- Bandwidth or throughput
  - Total work done in a given time
  - 10,000-25,000X improvement for processors
  - 300-1200X improvement for memory and disks

- Latency or response time
  - Time between start and completion of an event
  - 30-80X improvement for processors
  - 6-8X improvement for memory and disks

Performance

Comparing Machines/Systems

- Response Time (latency) = T
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?

avg/best case/worst case

- Throughput = γ
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How many queries per minute?

avg

What do we "really" want to know?

--- Which system works best in our larger system?

--- What costs can be traded off?

Time?

- Elapsed Time
  - Counts everything *(disk and memory accesses, I/O, etc.)* → "wall clock"
  - A useful number, but often not good for comparison purposes
    - E.g., OS & multiprogramming time make it difficult to compare CPUs

Depends on load, disk layout, ...

more abstract ↓

- CPU time (CPU = Central Processing Unit = processor)
  - Doesn't count I/O or time spent running other programs
  - Can be broken up into system time and user time → user cpu time

OS
I/O

- Our focus: user CPU time → Time CPU used for our job (+ overhead)
  - Time spent executing the lines of code that are "in" our program
  - Includes arithmetic, memory, and control instructions...

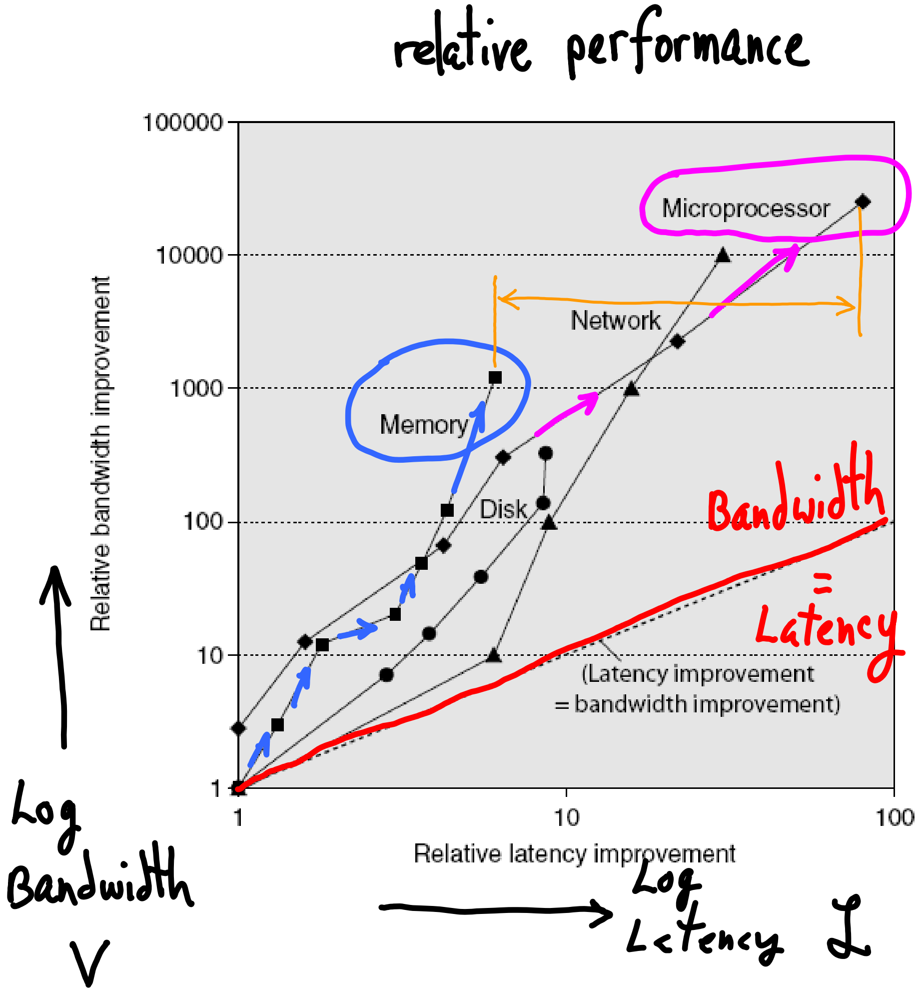
unix ⇒ localhost > time myjob

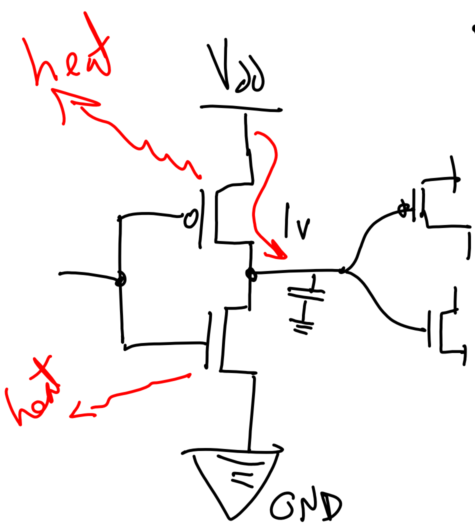
Latency vs. Bandwith

Increasing complexity/density/speed

1. V / L gets worse for each component.

2. Vproc / Lmem gets worse even faster





- Dynamic energy
  - Transistor switch from 0 -> 1 or 1 -> 0
  - ½ x Capacitive load x Voltage^2
- Dynamic power
  - ½ x Capacitive load x Voltage^2 x Frequency switched
- Reducing clock rate reduces power, not energy

$$f_{max} = \propto V$$

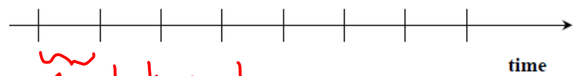
cpu Clock Cycles ⇒ cpu time

- Instead of reporting execution time in seconds, we often use cycles

$$\text{CPU Time} = \left(\frac{\text{seconds}}{\text{program}}\right) = \left(\frac{\text{cycles}}{\text{program}}\right) \times \left(\frac{\text{seconds}}{\text{cycle}}\right)$$

$$T_{cycle} = \left(\frac{\text{seconds}}{\text{Tick}}\right)$$

- Clock "ticks" indicate when to start activities:



1 clock cycle

$$\text{Freq} = \left(\frac{\text{Ticks}}{\text{sec}}\right) = 1/T_{cycle}$$

- Cycle time = time between ticks = seconds per cycle
- Clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)

$$2\text{ GHz clock} \Rightarrow \text{Freq} = \left(\frac{2\times10^9 \text{ ticks}}{\text{sec}}\right) \Rightarrow T_{cycle} = \left(\frac{1 \text{ sec}}{2\times10^9 \text{ Ticks}}\right)$$

$$= \tfrac{1}{2}\text{ ns}$$

$$= \tfrac{1}{2}(1{,}000\text{ ps})$$

$$= 500\text{ ps}$$

ps ← 10^{-12} sec

$$CR \equiv \text{Freq}$$

- User CPU execution time

$$\text{Execution Time} = \boxed{\underbrace{\text{Clock Cycles}}_{\#\text{ cycles}}} \text{ for Program} \times \boxed{\underbrace{\text{Clock Cycle Time}}_{T_{cycle}}}$$

rewrite

- Since Cycle Time is 1/Clock Rate (or clock frequency)

$$\text{Execution Time} = \frac{\text{Clock Cycles for Program}}{\text{Clock Rate}} = \#\text{cycles}\left(\frac{1}{T_{cycle}}\right)^{-1} = \frac{\#\text{cycles}}{CR}$$

- The program should be something real people care about
  - Desktop: MS office, edit, compile
  - Server: web, e-commerce, database
  - Scientific: physics, weather forecasting

real jobs? or benchmarks?

Measuring Clock Cycles

- Clock cycles/program is not an intuitive or easily determined value, so

$$\text{※ } \boxed{\textit{Clock Cycles}} = \textit{Instructions} \times \textit{Clock Cycles Per Instruction} = \text{※instr} \times \left(\text{avg } \frac{\text{※cycles}}{\text{instr}}\right) = \sum_{i=1}^{\text{※instr}} \text{※cycles}_i$$

each instruction is different

[lc3: ADD vs RTI]

- Cycles Per Instruction (CPI) used often

- CPI is an **average** since the number of cycles per instruction varies from instruction to instruction
  - Average depends on instruction mix, latency of each inst. type etc.

- CPIs can be used to compare two implementations of the same ISA, but is not useful alone for comparing different ISAs
  - An x86 add is different from a MIPS add

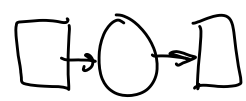
$CPI_{x86} \neq CPI_{mips}$

Depends on instruction mix,
system configuration,
data

$$(\text{※instr} \times CPI) = \text{※cycles} \qquad 1/CR = T_{cycle}$$

- Drawing on the previous equation:

$$\textit{Execution Time} = \underbrace{(\textit{Instructions} \times \textit{CPI})}_{\text{※cycles}} \times \underbrace{\textit{Clock Cycle Time}}_{T_{cycle}}$$



$$\textit{Execution Time} = \frac{\textit{Instructions} \times \textit{CPI}}{\textit{Clock Rate}} = \text{※instr}\left(\frac{CPI}{CR}\right)$$

- To improve performance (i.e., reduce execution time)
  - Increase clock rate (decrease clock cycle time) OR
  - Decrease CPI OR
  - Reduce the number of instructions

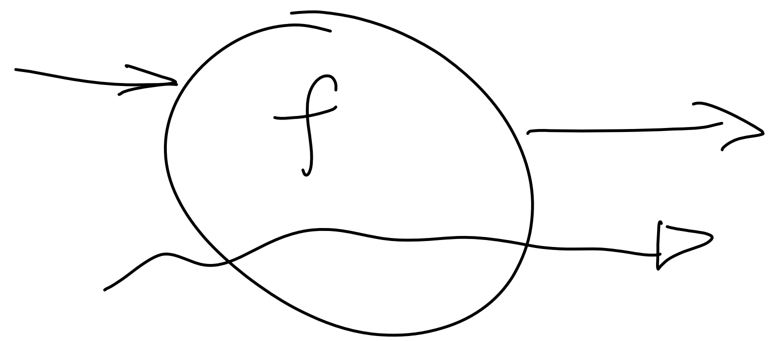
Time ↓

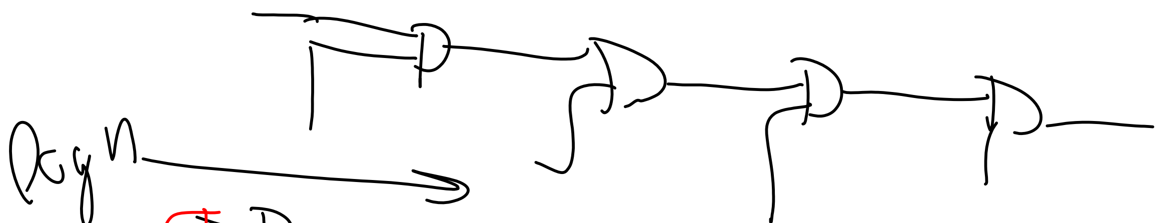
$\Rightarrow CR \uparrow \Rightarrow CPI \downarrow \Rightarrow \text{※instr} \uparrow$

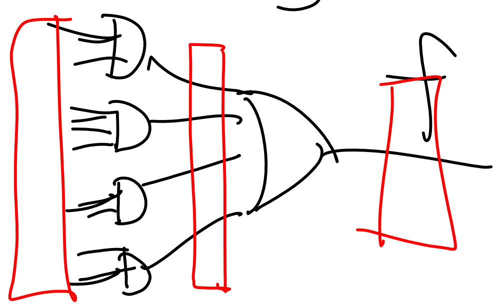
shallow logic — simpler instr — more instructions

- Designers balance cycle time against the number of cycles required
  - Improving one factor may make the other one worse…

[look for sweet spot]


f


log n


f

Clock Rate ≠ Performance

- Mobile Intel Pentium 4 Vs Intel Pentium M
  - 2.4 GHz 1.6 GHz
  - P4 is 50% faster?

$$\frac{CR_{P4}}{CR_{PM}} = \frac{2.4}{1.6} = 1.5$$

- Performance on Mobilemark with same memory and disk
  - Word, excel, photoshop, powerpoint, etc.
  - But – Mobile Pentium 4 is only 15% faster

Instr. Count

$$time = (\#instr)\left(\frac{CPI}{CR}\right)$$

- What is the relative CPI?
  - ExecTime = IC • CPI/Clock rate
  - $\text{ExecTime}_M = 1.15\ \text{ExecTime}_4$
  - $IC \cdot CPI_M/1.6 = 1.15 \cdot IC \cdot CPI_4/2.4$
  - $CPI_4/CPI_M = 2.4/(1.15 \cdot 1.6) = 1.3$

$$T_{P4} = IC_{P4}\left(\frac{CPI_{P4}}{CR_{P4}}\right)$$

$$T_{PM} = IC_{PM}\left(\frac{CPI_{PM}}{CR_{PM}}\right)$$

$$T_{P4}(1.15) = T_{PM}$$

$$\left(IC_{P4}\left(\frac{CPI_{P4}}{CR_{P4}}\right)\right)(1.15) = IC_{PM}\left(\frac{CPI_{PM}}{CR_{PM}}\right)$$

