CS252
Graduate Computer Architecture

Lecture 18:
Branch Prediction + analysis resources => ILP

April 2, 2002
Prof. David E. Culler
Computer Science 252
Spring 2002

Today's Big Idea

- Reactive: past actions cause system to adapt use
  – do what you did before better
  – ex: caches
  – TCP windows
  – URL completion, ...
- Proactive: uses past actions to predict future actions
  – optimize speculatively, anticipate what you are about to do
  – branch prediction
  – long cache blocks
  – ???

Review: Case for Branch Prediction when Issue N instructions per clock cycle

1. Branches will arrive up to *n* times faster in an *n*-issue processor
2. Amdahl's Law => relative impact of the control stalls will be larger with the lower potential CPI in an *n*-issue processor

conversely, need branch prediction to 'see' potential parallelism

Review: 7 Branch Prediction Schemes

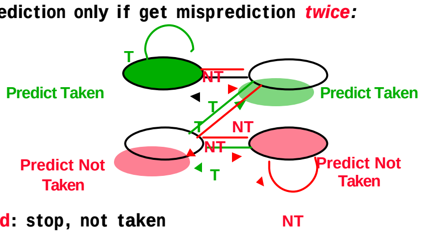
1. 1-bit Branch-Prediction Buffer
2. 2-bit Branch-Prediction Buffer
3. Correlating Branch Prediction Buffer
4. Tournament Branch Predictor
5. Branch Target Buffer
6. Integrated Instruction Fetch Units
7. Return Address Predictors

Review: 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 (saves HW, but may not be right branch)
- Problem: in a loop, 1-bit BHT will cause 2 mispredictions (avg is 9 iterations 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
  – Only 80% accuracy even if loop 90% of the time

Review: Dynamic Branch Prediction
(Jim Smith, 1981)

- Better 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

Consider 3 Scenarios

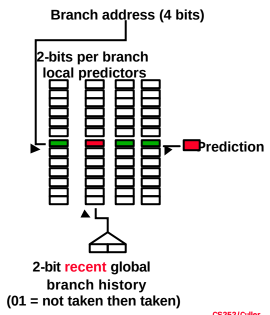
- Branch for loop test
- Check for error or exception
- Alternating taken / not-taken
  - example?

- Your worst-case prediction scenario

Correlating Branches

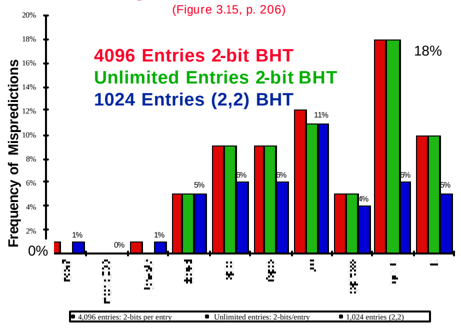
Idea: taken/not taken of recently executed branches is related to behavior of next branch (as well as the history of that branch behavior)
- Then behavior of recent branches selects between, say, 4 predictions of next branch, updating just that prediction

- (2,2) predictor: 2-bit global, 2-bit local


Branch address (4 bits)
2-bits per branch local predictors
Prediction
2-bit recent global branch history
(01 = not taken then taken)

Accuracy of Different Schemes

(Figure 3.15, p. 206)



What's missing in this picture?

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?

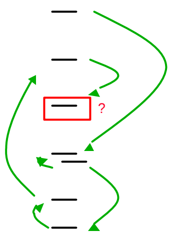
BHT Accuracy

- 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%
- For SPEC92, 4096 about as good as infinite table

Tournament Predictors

- Motivation for correlating branch predictors is 2-bit predictor failed on important branches; by adding global information, performance improved
- Tournament predictors: use 2 predictors, 1 based on global information and 1 based on local information, and combine with a selector
- Hopes to select right predictor for right branch (or right context of branch)

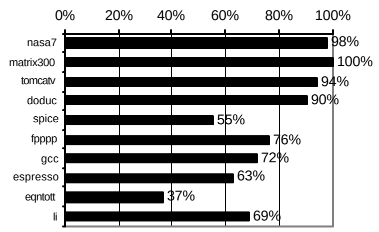
Dynamically finding structure in Spaghetti



