 bits (Crypto*)

* Not supported well in most programming languages and optimizing compilers

Style of Parallelism

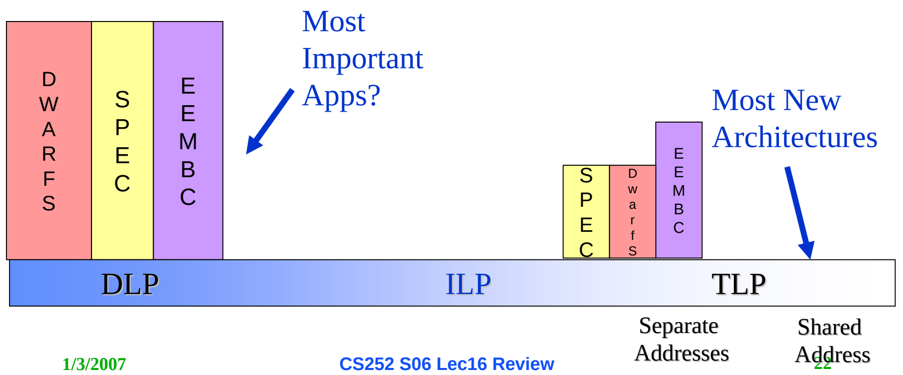

Simpler Programming model,
Less HW Control
More Flexible,
More HW Control
Data Level Parallel (Same operation lots of data, 1 PC)
Inst. Level Parallel (Different operations lots of data, 1 PC)
Thread Level Parallel (Different operations lots of data, N PCs)
Separate address spaces
Single address space

Programmer wants code to run on as many parallel architectures as possible so (if possible)

Architect wants to run as many different types of parallel programs as possible so

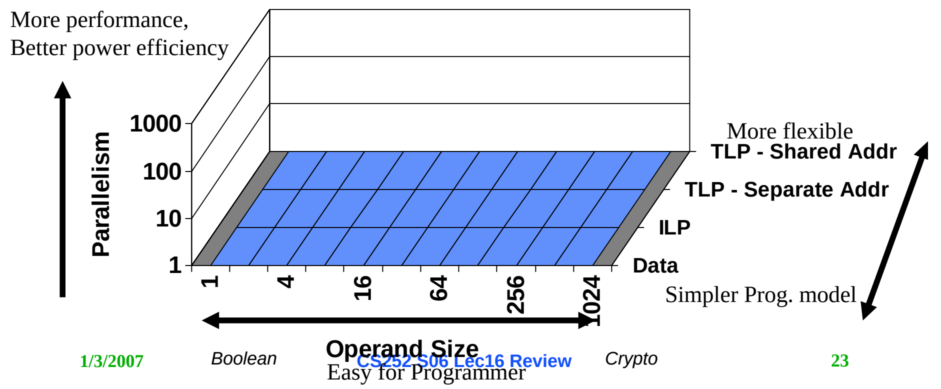
Parallel Framework – Apps (so far)

- Original 7 dwarves: 6 data parallel, 1 Sep. Addr.TLP
- Bonus 4 dwarves: 2 data parallel, 2 Separate Addr. TLP
- EEMBC (Embedded): DLP 19, 12 Separate Addr. TLP
- SPEC (Desktop): 14 DLP, 2 Separate Address TLP



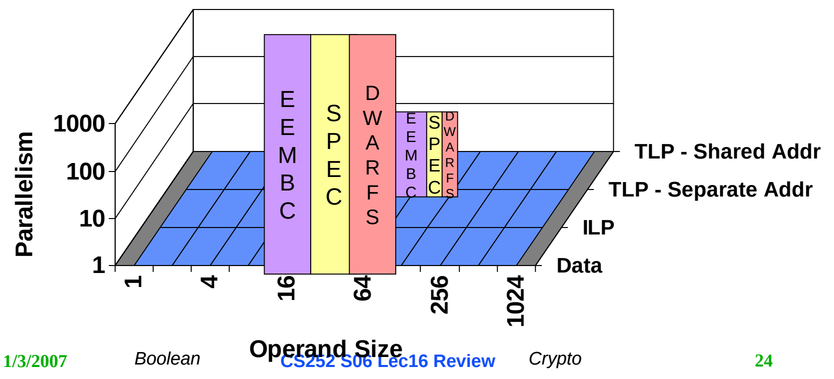
Amount of Explicit Parallelism

- Given natural operand size and level of parallelism, how parallel is computer or how must parallelism available in application?
- Proposed Parallel Framework