Same ISA

$$IC_{P4} = IC_{PM}$$

and

$$CR_{P4} = (1.5)\,CR_{PM}$$

$$\frac{CPI_{P4}}{(1.5)\,CR_{PM}}(1.15) = \frac{CPI_{PM}}{CR_{PM}}$$

rearrange

$$\frac{CPI_{P4}}{CPI_{PM}} = \frac{(1.5)}{(1.15)} = 1.304\ldots$$

⇒ 30% more cycles/instr on avg for P4

Break it down by classes

- Different instruction types require different numbers of cycles
- CPI is often reported for types of instructions

types

$$Clock\ Cycles = \sum_{i=1}^{n} (CPI_i \times IC_i)$$

#(type i instructions)

(avg CPI for type i instructions)

- where CPI_i is the CPI for the type of instructions and IC_i is the count of that type of instruction

- To compute the overall average CPI use

$$\frac{(\#\ cycles)}{IC} \quad CPI = \sum_{i=1}^{n} \left(CPI_i \times \frac{Instruction\ Count_i}{Instruction\ Count} \right)$$

$$IC = \sum_{1}^{n} IC_i$$

$$= \text{avg CPI}$$

$$\frac{1}{IC} \left(\sum_{1}^{n} \overbrace{(CPI_i \times IC_i)}^{\#Cycles_i} \right)$$

$$= \sum_{1}^{n} CPI_i \times \frac{IC_i}{IC}$$

$$= \sum CPI_i \times \%_i$$

CPI_i $\quad \%_i \quad \Rightarrow \left\{ CPI_i \left(\%_i \right) \right\}$ i^{th} contribution to overall avg CPI

Instruction Type	CPI	Frequency	CPI * Frequency
ALU	1	50%	0.5
Branch	2	20%	0.4
Load	2	20%	0.4
Store	2	10%	0.2

SUM = 1.5 = (avg) CPI

- Given this machine, the CPI is the sum of CPI **X** Frequency
- Average CPI is 0.5 + 0.4 + 0.4 + 0.2 = 1.5

$$= \sum_{i} \#(\text{cycles for type } i) / IC$$

- What fraction of the time for data transfer?

$$= \#\ cycles / IC$$

$$= \text{avg CPI}$$

$$\frac{T_{LD\text{-}ST}}{T_{Total}} = \frac{Cycles_{LD\text{-}ST} * (1/f)}{Cycles_{Total} * (1/f)} = \frac{\#(cycles\ LD\text{-}ST)}{\#\ cycles} = \frac{(\#Cycles\ LD) + (\#cycles\ ST)}{(\#\ cycles) = IC \times CPI}$$

cycle Time

$$= \left[\frac{(\#cycles\ LD)}{IC} + \frac{(\#\ cycles\ ST)}{IC}\right]\frac{1}{CPI}$$

$$= \left[\frac{CPI_{LD} \cdot IC_{LD}}{IC} + \frac{CPI_{ST} \cdot IC_{ST}}{IC}\right]\frac{1}{CPI}$$

$$= \left[CPI_{LD} \cdot \%_{LD} + CPI_{S} \cdot \%_{ST}\right]\left(1/1.5\right)$$

$$= \left[0.4 + 0.2\right](1/1.5) = 40\%$$

Speedup

- Speedup allows us to compare different CPUs or optimizations

$$Speedup = \frac{CPUtimeOld}{CPUtimeNew}$$

- Example
  - Original CPU takes 2sec to run a program
  - New CPU takes 1.5sec to run a program
  - Speedup = 1.333 or speedup or 33%

$T_{old} = 2s$

$T_{new} = 1.5s$

$$S_{new\text{-}old} = T_{old}/T_{new} = 2/1.5 = \frac{20}{15} = 1 + 5/15 = 1.3\ldots$$

$$\Rightarrow \frac{T_{old}}{1.3} = T_{new} \ ? \qquad S = 1/1.3 \ ?$$

Does this look like speedup? What do we mean by speedup?

$$S = \frac{V_{new}}{V_{old}} = \frac{(W_{new}/T_{new})}{(W_{old}/T_{old})} = T_{old}/T_{new} \Rightarrow V_{new} = V_{old}\,(1.3)$$

$\Rightarrow$ new is 30% faster

Assuming $W_{old} = W_{new}$

$$S = \frac{V_{old}}{V_{new}} = \frac{W/T_{old}}{W/T_{new}}$$

$$W = W_{parallel} + W_{sequential} = f \cdot W + (1-f) W$$

$$T = T_{parallel} + T_{sequential} \quad \left(\begin{matrix} parall = improvable \\ sequential = fixed \end{matrix}\right)$$

$$V_p = W_p / T_p \qquad V_s = W_s / T_s$$

old: $V_p = V_s$

new: $V_p = a V_s$ (ie. $S_p = a$)

Amdahl's Law

- If an optimization improves a fraction f of execution time by a factor of a

$$Speedup = \frac{Told}{[(1-f)+f/a]*Told} = \frac{1}{(1-f)+f/a}$$

Is f fixed? $W_p = O(n)$ $W_s = O(1)$

- This formula is known as Amdahl's Law
- Lessons from
  - If f→100%, then speedup = a
  - If a→∞, the speedup = 1/(1-f)
- Summary
  - Make the common case fast
  - Watch out for the non-optimized component

target speed up to max %

$$T_{old} = \frac{f \cdot W}{V_s} + \frac{(1-f)W}{V_s} = W/V_s$$

$$T_{new} = W_p/V_p + W_s/V_s = \frac{f \cdot W}{a V_s} + \frac{(1-f) \cdot W}{V_s} = \left(\frac{f}{a} + (1-f)\right) W/V_s$$

$$S_{new/old} = \frac{T_{old}}{T_{new}} = \frac{(W/V_s)}{(W/V_s)\left(f/a + (1-f)\right)} \Leftarrow$$

