Overview

Week 1, Lecture 1

VirginiaTech
Invent the Future

Course Goals

- Philosophy
  - A lot of theory that we could go into … but instead …
  - Pragmatic Approach
    - Focus on learning principles of concurrent programming in multicore architectures.
- "Hands-On" Course
  - Write correct and efficient multi-threaded code in C or Java using all the "state-of-the-art" tools and techniques
  - Understand how to accelerate code via multi-threading
  - Have notions of and some experience with distributed memory computing

Course Objectives

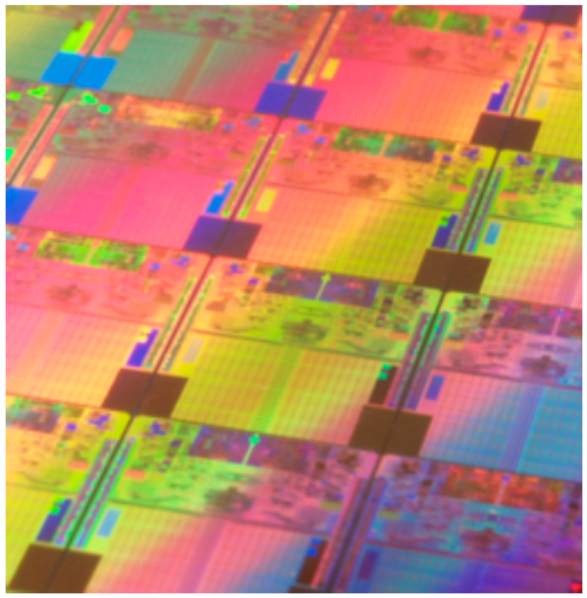
- Design applications software for chip multiprocessors (CMPs), e.g., traditional & massively parallel multicore.
- Understand fundamental differences in writing parallel code for traditional vs. massively parallel multicore.
- Understand importance of memory and network subsystems in emerging CMPs.
- Understand how to transform (amenable) serial code into GPGPU-accelerated code.
- Learn about parallelism models, communication models, and resource limitations of CMPs.

Course Objectives (continued)

- Explain the different layers of parallelism in a CMP.
- Understand the relationship between each of the above layers of abstraction, and more generally, the relationship between the CMP hardware and application software.

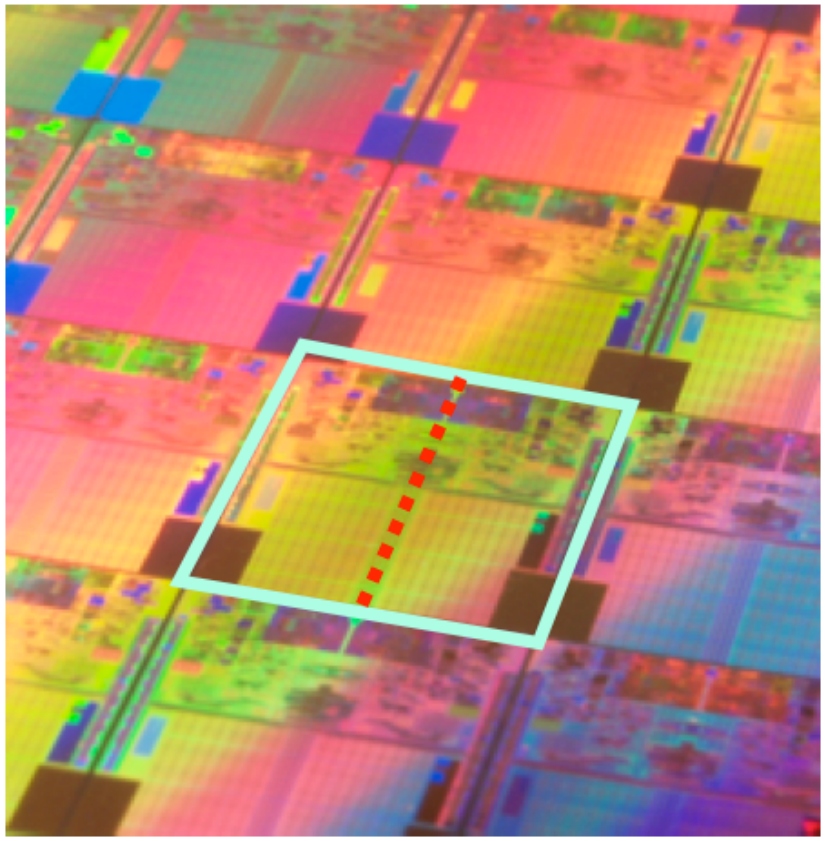
What is Multicore?