Amount of Explicit Parallelism

- Original 7 dwarves: 6 data parallel, 1 Sep. Addr.TLP
- Bonus 4 dwarves: 2 data parallel, 2 Separate Addr. TLP
- EEMBC (Embedded): DLP 19, 12 Separate Addr. TLP
- SPEC (Desktop): 14 DLP, 2 Separate Address TLP



What Computer Architecture brings to Table

- Other fields often borrow ideas from architecture
- Quantitative Principles of Design
  1. Take Advantage of Parallelism
  2. Principle of Locality
  3. Focus on the Common Case
  4. Amdahl's Law
  5. The Processor Performance Equation
- Careful, quantitative comparisons
  - Define, quantity, and summarize relative performance
  - Define and quantity relative cost
  - Define and quantity dependability
  - Define and quantity power
- Culture of anticipating and exploiting advances in technology
- Culture of well-defined interfaces that are carefully implemented and thoroughly checked

1) Taking Advantage of Parallelism

- Increasing throughput of server computer via multiple processors or multiple disks
- Detailed HW design
  - Carry lookahead adders uses parallelism to speed up computing sums from linear to logarithmic in number of bits per operand
  - Multiple memory banks searched in parallel in set-associative caches
- Pipelining: overlap instruction execution to reduce the total time to complete an instruction sequence.
  - Not every instruction depends on immediate predecessor ⇒ executing instructions completely/partially in parallel possible
  - Classic 5-stage pipeline:
    1) Instruction Fetch (Ifetch),
    2) Register Read (Reg),
    3) Execute (ALU),
    4) Data Memory Access (Dmem),
    5) Register Write (Reg)

Three Generic Data Hazards

- Read After Write (RAW)
  Instr$_J$ tries to read operand before Instr$_I$ writes it

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- Caused by a "Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.

Three Generic Data Hazards

- Write After Read (WAR)
  Instr$_J$ writes operand *before* Instr$_I$ reads it

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "anti-dependence" by compiler writers. This results from reuse of the name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Three Generic Data Hazards

- Write After Write (WAW)
  Instr$_J$ writes operand *before* Instr$_I$ writes it.

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "output dependence" by compiler writers
  This also results from the reuse of name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Will see WAR and WAW in more complicated pipes

Software Scheduling to Avoid Load Hazards

Try producing fast code for

a = b + c;

d = e – f;

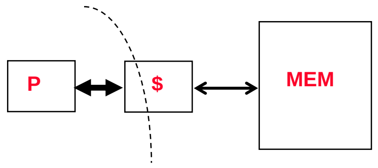
assuming a, b, c, d ,e, and f in memory.

Slow code:		Fast code:	
LW	Rb,b	LW	Rb,b
LW	Rc,c	LW	Rc,c
ADD	Ra,Rb,Rc	LW	Re,e
SW	a,Ra	ADD	Ra,Rb,Rc
LW	Re,e	LW	Rf,f
LW	Rf,f	SW	a,Ra
SUB	Rd,Re,Rf	SUB	Rd,Re,Rf
SW	d,Rd	SW	d,Rd

Compiler optimizes for performance. Hardware checks for safety.

2) The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight-line code, array access)
- Last 30 years, HW relied on locality for memory perf.



3) Focus on the Common Case

- Common sense guides computer design
  - Since its engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster => Amdahl's Law

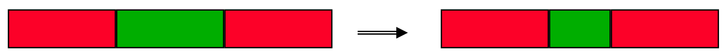
4) Amdahl's Law

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times \left[\left(1 - \text{Fraction}_{enhanced}\right) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right]$$

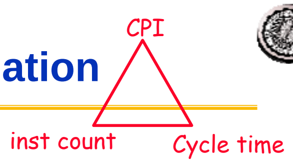
$$\text{Speedup}_{overall} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{\left(1 - \text{Fraction}_{enhanced}\right) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