Tournament Predictor in Alpha 21264

- 4K 2-bit counters to choose from among a global predictor and a local predictor
- Global predictor also has 4K entries and is indexed by the history of the last 12 branches; each entry in the global predictor is a standard 2-bit predictor
  – 12-bit pattern: ith bit 0 => ith prior branch not taken; ith bit 1 => ith prior branch taken;
- Local predictor consists of a 2-level predictor:
  – Top level a local history table consisting of 1024 10-bit entries; each 10-bit entry corresponds to the most recent 10 branch outcomes for the entry. 10-bit history allows patterns 10 branches to be discovered and predicted.
  – Next level Selected entry from the local history table is used to index a table of 1K entries consisting a 3-bit saturating counters, which provide the local prediction
- Total size: 4K*2 + 4K*2 + 1K*10 + 1K*3 = 29K bits! (~180,000 transistors)

% of predictions from local predictor in Tournament Prediction Scheme



Accuracy of Branch Prediction

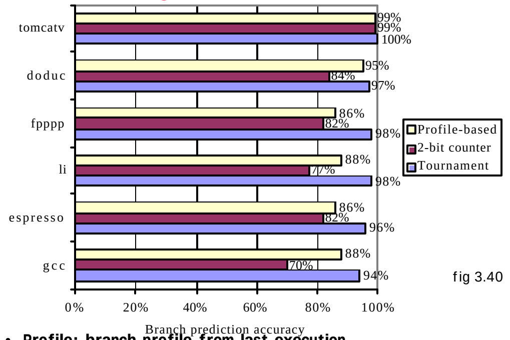
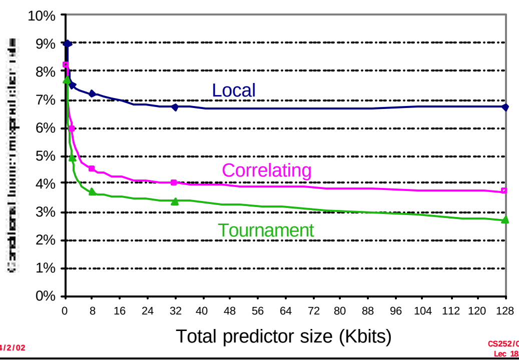


fig 3.40

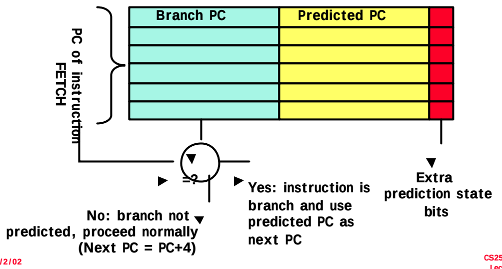
- Profile: branch profile from last execution (static in that in encoded in instruction, but profile)

Accuracy v. Size (SPEC89)



Need Address at Same Time as Prediction

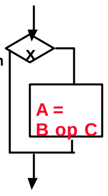
- Branch Target Buffer (BTB): Address of branch index to get prediction AND branch address (if taken)
  – Note: must check for branch match now, since can't use wrong branch address (Figure 3.19, 3.20)


Branch PC
Predicted PC
PC of instruction FETCH
=?
No: branch not predicted, proceed normally (Next PC = PC+4)
Yes: instruction is branch and use predicted PC as next PC
Extra prediction state bits

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

Special Case Return Addresses

- Register Indirect branch hard to predict address
- SPEC89 85% such branches for procedure return
- Since stack discipline for procedures, save return address in small buffer that acts like a stack: 8 to 16 entries has small miss rate

Pitfall: Sometimes bigger and dumber is better

- 21264 uses tournament predictor (29 Kbits)
- Earlier 21164 uses a simple 2-bit predictor with 2K entries (or a total of 4 Kbits)
- SPEC95 benchmarks, 22264 outperforms
  - 21264 avg. 11.5 mispredictions per 1000 instructions
  - 21164 avg. 16.5 mispredictions per 1000 instructions
- Reversed for transaction processing (TP) !
  - 21264 avg. 17 mispredictions per 1000 instructions
  - 21164 avg. 15 mispredictions per 1000 instructions
- TP code much larger & 21164 hold 2X branch predictions based on local behavior (2K vs. 1K local predictor in the 21264)