$$= 1 / \left(f/a + (1-f)\right)$$

$$\xrightarrow{f=1} 1/(1/a + 0) = a = S_{all\text{-}improved} = S_{max}$$

$$\xrightarrow{f=0} 1/(0/a + 1) = 1 = S_{none\text{-}improved} = S_{min}$$

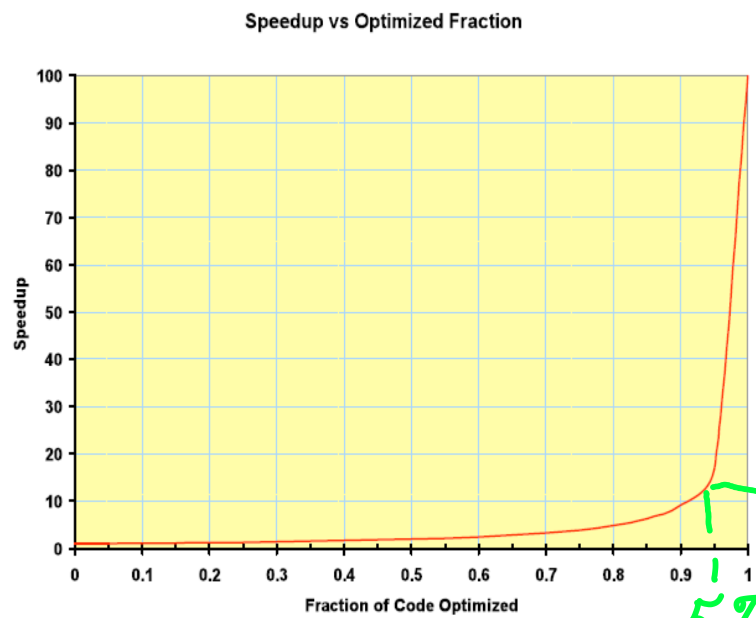
$$\xrightarrow{a \to \infty} 1/\left(f/\infty + (1-f)\right) = \frac{1}{(1-f)} = S_\infty \geq S_{actual}$$

$$f = 50\% \Rightarrow S_\infty = \frac{1}{(1-1/2)} = 2$$

max improvement for $a \to \infty$ is twice as fast

- If a=100, what is the overall speedup as a function of f?

e.g., Use 100 CPUs for $W_{parallel}$



S

f

$S = 17$?

5% sequential

$cost_{new} = 100 \cdot Cost_{old}$

Amdahl's Law Example

- Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?" $= S = 4$

$T_{old} = T_{other} + T_{mult} = 20_s + 80_s = 100_s$

find $S_p = \frac{T_{p\,old}}{T_{p\,new}} = ?$

- How about making it 5 times faster? $S = 5$?

$S_p = \frac{T_p^{old}}{T_p^{new}} = \frac{80}{5} = 16$

$$x \cdot 7 = x(4+2+1) = \underset{\substack{\text{2 shifts}\\ C}}{4x} + \underset{\substack{\text{1 shift}\\ B}}{2x} + \underset{\substack{\text{0 shifts}\\ A}}{x}$$

$y = 00\ldots0111$

$$\begin{array}{rrrrrrrrl}
 & & & 0 & 0 & \cdots & 0 & 0 & = S^0 \\
+ & & & x_n & x_{n-1} & \cdots & x_1 & x_0 & + (A) \\
\hline
 & & & S'_n & S'_{n-1} & \cdots & S'_1 & S'_0 & S' \\
\end{array}$$

shift

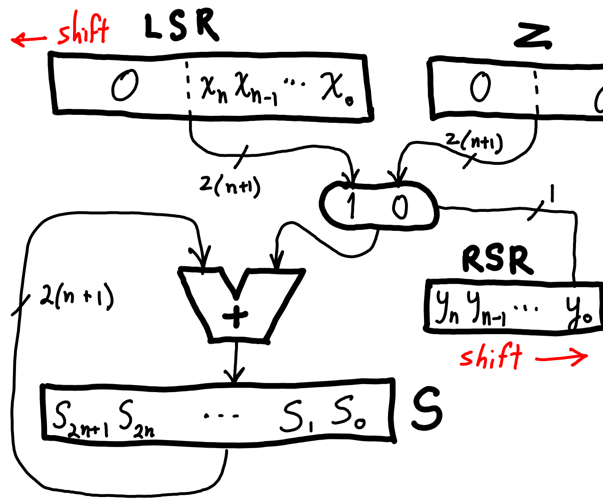
$$\begin{array}{rrrrrrrrl}
+ & & x_n & x_{n-1} & \cdots & x_1 & x_0 & 0 & + (B) \\
\hline
 & & S''_{n+1} & S''_n & \cdots & S''_2 & S''_1 & S''_0 & S'' \\
\end{array}$$

shift

$$\begin{array}{rrrrrrrrrl}
+ & x_n & x_{n-1} & \cdots & x_1 & x_0 & 0 & 0 & & + (C) \\
\hline
 & S'''_{n+2} & S'''_{n+1} & \cdots & S'''_3 & S'''_2 & S'''_1 & S'''_0 & & S''' \\
\end{array}$$

What if y has a 0 bit? Then add 0 instead of shifted x: e.g., y = 0...101 add 0, not **B**.

INT MULTIPLY: S = x * y
LSR: partial products, initially x.
S: partial sum, initially 0.
RSR: initially y.
Z: all 0s



e.g.

$$\begin{array}{r}
1011 \\
\times\ 0101 \\
\hline
\ldots 1011 \\
+\ .1011.. \\
\hline
0\ 0110111
\end{array}$$

(possible carry)

rewrite ⇒

$$\begin{array}{rcccccccclc}
 & & & & & 1 & 0 & 1 & 1 & & \text{multiplier bit} \\
 & & & & \times & 0 & 1 & 0 & 1 & & \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \leftarrow \text{start} & \\
+ & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \leftarrow \text{0-shift (1011)} & \leftarrow y_0 = 1 \\
+ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \leftarrow \text{1-shift ( )} & \leftarrow y_1 = 0 \\
+ & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & \leftarrow \text{2-shift ( )} & \leftarrow y_2 = 1 \\
+ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \leftarrow \text{3-shift ( )} & \leftarrow y_3 = 0 \\
\hline
= & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & &
\end{array}$$

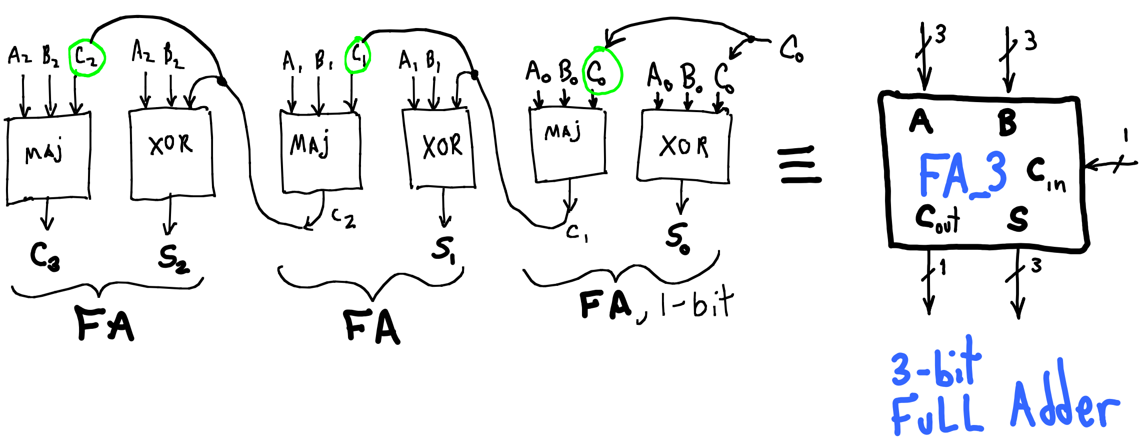
⇒ we shift left (1011) every time, but add either the shifted (1011) or all zeroes, depending on whether y_i is 1 or 0.

