Multicore & Parallel Processing

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

P&H Chapter 4.10-11, 7.1-6

Q: How to improve system performance?

→ Increase CPU clock rate?

→ But I/O speeds are limited

Disk, Memory, Networks, etc.

Recall: Amdahl's Law

Solution: Parallelism

Pipelining: execute multiple instructions in parallel

Q: How to get more instruction level parallelism?

A: Deeper pipeline

- Less work per stage $\Rightarrow$ shorter clock cycle

A: Multiple issue pipeline

- Start multiple instructions per clock cycle in duplicate stages
- Example: 1GHz 4-way multiple-issue

  Peak CPI = 0.25 → 16 billion instructions per second

Static Multiple Issue

a.k.a. Very Long Instruction Word (VLIW)

Compiler groups instructions to be issued together

- Packages them into “issue slots”
- Simple HW: Compiler detects and avoids hazards

Example: Static Dual-Issue MIPS

- Instructions come in pairs (64-bit aligned)
  - One ALU/branch instruction (or nop)
  - One load/store instruction (or nop)

Compiler scheduling for dual-issue MIPS…

```
TOP:   lw   $t0, 0($s1)          # $t0 = A[i]
       lw   $t1, 4($s1)          # $t1 = A[i+1]
       addu $t0, $t0, $s2        # add $s2
       addu $t1, $t1, $s2        # add $s2
       sw   $t0, 0($s1)          # store A[i]
       sw   $t1, 4($s1)          # store A[i]
       addi $s1, $s1, +8         # increment pointer
       bne  $s1, $s3, TOP        # continue if $s1!=end
```

	ALU/branch slot	Load/store slot	cycle
TOP:	nop	lw $t0, 0($s1)	1
	addu $s1, $s1, +8	lw $t1, 4($s1)	2
	addu $t0, $t0, $s2	nop	3
	addu $t1, $t1, $s2	sw $t0, -8($s1)	4
	bne $s1, $s3, TOP	sw $t1, -4($s1)	5

Dynamic Multiple Issue

a.k.a. SuperScalar Processor

- CPU examines instruction stream and chooses multiple instructions to issue each cycle
- Compiler can help by reordering instructions….
- … but CPU is responsible for resolving hazards

Even better: Speculation/Out-of-order Execution

- Guess results of branches, loads, etc.
- Execute instructions as early as possible
- Roll back if guesses were wrong
- Don’t commit results until all previous insts. are retired

Q: Does multiple issue / ILP work?

A: Kind of… but not as much as we'd like

Limiting factors?

- Programs dependencies
- Hard to detect dependencies → be conservative
  - e.g. Pointer Aliasing: A[0] += 1; B[0] *= 2;
- Hard to expose parallelism
  - Can only issue a few instructions ahead of PC
- Structural limits
  - Memory delays and limited bandwidth
- Hard to keep pipelines full