Dynamic Branch Prediction Summary

- Prediction becoming important part of scalar execution
- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch.
  - Either different branches
  - Or different executions of same branches
- Tournament Predictor: more resources to competitive solutions and pick between them
- Branch Target Buffer: include branch address & prediction
- Predicated Execution can reduce number of branches, number of mispredicted branches
- Return address stack for prediction of indirect jump

Administrivia

- Looking for initial project results next week
- Midterm back thurs
- Homework
- Dan Sorin to talk on April 16 @ 3:30, SafetyNet: Improving the Availability and Designability of Shared Memory

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Vector Processing: Explicit coding of independent loops as operations on large vectors of numbers
  - Multimedia instructions being added to many processors
- Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, Pentium III/4
- (Very) Long Instruction Words (V)LIW: fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates (TBD)
  - Intel Architecture-64 (IA-64) 64-bit address
    - » Renamed: "Explicitly Parallel Instruction Computer (EPIC)"
- Anticipated success of multiple instructions lead to Instructions Per Clock cycle (IPC) vs. CPI

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar MIPS: 2 instructions, 1 FP & 1 anything
  - Fetch 64-bits/clock cycle; Int on left, FP on right
  - Can only issue 2nd instruction if 1st instruction issues
  - More ports for FP registers to do FP load & FP op in a pair

Type	*Pipe Stages*						
Int. instruction	IF	ID	EX	MEM	WB		
FP instruction	IF	ID	EX	MEM	WB		
Int. instruction		IF	ID	EX	MEM	WB	
FP instruction		IF	ID	EX	MEM	WB	
Int. instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB

- 1 cycle load delay expands to 3 instructions in SS
  - instruction in right half can't use it, nor instructions in next slot

Multiple Issue Issues

- issue packet: group of instructions from fetch unit that could potentially issue in 1 clock
  - If instruction causes structural hazard or a data hazard either due to earlier instruction in execution or to earlier instruction in issue packet, then instruction does not issue
  - 0 to N instruction issues per clock cycle, for N-issue
- Performing issue checks in 1 cycle could limit clock cycle time: O(n^2-n) comparisons
  - => issue stage usually split and pipelined
  - 1st stage decides how many instructions from within this packet can issue, 2nd stage examines hazards among selected instructions and those already been issued
  - => higher branch penalties => prediction accuracy important

Multiple Issue Challenges

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations AND No hazards
- If more instructions issue at same time, greater difficulty of decode and issue:
  - Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue; (N-issue ~O(N^2-N) comparisons)
  - Register file: need 2x reads and 1x writes/cycle
  - Rename logic: must be able to rename same register multiple time s in one cycle! For instance, consider 4-way issue:

```
add r1, r2, r3           add p11, p4, p7
sub r4, r1, r2    Þ      sub p22, p11, p4
lw  r1, 4(r4)            lw  p23, 4(p22)
add r5, r1, r2           add p12, p23, p4
```

Imagine doing this transformation in a single cycle!

  - Result buses: Need to complete multiple instructions/cycle
    - » So, need multiple buses with associated matching logic at every reservation station.
    - » Or, need multiple forwarding paths

Dynamic Scheduling in Superscalar The easy way

- How to issue two instructions and keep in-order instruction issue for Tomasulo?
  - Assume 1 integer + 1 floating point
  - 1 Tomasulo control for integer, 1 for floating point
- Issue 2X Clock Rate, so that issue remains in order
- Only loads/stores might cause dependency between integer and FP issue:
  - Replace load reservation station with a load queue; operands must be read in the order they are fetched
  - Load checks addresses in Store Queue to avoid RAW violation
  - Store checks addresses in Load Queue to avoid WAR,WAW

Register renaming, virtual registers versus Reorder Buffers

- Alternative to Reorder Buffer is a larger virtual set of registers and register renaming
- Virtual registers hold both architecturally visible registers + temporary values
  - replace functions of reorder buffer and reservation station
- Renaming process maps names of architectural registers to registers in virtual register set
  - Changing subset of virtual registers contains architecturally visible registers
- Simplifies instruction commit: mark register as no longer speculative, free register with old value
- Adds 40-80 extra registers: Alpha, Pentium,…
  - Size limits no. instructions in execution (used until commit)

How much to speculate?