Best you could ever hope to do:

$$\text{Speedup}_{maximum} = \frac{1}{\left(1 - \text{Fraction}_{enhanced}\right)}$$



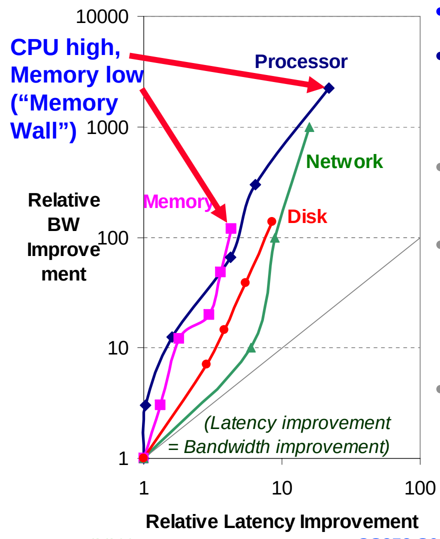
5) Processor performance equation



$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

Latency Lags Bandwidth (last ~20 years)



- **Performance Milestones**
- **Processor: '286, '386, '486, Pentium, Pentium Pro, Pentium 4** (21x,2250x)
- Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s (16x,1000x)
- Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM (4x,120x)
- Disk : 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)

Rule of Thumb for Latency Lagging BW

- **In the time that bandwidth doubles, latency improves by no more than a factor of 1.2 to 1.4**
  (and capacity improves faster than bandwidth)
- **Stated alternatively:**
  Bandwidth improves by more than the square of the improvement in Latency

Define and quantity power (1 / 2)

- For CMOS chips, traditional dominant energy consumption has been in switching transistors, called *dynamic power*

$$Power_{dynamic} = 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched$$

- For mobile devices, energy better metric

$$Energy_{dynamic} = CapacitiveLoad \times Voltage^2$$

- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy
- Capacitive load a function of number of transistors connected to output and technology, which determines capacitance of wires and transistors
- Dropping voltage helps both, so went from 5V to 1V
- To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)

Define and quantity power (2 / 2)

- Because leakage current flows even when a transistor is off, now *static power* important too

$$Power_{static} = Current_{static} \times Voltage$$

- Leakage current increases in processors with smaller transistor sizes
- Increasing the number of transistors increases power even if they are turned off
- In 2006, goal for leakage is 25% of total power consumption; high performance designs at 40%
- Very low power systems even gate voltage to inactive modules to control loss due to leakage

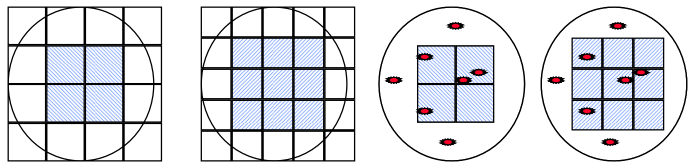
Define and quantity cost ICs

$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die yield}}$$

In 2006: $\alpha = 4$,
12" (30 cm) wafer = \$5k - \$6k,
Defect_Density = 0.4/cm^2

$$\text{Dies per wafer} = \frac{\pi\,(\text{Wafer\_diam}/2)^2}{\text{Die\_Area}} - \frac{\pi \times \text{Wafer\_diam}}{\sqrt{2 \cdot \text{Die\_Area}}} - \text{Test\_Die}$$



$$\text{Die Yield} = \text{Wafer\_yield} \times \left\{1 + \left(\frac{\text{Defect\_Density} \times \text{Die\_area}}{\alpha}\right)\right\}^{-\alpha}$$

- For cost effective dies, cost ≈ f(die_area^2)

Define and quantity dependability

- *Module reliability* = measure of continuous service accomplishment (or time to failure).
 2 metrics

1. *Mean Time To Failure* (*MTTF*) measures Reliability
2. *Failures In Time* (*FIT*) = 1/MTTF, the rate of failures
   - Traditionally reported as failures per billion hours of operation

- *Mean Time To Repair* (*MTTR*) measures Service Interruption
  - *Mean Time Between Failures* (*MTBF*) = MTTF+MTTR
- *Module availability* measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- *Module availability = MTTF / (MTTF + MTTR)*