ADDER

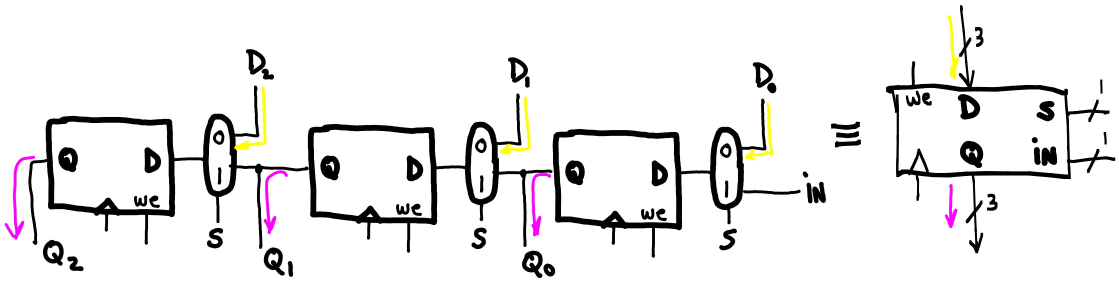
Delay is longest path.

2 levels per MAJ, n bit operands.

2n gate delays until result is ready.



Shift Register Delay = 1 (all stages in parallel)

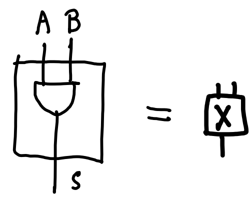


Overall delay: $(n \text{ shifts}) \times (2n \text{ delay}_{\text{ADDER}}) = 2n^2$

Improvement? | Recursive Refinement?

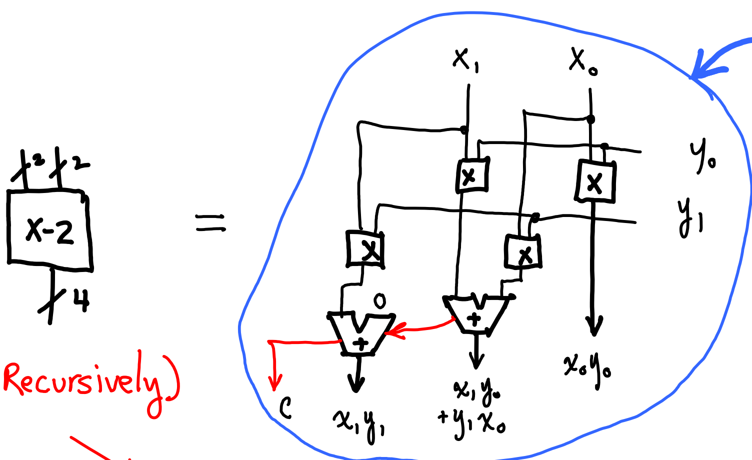
(1-bit MULT)

$$0 \times 0 = 0 \qquad 0 \times 1 = 0 \qquad 1 \times 0 = 0 \qquad 1 \times 1 = 1 \quad \Rightarrow$$



(2-bit MULT)

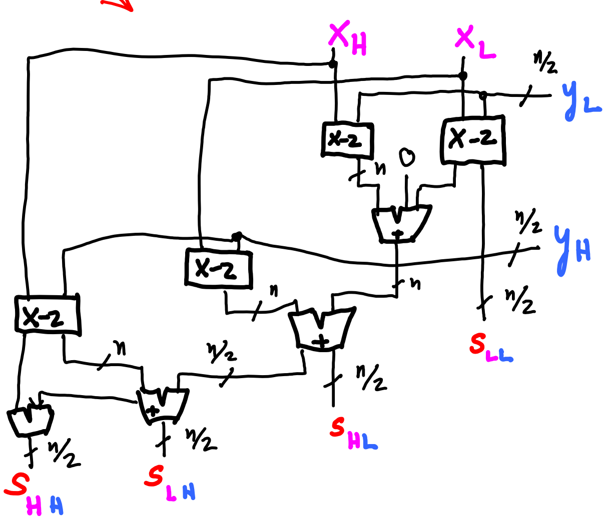
$$\begin{array}{ccccc}
 & & & (x_1 & x_0) \\
 & & \times & (y_1 & y_0) \\
\hline
 & & 0 & 0 & x_0 y_0 \\
+ & & 0 & x_1 y_0 & 0 \\
+ & & 0 & y_1 x_0 & 0 \\
+ & & x_1 y_1 & 0 & 0 \\
\hline
 & S_3 & S_2 & S_1 & S_0
\end{array}$$



(Use Recursively)

(n-bit)

$$\begin{array}{ccccc}
 & & & \boxed{x_3\, x_2} & \boxed{x_1\, x_0} \\
 & & \times & \boxed{y_3\, y_2} & \boxed{y_1\, y_0} \\
\hline
 & 0 & 0 & \multicolumn{2}{c}{\boxed{X_L \cdot Y_L}} \\
 & 0 & \multicolumn{2}{c}{\boxed{X_H \cdot Y_L}} & 0 \\
 & 0 & \multicolumn{2}{c}{\boxed{X_L \cdot Y_H}} & 0 \\
+ & \multicolumn{2}{c}{\boxed{X_H \cdot Y_H}} & 0 & 0 \\
\hline
 & S_{HH} & S_{LH} & S_{HL} & S_{LL}
\end{array}$$



⇒ longest delay, $f(n)$, is,

4 n-bit Add + $(n/2)$-bit MULT delay:

$$f(n) = 4(2n) + f(n/2)$$

$$f(n) = 8n + f(n/2) = 8n + 8(n/2) + f(n/4) \Rightarrow 16n\left(\tfrac{1}{2}+\tfrac{1}{4}+\tfrac{1}{8}+\cdots\right) \approx 16n$$

time improvement: $2n^2 \Rightarrow 16n$ Area increase: $7n \Rightarrow ?$ $A(n) = 4A(n/2) + 4(n)$