- A multicore chip is a single silicon die containing multiple fully functional sequential processor cores tied together to form a small parallel computer.



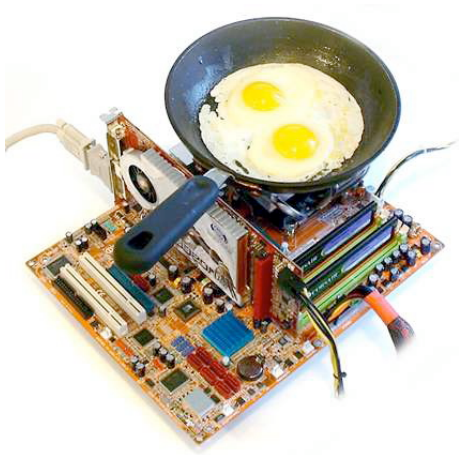
Source: Intel Corporation

A Closer Look at Multicore



- Multicore Highlighted
  - Intel Core 2 processor on a larger silicon wafer.
- Anatomy
  - Two cores of Intel Core 2 processor separated by a red line.
  - Cache is the very regular portion. Light colored.
  - Functional units are up above. Dark colored.

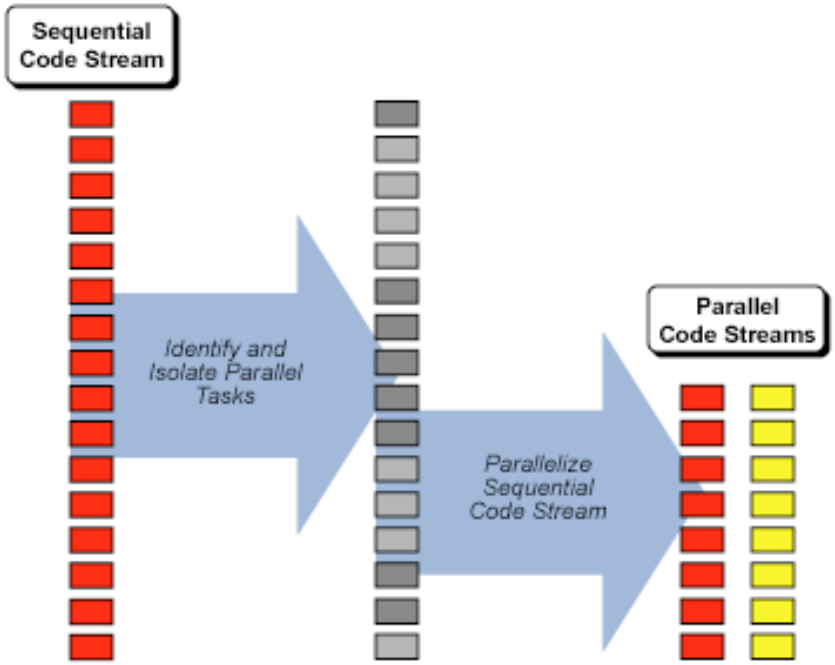
Why Multicore?



- The MHz race has run out of steam.
  - The “free ride” is now over.
  - Trade-off between raw performance and practical physical implementation and utilization.
    - Power consumption an increasingly “hot” issue … enough to nearly fry an egg.

- Result
  - Virtually impossible to buy a non-multicore processor in any computer now.