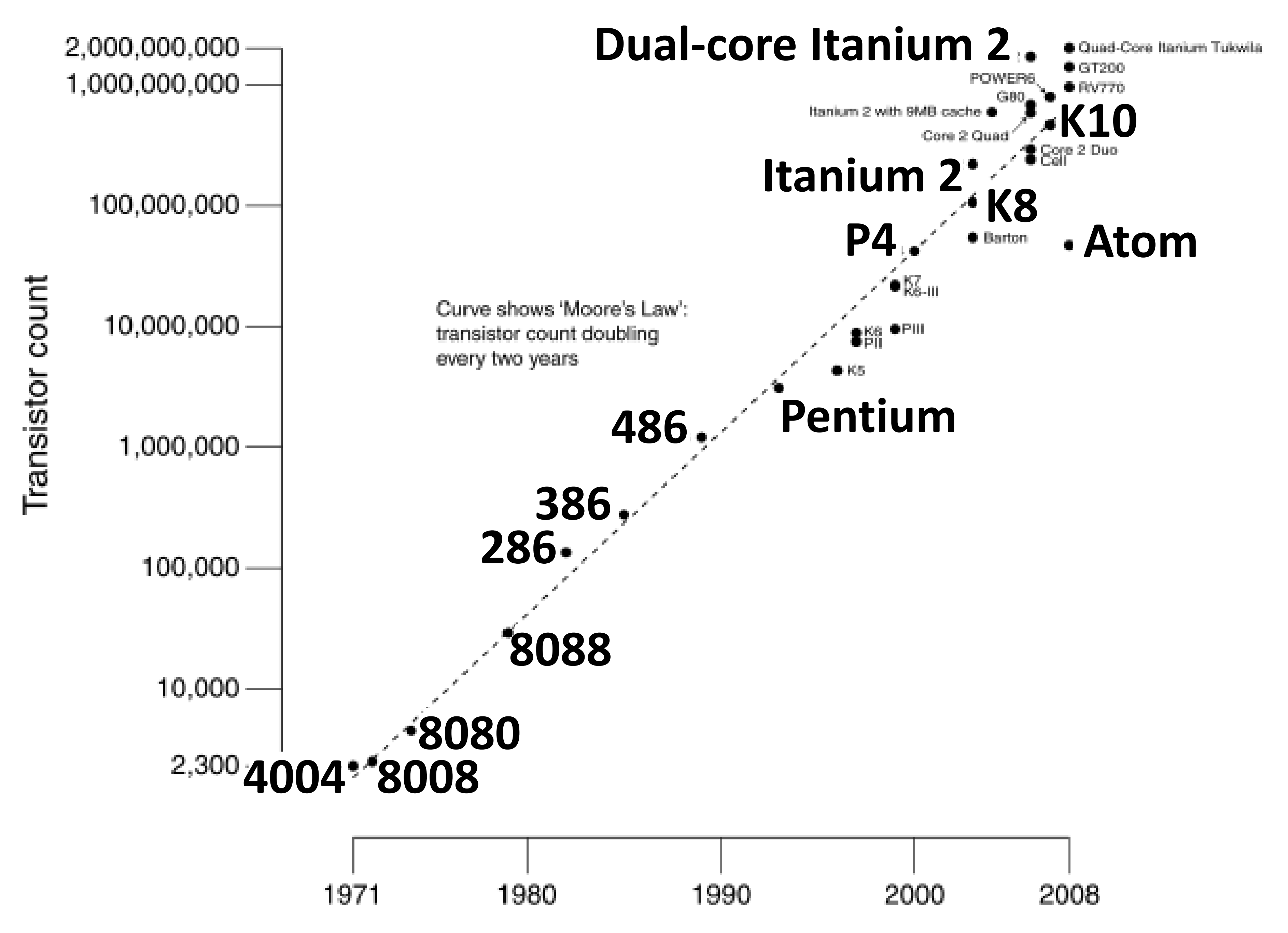
Q: Does multiple issue / ILP cost much?

A: Yes.

→ Dynamic issue and speculation requires power

CPU	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

→ Multiple simpler cores may be better?