Example calculating reliability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):

$$\begin{aligned} FailureRate &= 10 \times (1/1{,}000{,}000) + 1/500{,}000 + 1/200{,}000 \\ &= 10 + 2 + 5/1{,}000{,}000 \\ &= 17/1{,}000{,}000 \\ &= 17{,}000 FIT \\ MTTF &= 1{,}000{,}000{,}000/17{,}000 \\ &\approx 59{,}000 hours \end{aligned}$$

How Summarize Suite Performance

- Since ratios, proper mean is geometric mean (SPECRatio unitless, so arithmetic mean meaningless)

$$GeometricMean = \sqrt[n]{\prod_{i=1}^{n} SPECRatio_i}$$

1. Geometric mean of the ratios is the same as the ratio of the geometric means
2. Ratio of geometric means = Geometric mean of performance ratios ⇒ choice of reference computer is irrelevant!

- These two points make geometric mean of ratios attractive to summarize performance

How Summarize Suite Performance

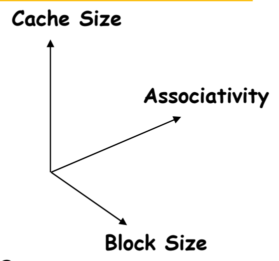
- Does a single mean well summarize performance of programs in benchmark suite?
- Can decide if mean a good predictor by characterizing variability of distribution using standard deviation
- Like geometric mean, geometric standard deviation is multiplicative rather than arithmetic
- Can simply take the logarithm of SPECRatios, compute the standard mean and standard deviation, and then take the exponent to convert back:

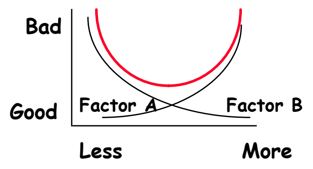
$$GeometricMean = \exp\left(\frac{1}{n} \times \sum_{i=1}^{n} \ln(SPECRatio_i)\right)$$

$$GeometricStDev = \exp(StDev(\ln(SPECRatio_i)))$$

Summary #1/3: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- The optimal choice is a compromise
  - depends on access characteristics
    - » workload
    - » use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins





Summary #2/3: Caches

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
    - » Temporal Locality: Locality in Time
    - » Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!
- Write Policy: Write Through vs. Write Back
- Today CPU time is a function of (ops, cache misses) vs. just f(ops): affects Compilers, Data structures, and Algorithms

Summary #3/3: TLB, Virtual Memory

- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
  - funny times, as most systems can't access all of 2nd level cache without TLB misses!
- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
  1) Where can block be placed?
  2) How is block found?
  3) What block is replaced on miss?
  4) How are writes handled?
- Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is more important than memory hierarchy benefits, but computers insecure

Instruction-Level Parallelism (ILP)

- Basic Block (BB) ILP is quite small
  - BB: a straight-line code sequence with no branches in except to the entry and no branches out except at the exit
  - average dynamic branch frequency 15% to 25% => 4 to 7 instructions execute between a pair of branches
  - Plus instructions in BB likely to depend on each other
- To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks
- Simplest: loop-level parallelism to exploit parallelism among iterations of a loop. E.g.,

```
for (i=1; i<=1000; i=i+1)
        x[i] = x[i] + y[i];
```

Loop-Level Parallelism

- Exploit loop-level parallelism to parallelism by "unrolling loop" either by

1. dynamic via branch prediction or
2. static via loop unrolling by compiler

(Another way is vectors, to be covered later)

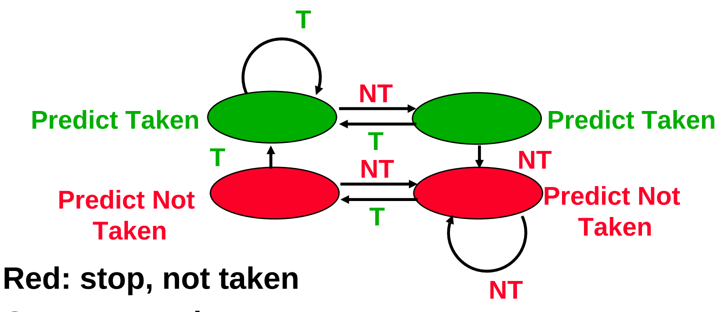
- Determining instruction dependence is critical to Loop Level Parallelism
- If 2 instructions are
  - parallel, they can execute simultaneously in a pipeline of arbitrary depth without causing any stalls (assuming no structural hazards)
  - dependent, they are not parallel and must be executed in order, although they may often be partially overlapped

Dynamic Branch Prediction

- Performance = *ƒ*(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

- Solution: 2-bit scheme where change prediction only if get misprediction *twice*



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

Why can Tomasulo overlap iterations of loops?