32-bit $2(2^5)^2 = 2^{11} \Rightarrow 2^4(2^5) = 2^9$ $S_{32\text{-bit}} = 2^{11}/2^9 = 2^2 = 4$

$S_{64\text{-bit}} = 8$

$S_{128\text{-bit}} = 16$

$$4 = S_{overall} = \frac{V_{new}}{V_{old}} = \frac{W/T_{new}}{W/T_{old}} = \frac{T_{old}}{T_{new}} = \frac{W_s/V_{s\text{-}old} + W_p/V_{p\text{-}old}}{W_s/V_{s\text{-}new} + W_p/V_{p\text{-}new}}$$

OUR Requirement

$$= \frac{(0.2)W/V_{p\text{-}old} + (0.8)W/V_{p\text{-}old}}{(0.2)W/V_{p\text{-}old} + (0.8)W/S_p \cdot V_{p\text{-}old}}$$

[Assume
$V_{p\text{-}old} = V_{s\text{-}old} = V_{s\text{-}new}$
$V_{p\text{-}new} = S_p V_{p\text{-}old}$]

$$= 1/(0.2 + 0.8/S_p) = 4$$

$$\Rightarrow 0.25 = 0.2 + 0.8/S_p$$

$$\Rightarrow 0.05 = 0.8/S_p \quad \Rightarrow S_p = \frac{0.8}{0.05} = \frac{80}{5} = 16 = S_{128\text{-MULT}}$$

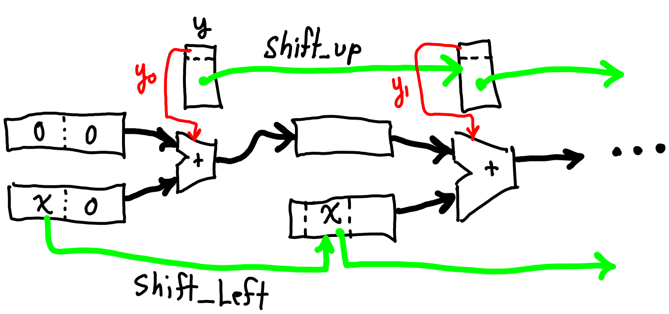
OK!

We can just make it if our data is 128-bit

$S_{overall} = 5$? $1/(0.2 + 0.8/S_p) \stackrel{?}{=} 5$ Can we?

$S_{overall\text{-}\infty} = 1/(0.2 + 0.8/\infty) = 1/0.2 = 5$ Possible? if we eliminate MULT time?

what else could we Try?



n-stage pipeline

1 result every shift

delay $= 4n$ (2n-bit ADD)

(32-bit):

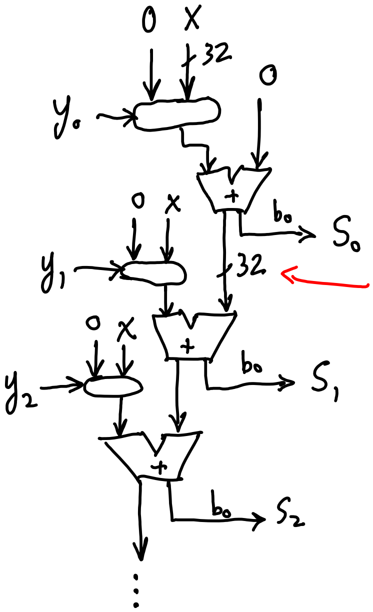
$$S_{32\text{-MULT}} = \frac{2(2^5)^2}{4(2^5)} = 2^4 = 16$$

[Can we keep it full?]

$$\text{Latency} = 4n \times (n) = 4n^2$$

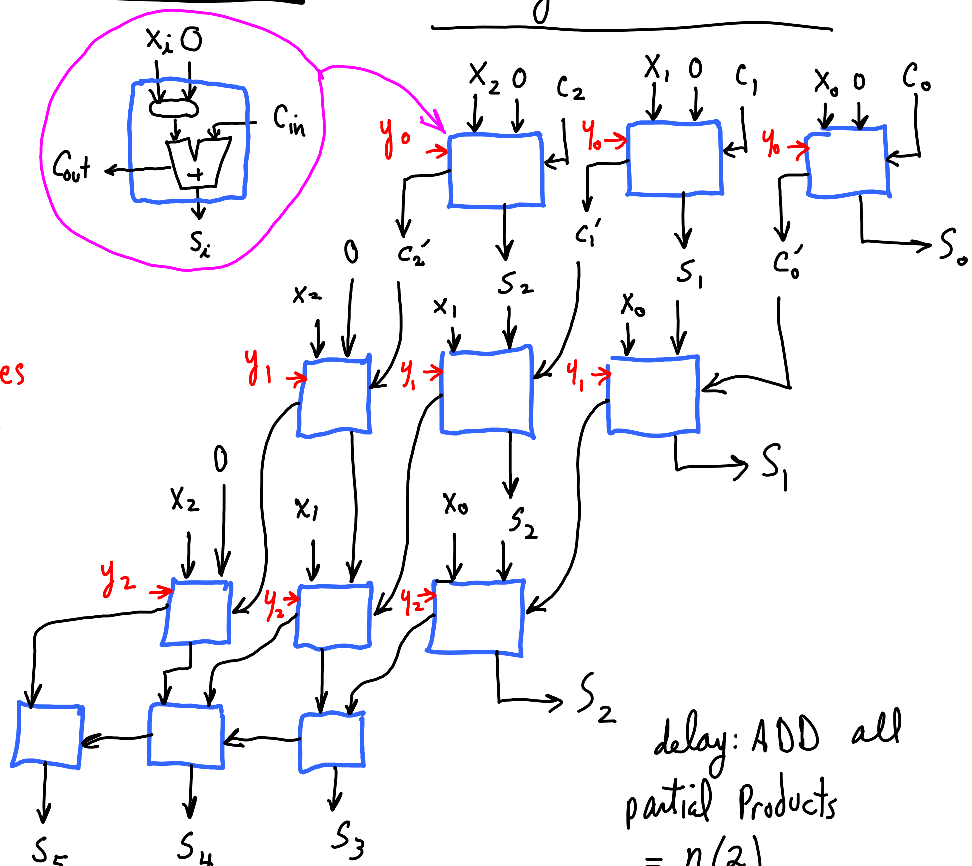
Another Approach? Back to square 1

Adding Partial Products



includes carry

carry-save adder



delay: ADD all partial Products $= n(2)$

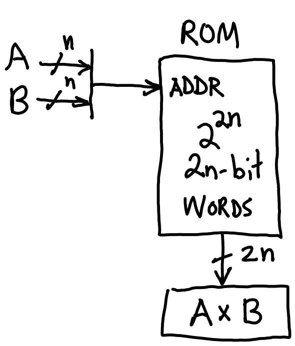
$$S_{32\text{-}MULT} = 2^{11}/2^{6} = 32$$

