The Multicore Challenge

Gregory W. Donohoe
Electrical & Computer Engineering
and
Computer Science

Aug 29, 2013
CS 451 & 551

University of Idaho
College of Engineering

Why this talk?

Article: *The Trouble with Multicore*, David Patterson, IEEE Spectrum, July 2010.

Thesis: computing speed has increased exponentially since the 1940's due to advancement in the implementation technology (vacuum tubes, transistors, integrated circuits, VLSI.) Increased computing power, decreasing cost and size have created an explosion of products that have changed life everywhere on the planet.

- This trend has stalled due to physical limits.
- Future increases in computing performance must come from parallelism.
- This fact has spawned the *Third Software Crisis.*

Will the world really be able to keep the party going through parallelism?

University of Idaho
College of Engineering

Moore's Law

"The number of transistors we can fit on a chip for a given cost will double every two years."

-- Dr. Gordon Moore, Intel Co-founder, 1965

Implications

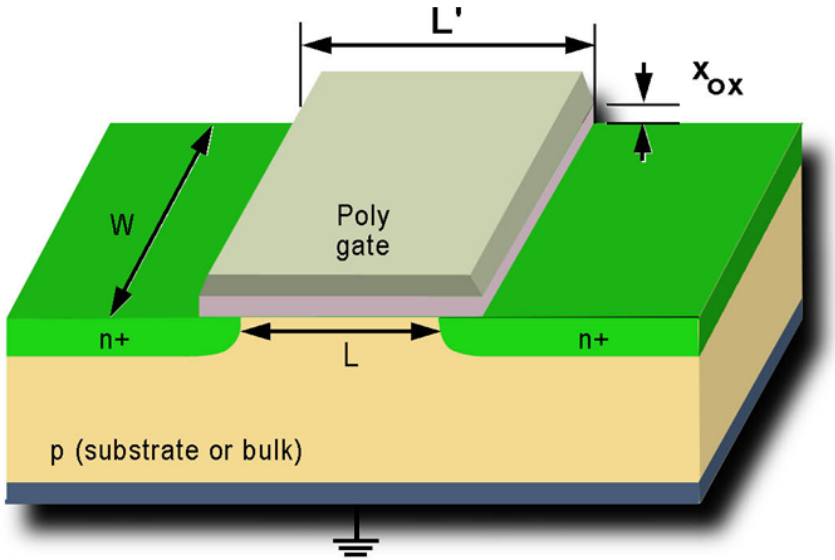
- Transistor linear dimension cut in half; area by 1/4
  - Smaller distances, signals don't have to travel as far
  - Processor speed increases
  - Amount of memory will double
- Computing products shrink in size
- Prices of computing products fall dramatically

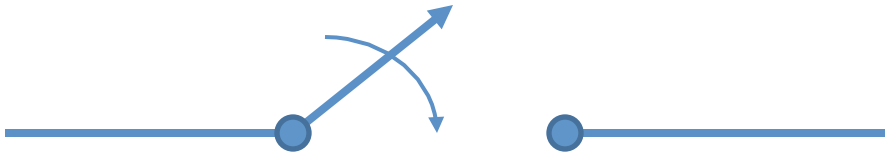
The shrinking transistor causes computing products to became faster, more capable, smaller, lighter, cheaper.

University of Idaho
College of Engineering

Metal Oxide Semiconductor Transistor

Lowest-level building block of computing systems.