- Speculation Pro: uncover events that would otherwise stall the pipeline (cache misses)
- Speculation Con: speculate costly if exceptional event occurs when speculation was incorrect
- Typical solution: speculation allows only low-cost exceptional events (1st-level cache miss)
- When expensive exceptional event occurs, (2nd-level cache miss or TLB miss) processor waits until the instruction causing event is no longer speculative before handling the event
- Assuming single branch per cycle: future may speculate across multiple branches!

Limits to ILP

- Conflicting studies of amount
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?
  - Intel MMX, SSE (Streaming SIMD Extensions): 64 bit ints
  - Intel SSE2: 128 bit, including 2 64-bit Fl. Pt. per clock
  - Motorola AltaVec: 128 bit ints and FPs
  - Supersparc Multimedia ops, etc.

Limits to ILP

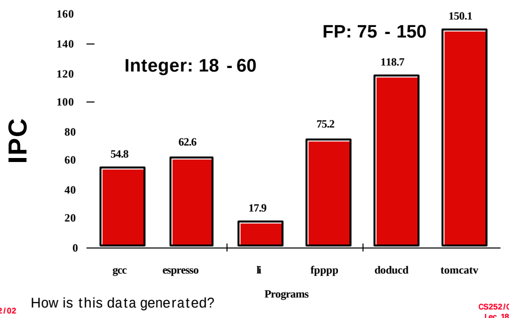
Initial HW Model here; MIPS compilers.
Assumptions for ideal/perfect machine to start:

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
2 & 3 => machine with perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis* – addresses are known & a store can be moved before a load provided addresses not equal

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

Upper Limit to ILP: Ideal Machine

(Figure 3.35 p. 242)

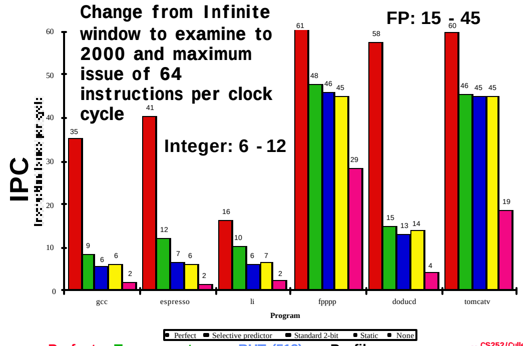


How is this data generated?

More Realistic HW: Branch Impact

Figure 3.37

Change from Infinite window to examine to 2000 and maximum issue of 64 instructions per clock cycle

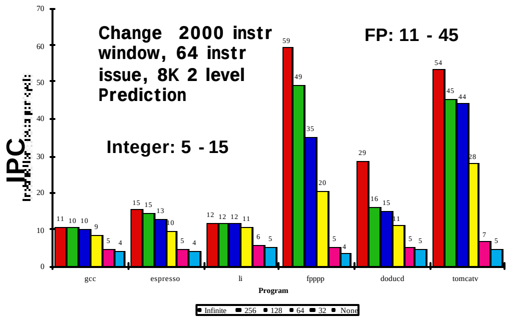


Perfect Tournament BHT (512) Profile No prediction

More Realistic HW: Renaming Register Impact

Figure 3.41

Change 2000 instr window, 64 instr issue, 8K 2 level Prediction

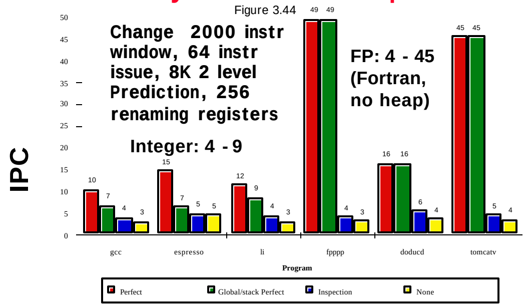


Infinite 256 128 64 32 None

More Realistic HW: Memory Address Alias Impact

Figure 3.44

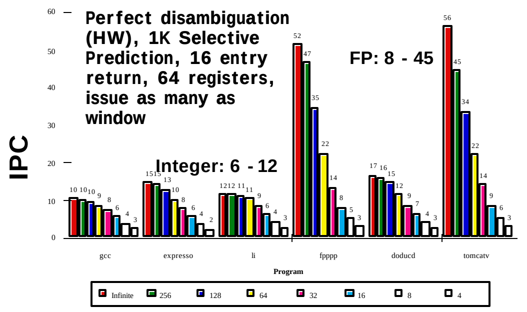
Change 2000 instr window, 64 instr issue, 8K 2 level Prediction, 256 renaming registers