$$S = 1/(0.2 + 0.8/32)$$

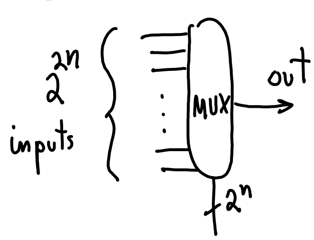
$$= 1/\left((2/10) + (2^3/10)/2^5\right)$$

$$= 10/(2 + 1/4) = 40/(8+1) = \frac{40}{9} = 4 + 4/9$$

Other Ideas?

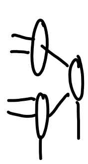


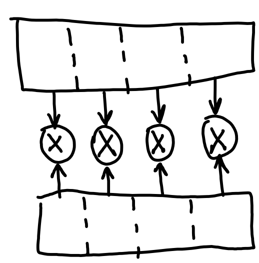
How fast?



delay?

area?





Combine

Do mod arith in parallel, combine results

How?

Evaluating Performance

- Performance best determined by running a real application
  - Use programs typical of expected workload
    - e.g., compilers/editors, scientific applications, graphics, etc.
- Microbenchmarks
  - Small programs, synthetic or kernels from larger applications
  - Nice for architects and designers
  - Can be misleading
- Benchmarks
  - Collection of real programs that companies have agreed on
  - Components: programs, inputs & outputs, measurements, rules, metrics
  - Can still be abused

in typical environment?

⇒ Build compiler optimized for benchmark?
⇒ "Buggy" ⇒ skips work?

The SPEC CPU Benchmark Suite
(System Performance Evaluation Cooperative)

gcc

GO

QM sim

XML parse

speech recog.

SPEC2006 benchmark description	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
	Benchmark name by SPEC generation				
GNU C compiler	←	←	←	←	gcc
Interpreted string processing	←	←	perl		espresso
Combinatorial optimization	←	mcf			li
Block-sorting compression	←	bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

Other Benchmarks

- Scientific computing: Linpack, SpecOMP, SpecHPC, …
- Embedded benchmarks: EEMBC, Dhrystone, …
- Enterprise computing
  - TCP-C, TPC-W, TPC-H
  - SpecJbb, SpecSFS, SpecMail, Streams,
- Other
  - 3Dmark, ScienceMark, Winstone, iBench, AquaMark, …

- Watch out: your results will be as good as your benchmarks
  - Make sure you know what the benchmark is designed to measure
  - Performance is not the only metric for computing systems
    - Cost, power consumption, reliability, real-time performance, …

Summarizing Performance

- Combining results from multiple programs into 1 benchmark score
  - Sometimes misleading, always controversial…and inevitable
  - We all like quoting a single number

- 3 types of means
  - Arithmetic: for times
  - Harmonic: for rates
  - Geometric: for ratios

$$AM = \frac{1}{n}\sum_{i=1}^{n}\left(Weight_i\right)\cdot Time_i$$

$$HM = \frac{\cancel{n}}{\sum_{i=1}^{n}\frac{\left(Weight_i\right)}{Rate_i}}$$

$$GM = \left(\prod_{i=1}^{n} Ratio_i\right)^{\left(\frac{1}{n}\right)}$$

find ratio $\bar{r}$ s.t.

$r_1 \cdot r_2 \cdots r_n x = (\bar{r})^n x$

$$R \Rightarrow (T_{R_1},\ 10\,T_{R_1})$$

We normalize by using a reference machine R to get the speedups w.r.t. benchmark-1 and benchmark-2 (b-1, b-2).

$$S_{A\text{-}R_1} = \frac{T_{R_1}}{100} \qquad S_{A\text{-}R_2} = \frac{10\,T_{R_1}}{4}$$

$$S_{B\text{-}R_1} = \frac{T_{R_1}}{200} \qquad S_{B\text{-}R_2} = \frac{10\,T_{R_1}}{1}$$

Suppose R's time on b-2 is 10 times its time for b-1. (We can always express R's times in terms of one of its benchmark times, no matter how many b's there are.)

To combine our speedups w.r.t R, let's try getting a mean of these for A, and a mean for B, then taking the ratio of those mean speedups.

$$\overline{S_{A\text{-}R}} = \tfrac{1}{2}\left(\frac{T_{R_1}}{100} + \frac{10\,T_{R_1}}{4}\right)$$

$$\overline{S_{B\text{-}R}} = \tfrac{1}{2}\left(\frac{T_{R_1}}{200} + \frac{10\,T_{R_1}}{1}\right)$$

$$\Rightarrow \frac{\overline{S_{A\text{-}R}}}{\overline{S_{B\text{-}R}}} = \frac{T_{R_1}/2\left(\frac{4+1000}{400}\right)}{T_{R_1}/2\left(\frac{1+2000}{200}\right)} = \left(\tfrac{1}{2}\right)\left(\frac{1004}{2001}\right) \cong 1/4\ ?$$

$$\left(\frac{4+100r}{1+200r}\right) \to \tfrac{1}{2} \quad r\to\infty, \qquad \to 4 \quad r \to 0$$

This doesn't seem to have worked very well. The effect of R can make all the difference: if R's time on b-2 had been one-thousandth of its b-1 time, the result would have been an overall speedup of 2. Let's try a different mean, the geometric mean.