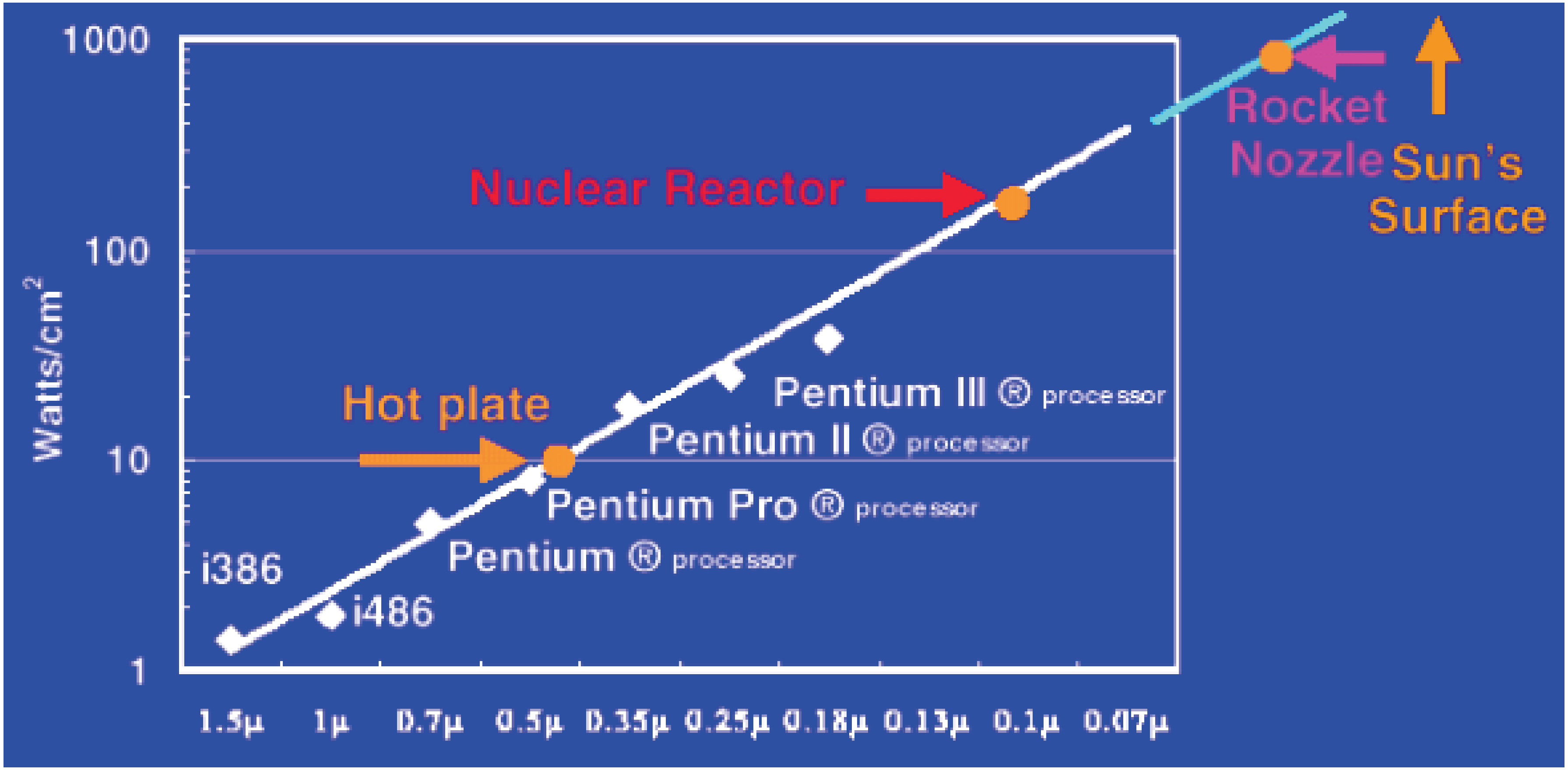
Transistor count
2,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
1971
1980
1990
2000
2008
Curve shows 'Moore's Law': transistor count doubling every two years
Dual-core Itanium 2
Quad-Core Itanium Tukwila
GT200
POWER6
RV770
G80
Itanium 2 with 9MB cache
K10
Core 2 Quad
Core 2 Duo
Cell
Itanium 2
K8
Barton
Atom
P4
K7
K6-III
K6
PIII
PII
K5
Pentium
486
386
286
8088
8080
4004
8008

Moore's law

- A law about transistors
- Smaller means more transistors per die
- And smaller means faster too

But: Power consumption growing too…

Power Limits Performance



Power = capacitance * voltage^2 * frequency

In practice: Power ~ voltage^3

Reducing voltage helps (a lot)