- Register renaming
  - Multiple iterations use different physical destinations for registers (dynamic loop unrolling).
- Reservation stations
  - Permit instruction issue to advance past integer control flow operations
  - Also buffer old values of registers - totally avoiding the WAR stall
- Other perspective: Tomasulo building data flow dependency graph on the fly

Tomasulo's scheme offers 2 major advantages

1. Distribution of the hazard detection logic
   - distributed reservation stations and the CDB
   - If multiple instructions waiting on single result, & each instruction has other operand, then instructions can be released simultaneously by broadcast on CDB
   - If a centralized register file were used, the units would have to read their results from the registers when register buses are available
2. Elimination of stalls for WAW and WAR hazards

Tomasulo Drawbacks

- Complexity
  - delays of 360/91, MIPS 10000, Alpha 21264, IBM PPC 620 in CA:AQA 2/e, but not in silicon!
- Many associative stores (CDB) at high speed
- Performance limited by Common Data Bus
  - Each CDB must go to multiple functional units ⇒high capacitance, high wiring density
  - Number of functional units that can complete per cycle limited to one!
    - » Multiple CDBs ⇒ more FU logic for parallel assoc stores
- Non-precise interrupts!
  - We will address this later

Tomasulo

- Reservations stations: *renaming* to larger set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards
  - Allows loop unrolling in HW
- Not limited to basic blocks (integer units gets ahead, beyond branches)
- Helps cache misses as well
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 descendants are Intel Pentium 4, IBM Power 5, AMD Athlon/Opteron, …

ILP

- Leverage Implicit Parallelism for Performance: Instruction Level Parallelism
- Loop unrolling by compiler to increase ILP
- Branch prediction to increase ILP
- Dynamic HW exploiting ILP
  - Works when can't know dependence at compile time
  - Can hide L1 cache misses
  - Code for one machine runs well on another

Limits to ILP

- Most techniques for increasing performance increase power consumption
- The key question is whether a technique is *energy efficient*: does it increase power consumption faster than it increases performance?
- Multiple issue processors techniques all are energy inefficient:
  1. Issuing multiple instructions incurs some overhead in logic that grows faster than the issue rate grows
  2. Growing gap between peak issue rates and sustained performance
- Number of transistors switching = f(peak issue rate), and performance = f(sustained rate), growing gap between peak and sustained performance ⇒ increasing energy per unit of performance

Limits to ILP

- Doubling issue rates above today's 3-6 instructions per clock, say to 6 to 12 instructions, probably requires a processor to
  - Issue 3 or 4 data memory accesses per cycle,
  - Resolve 2 or 3 branches per cycle,
  - Rename and access more than 20 registers per cycle, and
  - Fetch 12 to 24 instructions per cycle.
- Complexities of implementing these capabilities likely means sacrifices in maximum clock rate
  - E.g, widest issue processor is the Itanium 2, but it also has the slowest clock rate, despite the fact that it consumes the most power!

Limits to ILP

Initial HW Model here; MIPS compilers.

Assumptions for ideal/perfect machine to start:

1. *Register renaming* – infinite virtual registers => all register WAW & WAR hazards are avoided
2. *Branch prediction* – perfect; no mispredictions
3. *Jump prediction* – all jumps perfectly predicted (returns, case statements)
   2 & 3 ⇒ no control dependencies; perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis* – addresses known & a load can be moved before a store provided addresses not equal; 1&4 eliminates all but RAW