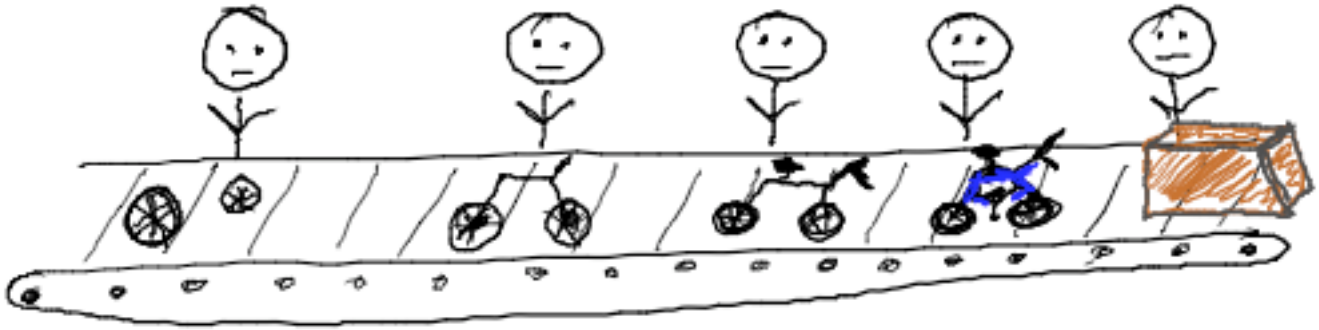
How to Improve Performance



- Increase instructions per clock cycle.
- Increase throughput or work completed per unit time.
- Lower latencies intrinsic in the system that limit the above metrics.

Tried-and-True Technique: Pipelining

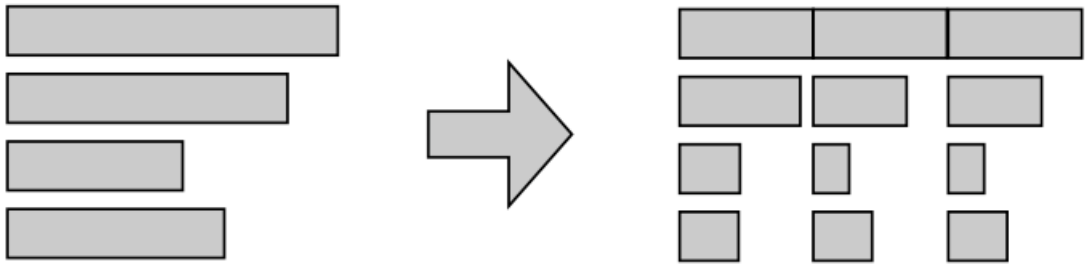
- Basic Idea
  - Breakdown complex instructions into a set of smaller steps that are executed in order like a factory assembly line.



Source: Matthew Sottile

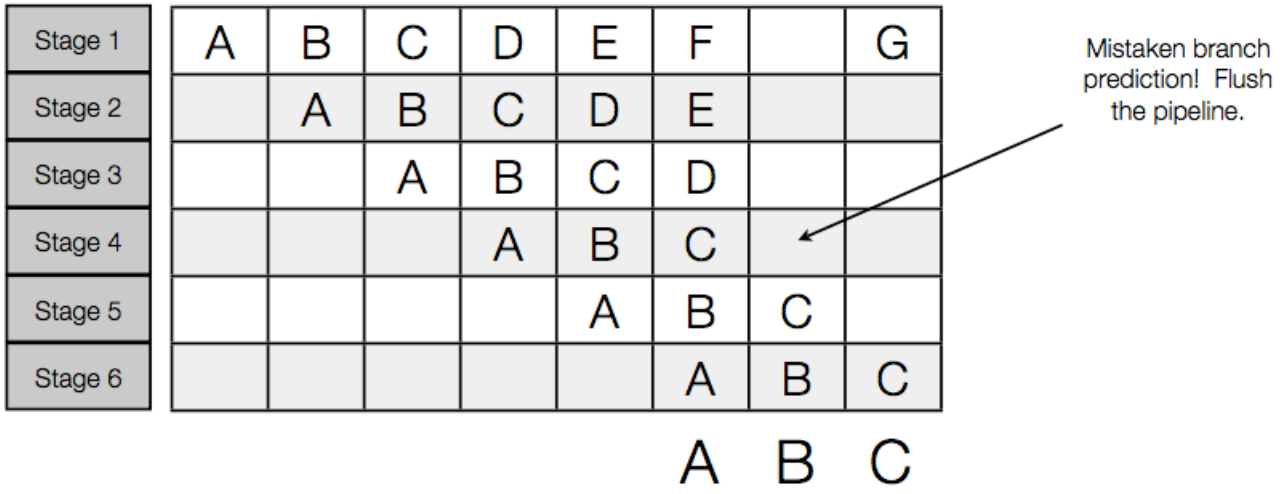
Evolution of Performance and Pipelines

- Increase # of steps, thus decreasing complexity of each step and allowing each step to complete faster.
- Fine-grained steps reduce the difference between times in each stage for instructions of differing complexity.
- If all instructions use the same pipeline, each instruction takes effectively the same amount of time to complete.