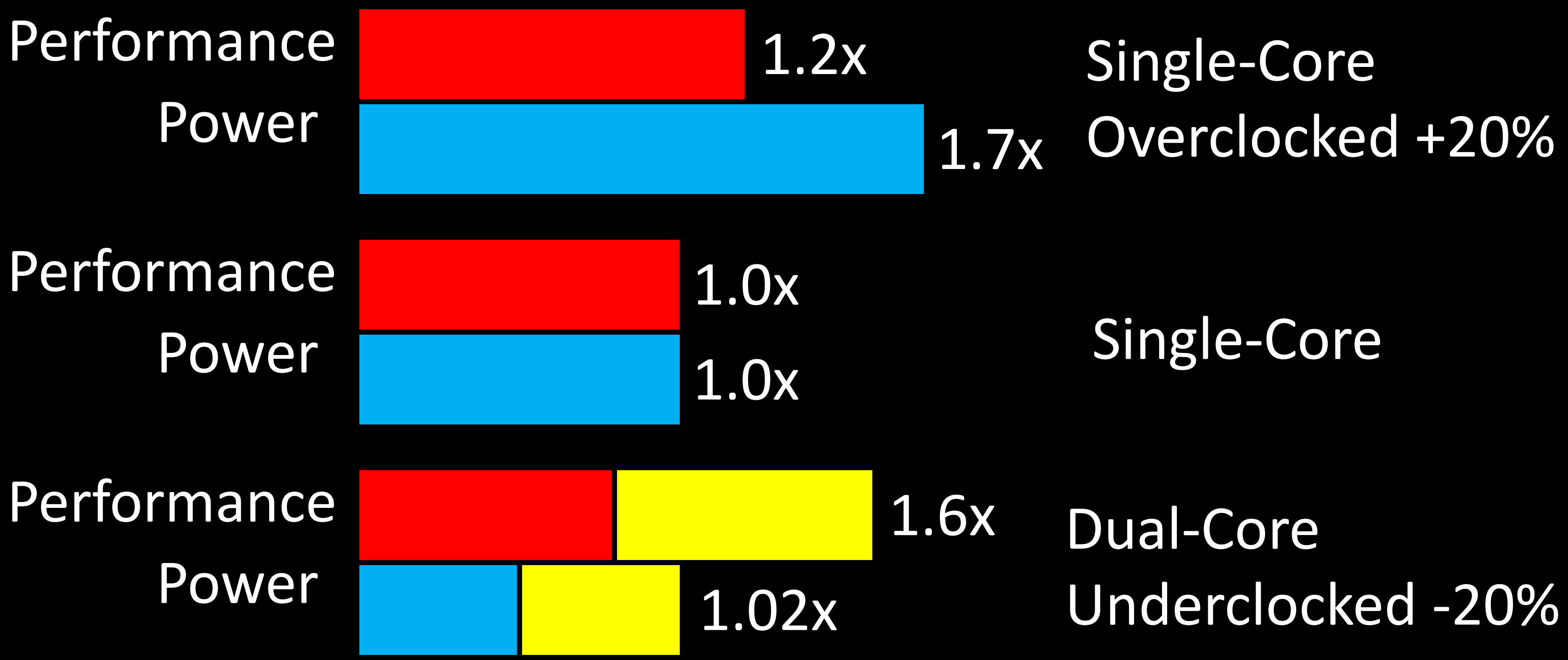
... so does reducing clock speed

Better cooling helps

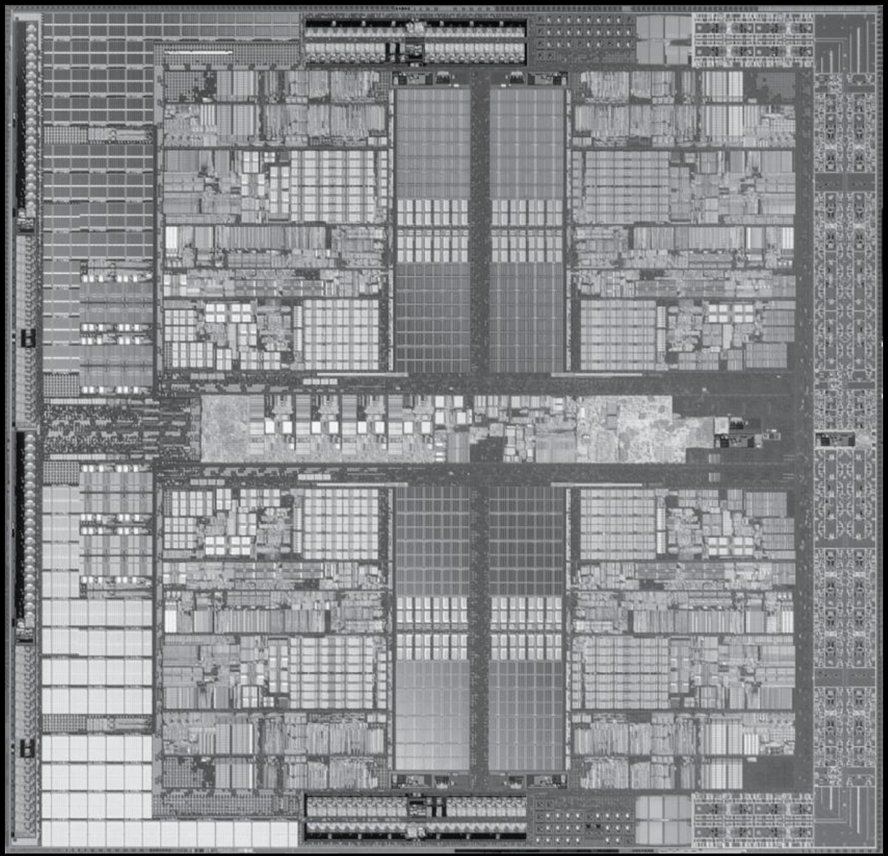