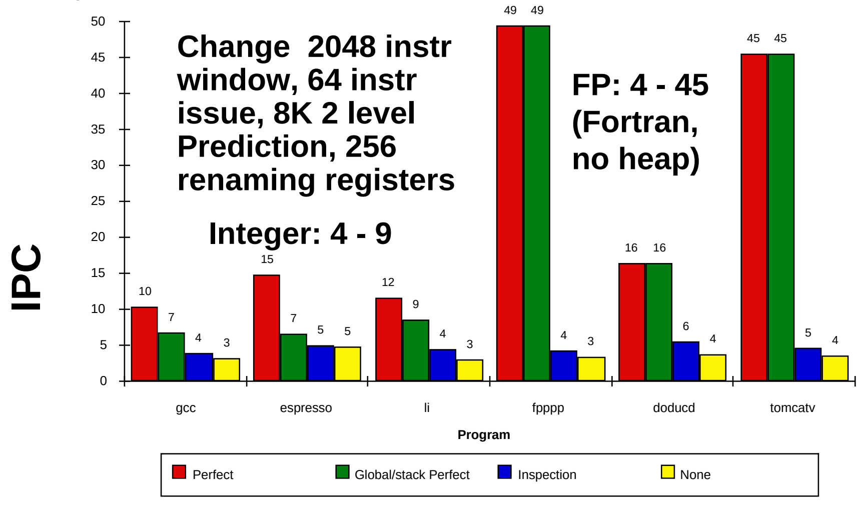
Also: perfect caches; 1 cycle latency for all instructions (FP *,/); unlimited instructions issued/clock cycle;

Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	256 Int + 256 FP	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	8K 2-bit	Perfect	Tournament
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect v. Stack v. Inspect v. none	Perfect	Perfect

More Realistic HW: Memory Address Alias Impact

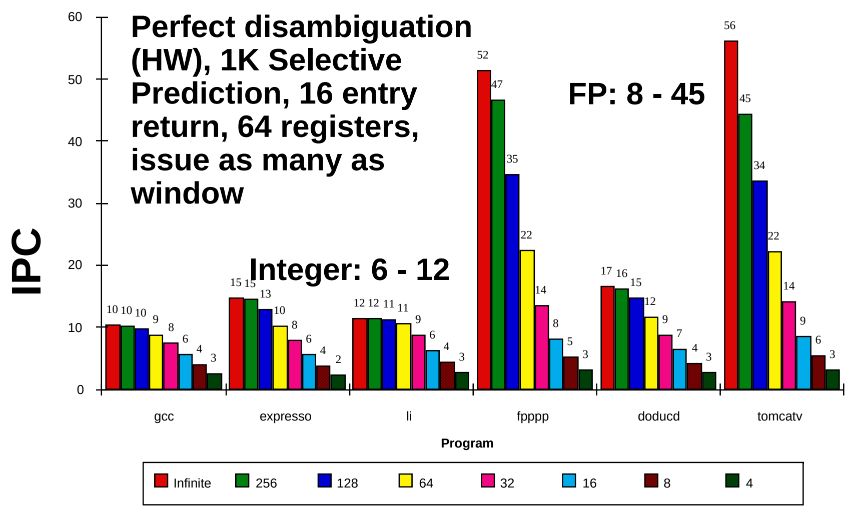
Figure 3.6



Perfect Global/Stack perf; heap conflicts Inspec. Assem. None

Realistic HW: Window Impact

(Figure 3.7)



Infinite 256 128 64 32 16 8 4

Vector Instruction Set Advantages

- Compact
  - one short instruction encodes N operations
- Expressive, tells hardware that these N operations:
  - are independent
  - use the same functional unit
  - access disjoint registers
  - access registers in the same pattern as previous instructions
  - access a contiguous block of memory (unit-stride load/store)
  - access memory in a known pattern (strided load/store)
- Scalable
  - can run same object code on more parallel pipelines or *lanes*

Vector Execution Time

- Time = f(vector length, data dependicies, struct. hazards)
- *Initiation rate*: rate that FU consumes vector elements (= number of lanes; usually 1 or 2 on Cray T-90)
- *Convoy*: set of vector instructions that can begin execution in same clock (no struct. or data hazards)
- *Chime*: approx. time for a vector operation
- *m convoys take m chimes*; if each vector length is n, then they take approx. *m* x *n* clock cycles (ignores overhead; good approximization for long vectors)

```
1:  LV      V1,Rx      ;load vector X
2:  MULV    V2,F0,V1   ;vector-scalar mult.
    LV      V3,Ry      ;load vector Y
3:  ADDV    V4,V2,V3   ;add
4:  SV      Ry,V4      ;store the result
```