Pipelining in Practice

- A fully saturated pipeline can ideally yield one completed instruction per cycle.
- As pipelines get deeper and deeper, more important than ever to avoid bubbles or pipeline flushes that result in an increase in the average cycles per instruction.



Pipelining Requires ILP

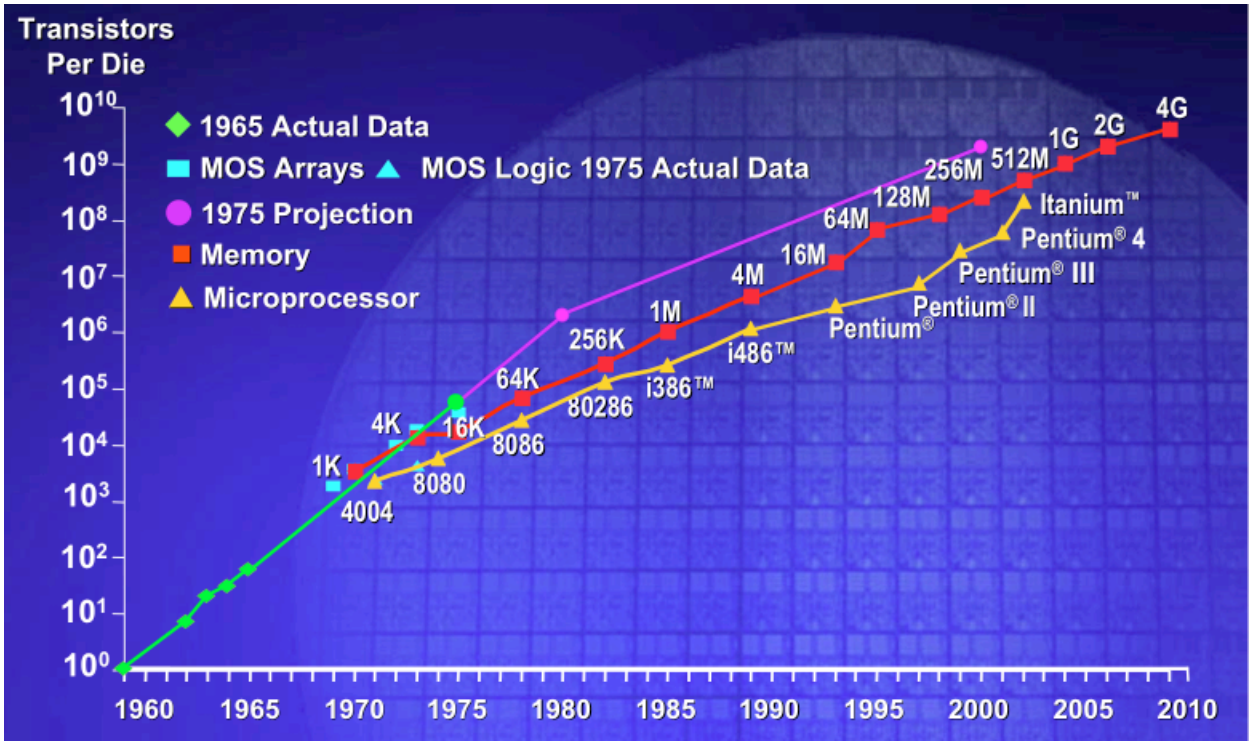
- Why does pipelining work?
- Instruction level parallelism (ILP)
  - Non-trivial portions of program code are composed of sequences of instructions that can be reordered and executed in parallel without impacting the output of the program.

```
mov ax,4
mov bx,5
add cx,ex,ex
mul dx,ex,ex
bne cx,dx,label
```

- The pipeline allows multiple instructions to be “in-flight” at any given time.