The power wall

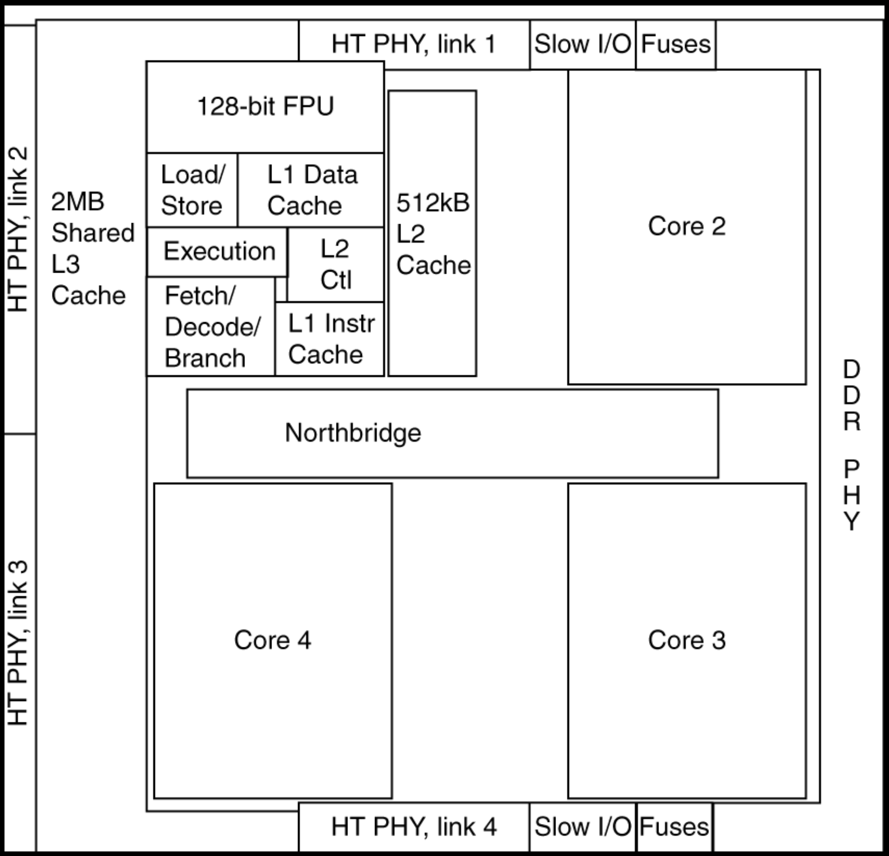
- We can’t reduce voltage further
- We can’t remove more heat

Why Multicore?