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

Realistic HW: Window Impact

(Figure 3.46)

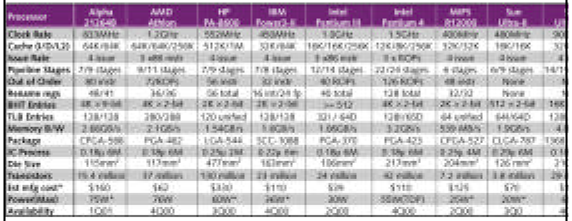


Infinite 256 128 64 32 16 8 4

How to Exceed ILP Limits of this study?

- WAR and WAW hazards through memory
  - eliminated WAW and WAR hazards on registers through renaming, but not in memory usage
- Unnecessary dependences (compiler not unrolling loops so iteration variable dependence)
- Overcoming the data flow limit: value prediction, predicting values and speculating on prediction
  - Address value prediction and speculation predicts addresses and speculates by reordering loads and stores; could provide better aliasing analysis, only need predict if addresses =
- Use multiple threads of control

Workstation Microprocessors 3/2001

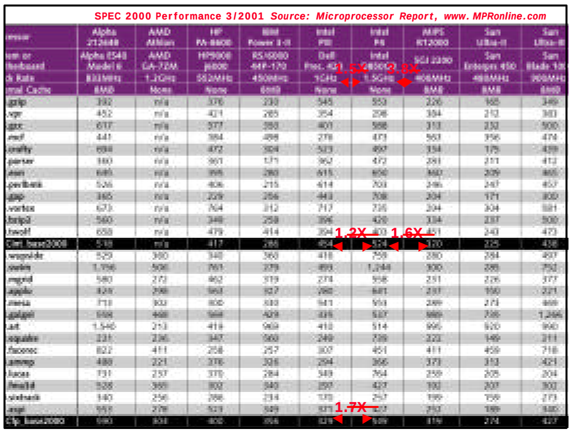


Processor	Alpha 21264B	AMD Athlon	HP PA-8600	IBM Power3-II	Intel Pentium III	Intel Pentium 4	MIPS R12000	Sun Ultra-II	U
Clock Rate	833MHz	1.2GHz	552MHz	450MHz	1.0GHz	1.5GHz	400MHz	480MHz	90
Cache (I/D/L2)	64K/64K	64K/64K/256K	512K/1M	32K/64K	16K/16K/256K	12K/8K/256K	32K/32K	16K/16K	32
Issue Rate	4 issue	3 x86 instr	4 issue	4 issue	3 x86 instr	3 x ROPs	4 issue	4 issue	4
Pipeline Stages	7/9 stages	9/11 stages	7/9 stages	7/8 stages	12/14 stages	22/24 stages	6 stages	6/9 stages	14/1
Out of Order	80 instr	72 ROPs	56 instr	32 instr	40 ROPs	126 ROPs	48 instr	None	N
Rename regs	48/41	36/36	56 total	16 int/24 fp	40 total	128 total	32/32	None	N
BHT Entries	4K x 9-bit	4K x 2-bit	2K x 2-bit	2K x 2-bit	>= 512	4K x 2-bit	2K x 2-bit	512 x 2-bit	16K
TLB Entries	128/128	280/288	120 unified	128/128	32I / 64D	128I/65D	64 unified	64I/64D	128
Memory B/W	2.66GB/s	2.1GB/s	1.54GB/s	1.6GB/s	1.06GB/s	3.2GB/s	539 MB/s	1.9GB/s	4.8
Package	CPGA-588	PGA-462	LGA-544	SCC-1088	PGA-370	PGA-423	CPGA-527	CLGA-787	1368
IC Process	0.18µ 6M	0.18µ 6M	0.25µ 2M	0.22µ 6m	0.18µ 6M	0.18µ 6M	0.25µ 4M	0.29µ 6M	0.18
Die Size	115mm²	117mm²	477mm²	163mm²	106mm²	217mm²	204mm²	126 mm²	21
Transistors	15.4 million	37 million	130 million	23 million	24 million	42 million	7.2 million	3.8 million	29
Est mfg cost*	$160	$62	$330	$110	$39	$110	$125	$70	$
Power(Max)	75W*	76W	60W*	36W*	30W	55W(TDP)	25W*	20W*	6
Availability	1Q01	4Q00	3Q00	4Q00	2Q00	4Q00	2Q00	3Q00	4