$$\overline{S_{A\text{-}R}} = G(S_{A\text{-}R_1}, S_{A\text{-}R_2}) = \sqrt{\left(\frac{T_{R_1}}{100}\right)\left(\frac{10\,T_{R_1}}{4}\right)} = \sqrt{\frac{10\,T_{R_1}^2}{400}} = \frac{(\sqrt{10}\,T_{R_1})}{2\cdot 10}$$

$$\overline{S_{B\text{-}R}} = G(S_{B\text{-}R_1}, S_{B\text{-}R_2}) = \sqrt{\left(\frac{T_{R_1}}{200}\right)\left(\frac{10\,T_{R_1}}{1}\right)} = \sqrt{\frac{10\,T_{R_1}^2}{200}} = \frac{(\sqrt{10}\,T_{R_1})}{\sqrt{2}\cdot 10}$$

$$\frac{\overline{S_{A\text{-}R}}}{\overline{S_{B\text{-}R}}} = \frac{(\sqrt{10}\,T_{R_1})}{(\sqrt{10}\,T_{R_1})}\left(\frac{T_{B_1}\cdot T_{B_2}}{T_{A_1}\cdot T_{A_2}}\right)^{1/2} \cong 0.7 \qquad \left[\text{recall} \quad S_{A\text{-}B_1} = 2,\ S_{A\text{-}B_2} = 1/4\right]$$

weights on R's times cancel.
---- (A's point of view): S_avg in [2, 0.25] ===> 30 % slowdown
---- (B's point of view): S_avg in [0.5, 4] ===> 40 % speedup

at least it's consistent

The real world? Suppose job mix = (**n1** runs of b-1) + (**n2** runs of b-2)

$$S_{A\text{-}B} = \frac{n_1 T_{B_1} + n_2 T_{B_2}}{n_1 T_{A_1} + n_2 T_{A_2}} = \frac{200\,n_1 + n_2}{100\,n_1 + 4\,n_2} = \frac{200 + a}{100 + 4a} \quad (a = n_2/n_1)$$

$$\to 1/4,\ a \to \infty \qquad \to 2,\ a \to 0$$

Sanity check: Given our result above, what **a** does GM appear to assume?

$$\frac{200+a}{(100+4a)} \approx 3/4 \Rightarrow (200+a)4 = (100+4a)3 \Rightarrow 500 = 8a$$

$$a \cong 62$$

$$n_2 = 62\, n_1$$

For every short job (b1)
we do 62 long jobs (b2)?

What if we hadn't taken the SQRT in GM?

$$\bar{S}_{A\text{-}B} = \frac{1}{2} \Rightarrow (200+a)2 = (100+4a) \Rightarrow a = 50$$

$$HM = \frac{1}{\sum r_i} \quad \text{where } r_1 = \frac{W_1}{T_1}, \quad r_2 = \frac{W_2}{T_2} \quad \cdots \qquad [\, r_i = \mathcal{V}_i \,]$$

[find $\bar{r}$ s.t., if we did all work at same rate, takes same time]

$$\frac{(W_1 + W_2 + \cdots W_n)}{\bar{r}} = (T_1 + T_2 + \cdots T_n)$$

$$\bar{r} = \frac{W}{\left(\frac{W_1}{r_1}\right) + \left(\frac{W_2}{r_2}\right) \cdots \left(\frac{W_n}{r_n}\right)}$$

$$= 1 \Big/ \sum \frac{(W_i/W)}{r_i} \qquad (W_i/W) = \omega_i$$

$$W = \sum W_i$$

We have reference Times:

$$T_{R_1} = \frac{W_1}{\mathcal{V}_{R\text{-}1}} \qquad T_{R_2} = W_2/\mathcal{V}_{R\text{-}2} \qquad [\text{assume } \mathcal{V}_{R\text{-}1} = \mathcal{V}_{R\text{-}2} = \mathcal{V}_R]$$

$$W_1 = T_{R_1}\mathcal{V}_R \qquad W_2 = T_{R\text{-}2}\mathcal{V}_R$$

Use that to get weights:

$$\omega_i = W_i \Big/ \sum W_i$$

$$\omega_1 = \frac{W_1}{W_1+W_2} = \frac{T_{R_1}\mathcal{V}_R}{(T_{R_1}\mathcal{V}_R + T_{R_2}\mathcal{V}_R)} = \frac{T_{R_1}}{T_{R_1}+T_{R_2}}$$

$$= 1/(1+10) = 1/11$$

$$\omega_2 = \frac{T_{R_2}}{T_{R_1}+T_{R_2}} = \frac{10}{1+10} = 10/11$$

$$\overline{V}_A = \frac{1}{\sum \omega_i / V_{A-i}}$$

Given our assumption that $V_{R-1} = V_{R-2} = V_R$

$$\frac{W_1 = T_{R-1} V_R}{W_2 = T_{R-2} V_R} = \frac{T_{R-1}}{T_{R-2}}$$

$$= \frac{T_{R-1}}{10 T_{R-1}}$$

$$= 1/10$$

or $W_2 = 10\, W_1$

$$V_{A-1} = \frac{W_1}{T_{A-1}} = \frac{W_1}{100}$$

$$V_{A-2} = \frac{W_2}{T_{A-2}} = \frac{10\, W_1}{4}$$

$$\overline{V}_A = \frac{1}{\left(\frac{(1/11)}{W_1/100} + \frac{(10/11)}{10 W_1/4} \right)} = \frac{(11)}{\frac{100}{W_1} + \frac{10 \cdot 4}{10\, W_1}} = \frac{11\, W_1}{(\underset{T_{A-1}}{100} + \underset{T_{A-2}}{4})}$$

$$\overline{V}_B = \frac{11\, W_1}{(200+1)}$$

$$\longrightarrow \quad S^{HM}_{A-B} = \overline{V}_A / \overline{V}_B = \frac{(200+1)}{(100+4)} \cong 2$$

Suppose, again, n_1 runs of b-1 and n_2 runs of b_2, w/ $a = n_2/n_1$. Comparing our "real world" performance, what a does HM imply?

$$S^{real}_{A-B} = \left(\frac{200 + a}{100 + 4a} \right) = S^{HM}_{A-B} = \frac{2001}{104}$$

$$\Rightarrow 104(200 + a) = 201(100 + 4a)$$

$$\Rightarrow 20800 + 104a = 20100 + 804a$$

$$\Rightarrow 700 = 700a$$

$$\boxed{a = 1}$$

Principles of Computer Design

- Take Advantage of Parallelism
  - e.g. multiple processors, disks, memory banks, pipelining, multiple functional units
- Principle of Locality
  - Reuse of data and instructions
- Focus on the Common Case
  - Amdahl's Law

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

1.02 #bytes per frame, time per file (cache, DRAM, ...)
1.03 avg CPI, CR, performance
1.04-05 CPI by class, CR, instr. mix,
1.06 compilers, avg CPI, CR, speedup, CPI by class, peak performance versus
1.07 Voltage scaling laws, C, power, GM, %change,
1.08 dynamic power, C, V
1.09 static and dynamic power, voltage dependence
1.10 multi-cores, #instructions, CPIs, execution time, power
1.11 die yield and cost
1.12 SPEC ratio from times
1.13 Faster clock, change ISA ==> fewer instructions executed, CPI vs CR
1.14 Performance measured by MFLOPS or MIPS versus overall
1.15 Amdahl's Law (improving only a fraction)
1.16 Speedup w/ communication costs