AMD Barcelona: 4 processor cores





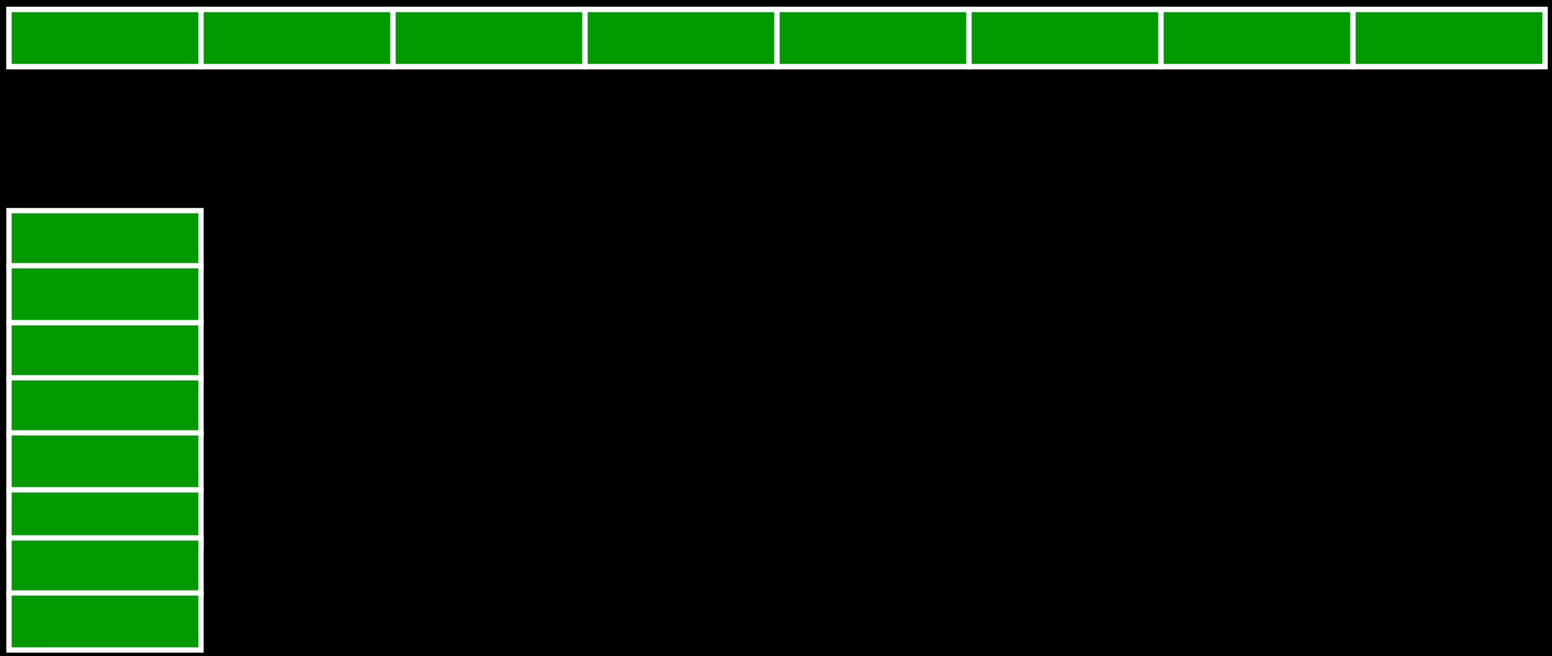
Q: So lets just all use multicore from now on!