- Max issue: 4 instructions (many CPUs)
  Max rename registers: 128 (Pentium 4)
  Max BHT: 4K x 9 (Alpha 21264B), 16Kx2 (Ultra III)
  Max Window Size (OOO): 126 intructions (Pent. 4)
  Max Pipeline: 22/24 stages (Pentium 4)

Source: Microprocessor Report, www.MPRonline.com

SPEC 2000 Performance 3/2001 *Source: Microprocessor Report, www.MPRonline.com*



essor	Alpha 21264B	AMD Athlon	HP PA-8600	IBM Power 3-II	Intel PIII	Intel P4	MIPS R12000	Sun Ultra-II	Sun Ultra-III
em or Motherboard	Alpha ES40 Model 6	AMD GA-7ZM	HP9000 J6000	RS/6000 44P-170	Dell Prec. 420	Intel D850GB	SGI 2200	Sun Enterprise 450	Sun Blade 100
k Rate	833MHz	1.2GHz	552MHz	450MHz	1GHz	1.5GHz	400MHz	480MHz	900MHz
rnal Cache	8MB	None	None	8MB	None	None	8MB	8MB	8MB
gzip	392	n/a	376	230	545	553	226	165	349
vpr	452	n/a	421	285	354	298	384	212	383
gcc	617	n/a	577	350	401	588	313	232	500
mcf	441	n/a	384	498	276	473	563	356	474
crafty	694	n/a	472	304	523	497	334	175	439
parser	360	n/a	361	171	362	472	283	211	412
eon	645	n/a	395	280	615	650	360	209	465
perlbmk	526	n/a	406	215	414	703	246	247	457
gap	365	n/a	229	256	443	708	204	171	300
vortex	673	n/a	764	312	717	735	294	304	581
bzip2	560	n/a	349	258	396	420	334	237	500
twolf	658	n/a	479	414	394	403	451	243	473
CINT_base2000	518	n/a	417	286	454	524	320	225	438
wupwise	529	360	340	360	418	759	280	284	497
swim	1,156	506	761	279	493	1,244	300	285	752
mgrid	580	272	462	319	274	558	231	226	377
applu	424	298	563	327	280	641	237	150	221
mesa	713	302	300	330	541	553	289	273	469
galgel	558	468	569	429	335	537	989	726	1,266
art	1,540	213	419	969	410	514	995	920	990
equake	231	236	347	560	249	739	222	149	211
facerec	822	411	258	257	307	451	411	459	718
ammp	488	221	376	326	294	366	373	313	421
lucas	731	237	370	284	349	764	259	205	204
fma3d	528	365	302	340	297	427	192	207	302
sixtrack	340	256	286	234	170	257	199	159	273
apsi	553	278	523	349	371	427	252	189	445
CFP_base2000	590	304	400	356	329	549	319	274	427

1.5X 3.8X

1.2X 1.6X

1.7X

Conclusion

- 1985-2000: 1000X performance
  - Moore's Law transistors/chip => Moore's Law for Performance/MPU
- Hennessy: industry been following a roadmap of ideas known in 1985 to exploit Instruction Level Parallelism and (real) Moore's Law 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?
  - Otherwise drop to old rate of 1.3X per year?
  - Less than 1.3X because of processor-memory performance gap?
- Impact on you: if you care about performance, better think about explicitly parallel algorithms vs. rely on ILP?