4 convoys, 1 lane, VL=64
=> 4 x 64 = 256 clocks
(or 4 clocks per result)

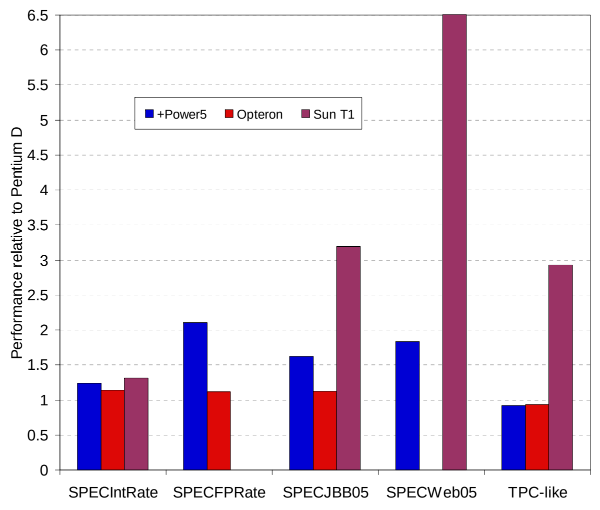
MP and caches

- Caches contain all information on state of cached memory blocks
- Snooping cache over shared medium for smaller MP by invalidating other cached copies on write
- Sharing cached data ⇒ Coherence (values returned by a read), Consistency (when a written value will be returned by a read)
- Snooping and Directory Protocols similar; bus makes snooping easier because of broadcast (snooping => uniform memory access)
- Directory has extra data structure to keep track of state of all cache blocks
- Distributing directory => scalable shared address multiprocessor
  => Cache coherent, Non uniform memory access

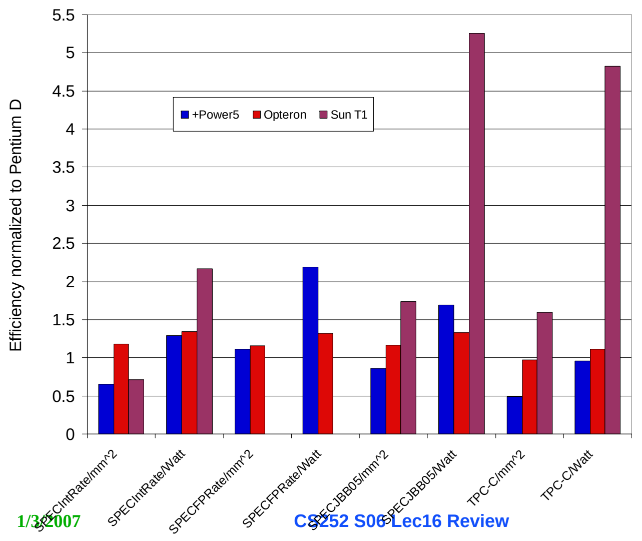
Microprocessor Comparison

Processor	SUN T1	Opteron	Pentium D	IBM Power 5
Cores	**8**	2	2	2
Instruction issues / clock / core	1	3	3	4
Peak instr. issues / chip	**8**	6	6	**8**
Multithreading	Fine-grained	No	SMT	SMT
L1 I/D in KB per core	16/8	**64/64**	12K uops/16	64/32
L2 per core/shared	**3 MB** shared	1MB / core	1MB/ core	1.9 MB shared
Clock rate (GHz)	1.2	2.4	**3.2**	1.9
Transistor count (M)	**300**	233	230	276
Die size (mm^2)	379	199	206	**389**
Power (W)	**79**	110	130	125

Performance Relative to Pentium D



Performance/mm^2, Performance/Watt


Efficiency normalized to Pentium D
+Power5
Opteron
Sun T1
SPECIntRate/mm^2
SPECIntRate/Watt
SPECFPRate/mm^2
SPECFPRate/Watt
SPECJBB05/mm^2
SPECJBB05/Watt
TPC-C/mm^2
TPC-C/Watt