A: Software must be written as parallel program

Multicore difficulties

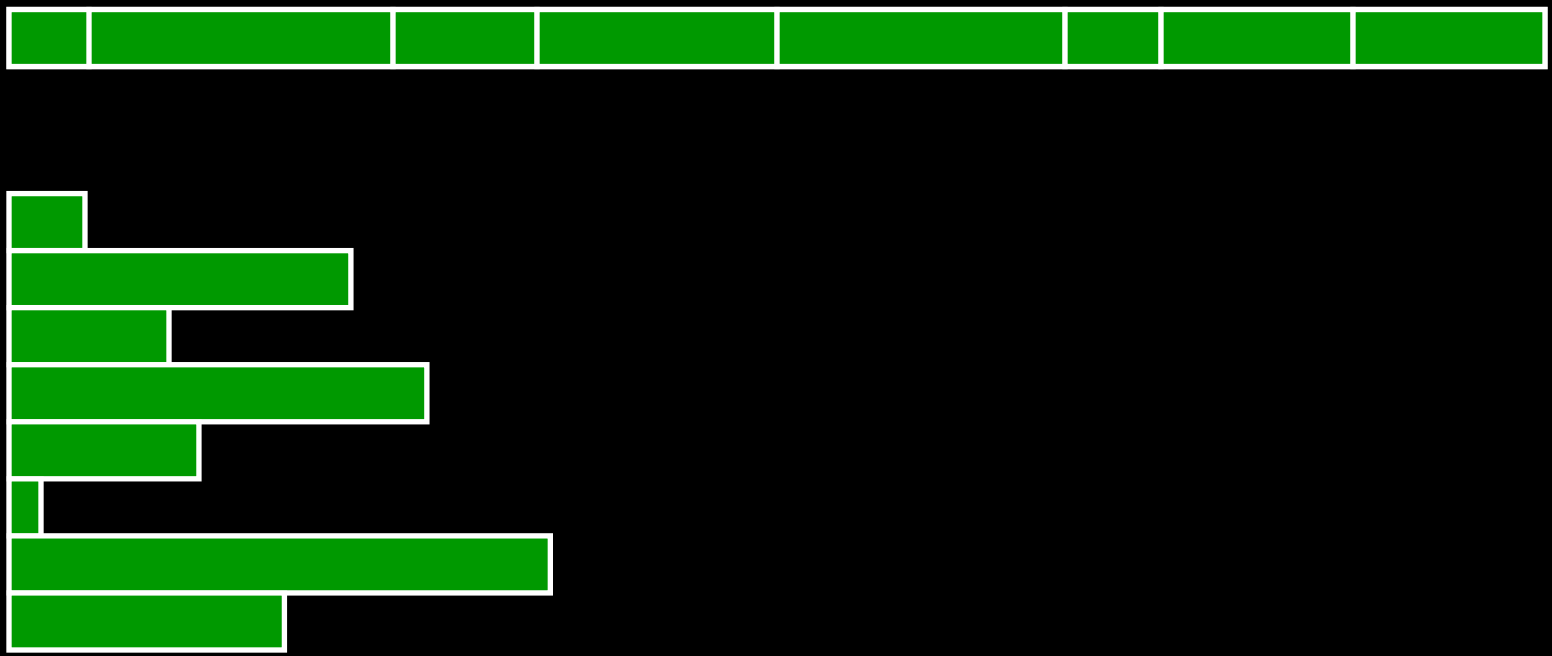
- Partitioning work
- Coordination & synchronization
- Communications overhead
- Balancing load over cores
- How do you write parallel programs?
  - ... without knowing exact underlying architecture?

Partition work so all cores have something to do



Load Balancing

Need to partition so all cores are actually working



If tasks have a serial part and a parallel part…

Example:

step 1: divide input data into *n* pieces

step 2: do work on each piece

step 3: combine all results

Recall: Amdahl's Law

As number of cores increases …

- time to execute parallel part?
- time to execute serial part?