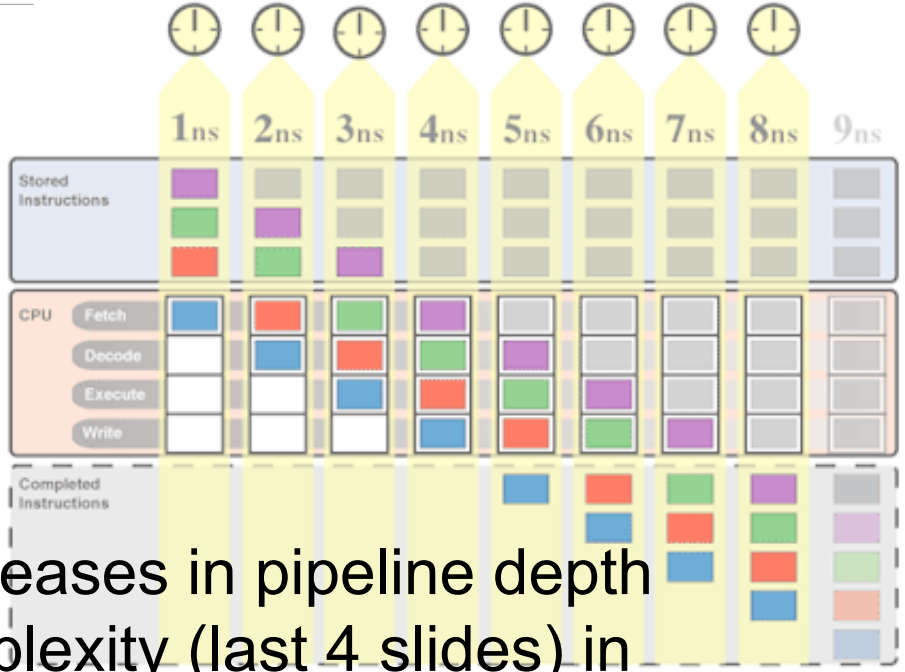
Complexity and Moores' Law

- Complexity growth in the form of Moore's Law
  - Transistor count doubles once every ~24 months.



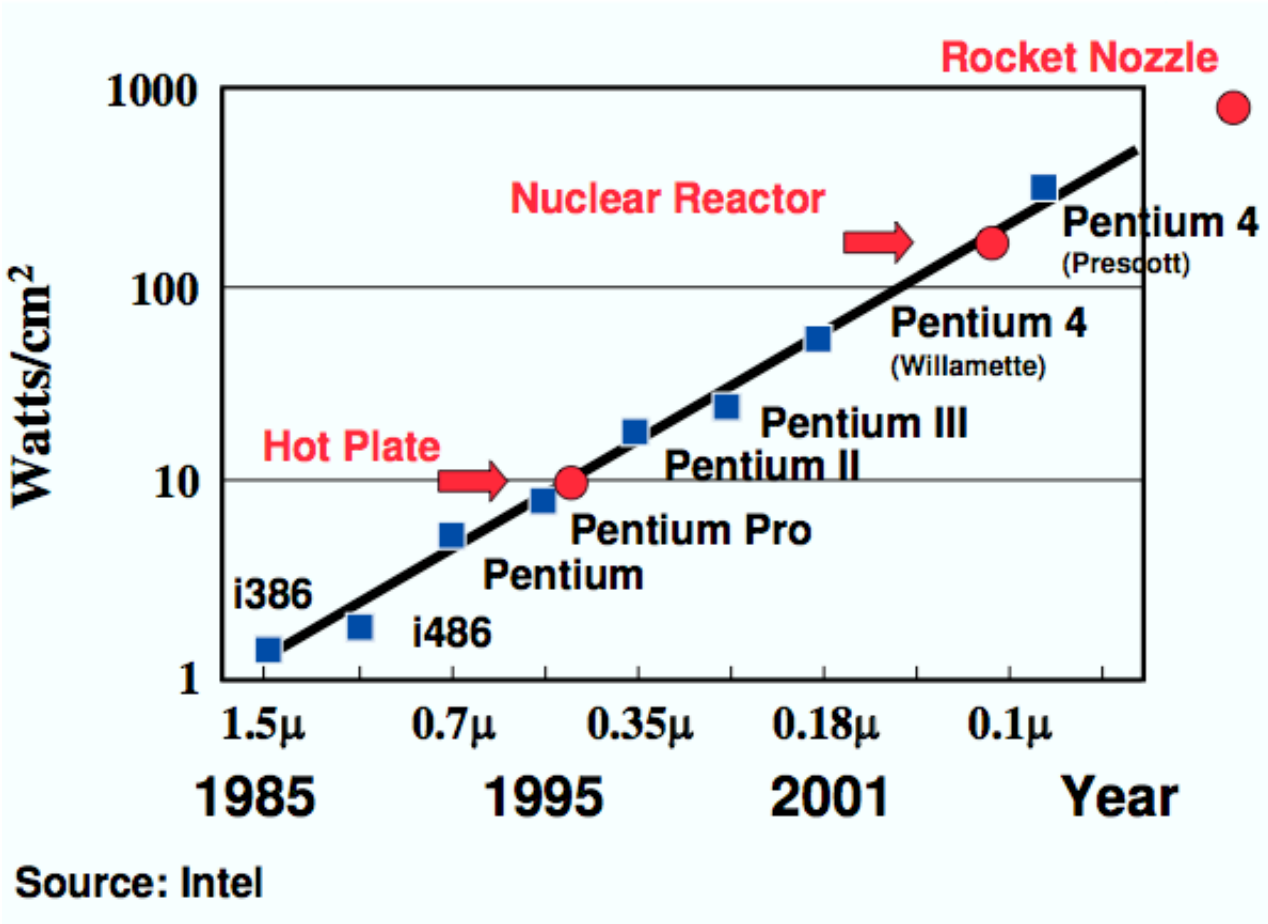
Source: Intel

The MHz Race …

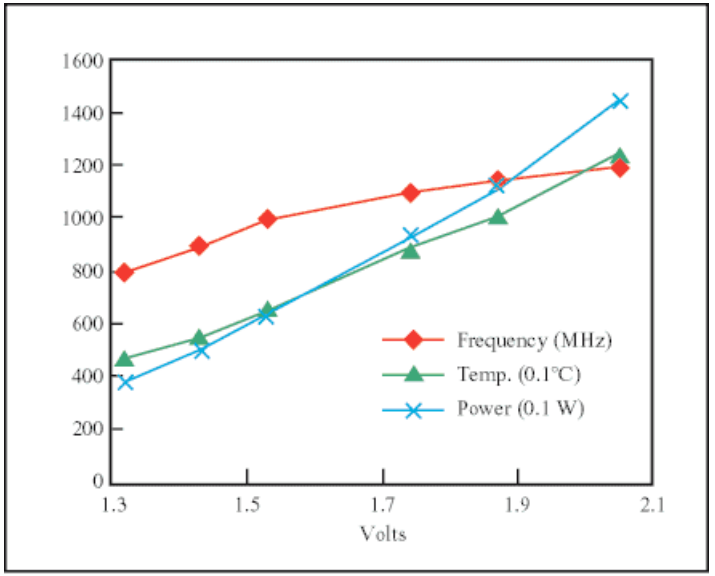


- Industry got away with increases in pipeline depth and related hardware complexity (last 4 slides) in order to ramp-up processor clock speeds.
- "Houston … we have a problem …"
  - *Power and cooling are design constraints of equal importance to performance now.*
    - Cooling: amount of cooling necessary, type of cooling, packaging.
    - Power: battery requirements, electric bills.

Power Density

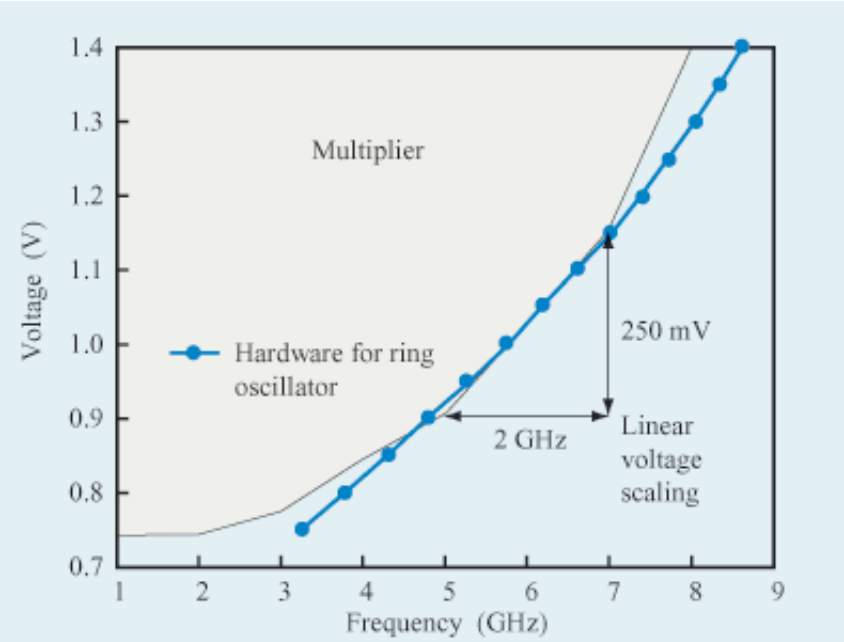


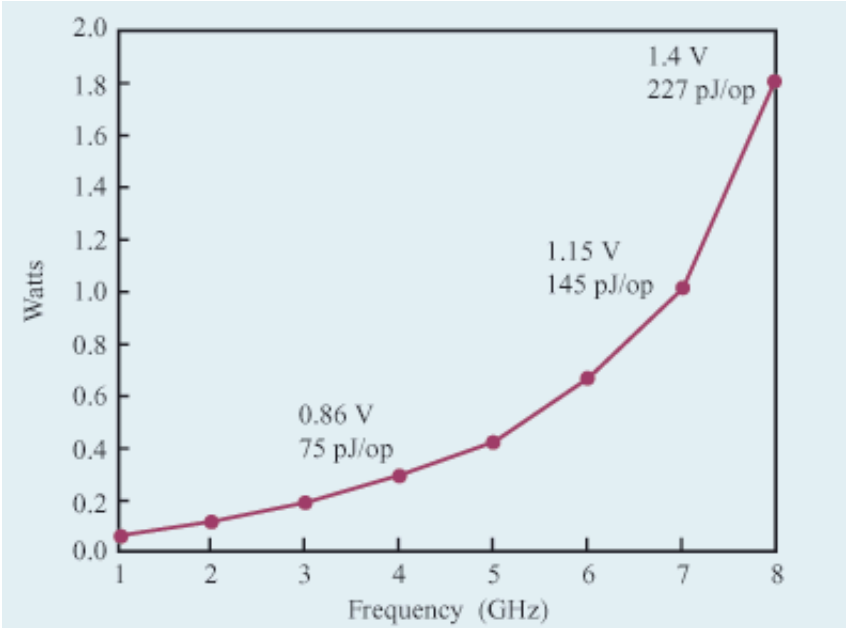
Power



- Power is really a function of …
  - Die size
    - More transistors and wires to feed
  - Frequency *f* → How often do you need to feed them
  - Voltage *V* → At what level do you need to feed them
- Pipelining: A double-edged sword
  - Better Performance (Maybe) and Higher Power Consumption
    - Increase # of small steps to realize an instruction. **More small steps = deeper pipeline = higher *f* and *V*,** *thus enabling better performance* ***but***
    - Better performance *only* if pipeline can be kept full. Difficult to do as pipelines get deeper.

Frequency, Voltage, and Power





Solutions to Power & Cooling

- Reduce component count (transistors and wires)
  - Reduce *complexity*
- Shrink components
- Manage power intelligently
  - Example: C. Hsu and W. Feng, "A Power-Aware Run-Time System for High-Performance Computing," *ACM/IEEE SC2005*, November 2005.
- Reduce leakage of transistors

Key to Performance: ILP?

- Smaller steps = deeper pipeline = more ILP = higher *f but*
  - Hardware much more complex.
  - Hardware runs hotter.
  - Deeper pipelines and better ILP identification is getting harder and more costly → diminishing returns

- Solution?
  - Multicore. A change in technology direction that forces a fundamental rethinking of how to continue boosting performance.

Multicore: Solution to Power & Performance

- Multicore Recipe
  - Stop increasing the complexity of single cores.
  - Replicate simpler cores on a die and exploit thread-level parallelism (TLP).
  - Reduce clock rates as pipelines do *not* have to be so deep, thus reducing power requirements.
- Hyperthreading NOT!
  - Many units in the CPU were shared between the “virtual cores” that hyperthreading provided, limiting performance seen in practice.
  - Hyperthreaded processors (SMT) still suffer from the complexity problem, leading to extreme power and cooling requirements.

Don't Be Misled!

- Pipelining is here to stay.
  - At a certain granularity, blocks of instructions in typical programs exhibit a good degree of ILP.
- New CPUs still improve pipelining and related single-threaded performance improvements.
  - *… but it's not the focus anymore.*

Reading Assignment

- Brian Hayes, "Computing in a Parallel Universe," *American Scientist*, November-December 2007. http://www.americanscientist.org/issues/pub/computing-in-a-parallel-universe
- Herb Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software," *Dr. Dobb's Journal*, 30(3), March 2005. http://gotw.ca/publications/concurrency-ddj.htm