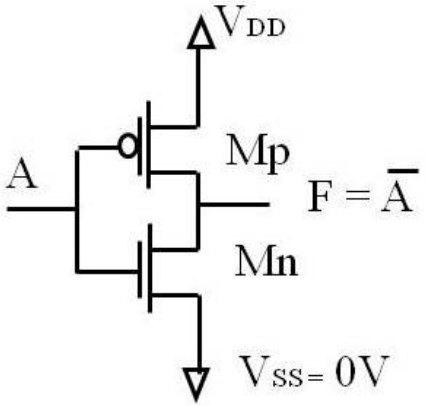




University of Idaho
College of Engineering

Gregory W. Donohoe

Complementary Metal Oxide Semiconductor (CMOS) Inverter

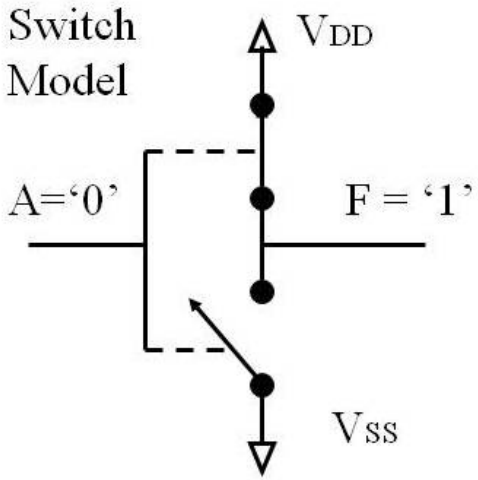


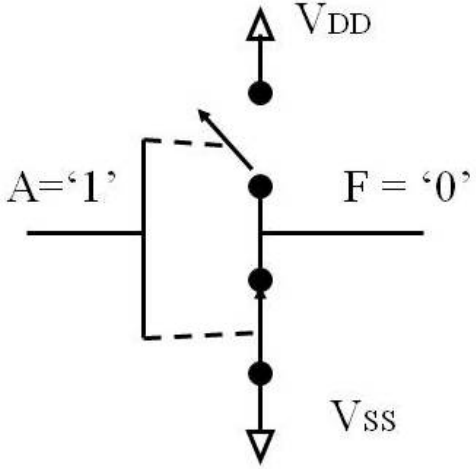
Let:

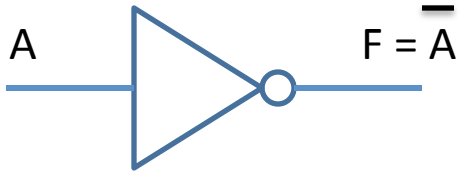
V_{DD} = logic '1'

0V = logic '0'

A	F
1	0
0	1



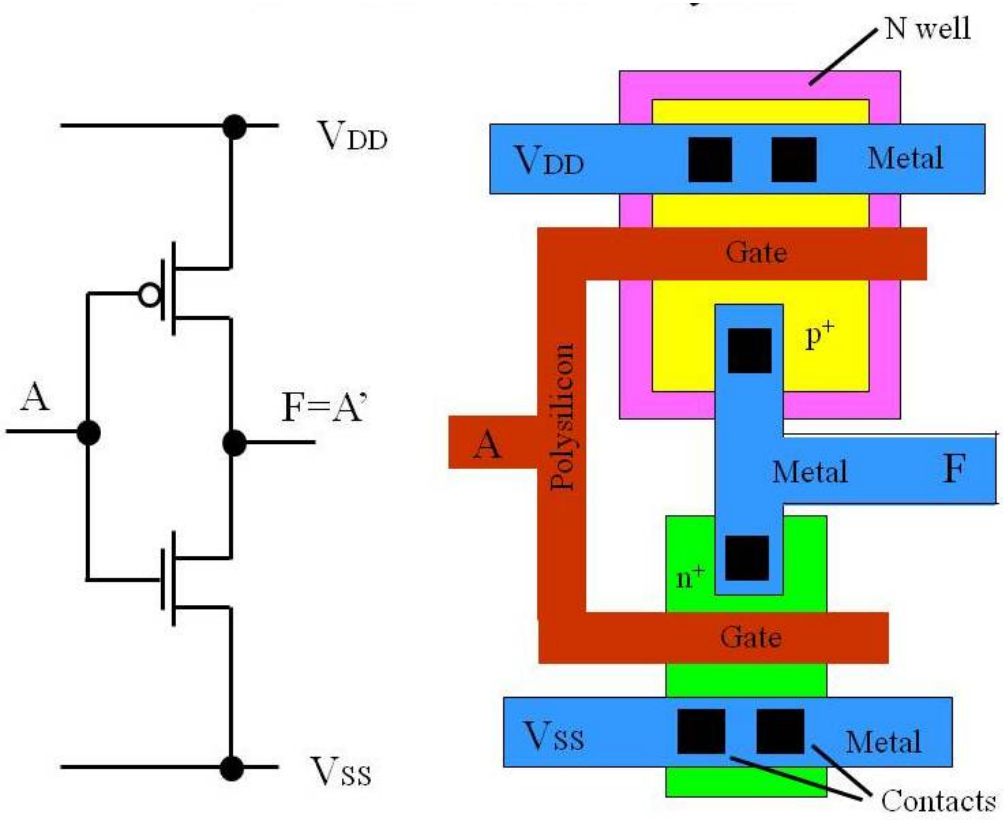




University of Idaho
College of Engineering

Gregory W. Donohoe

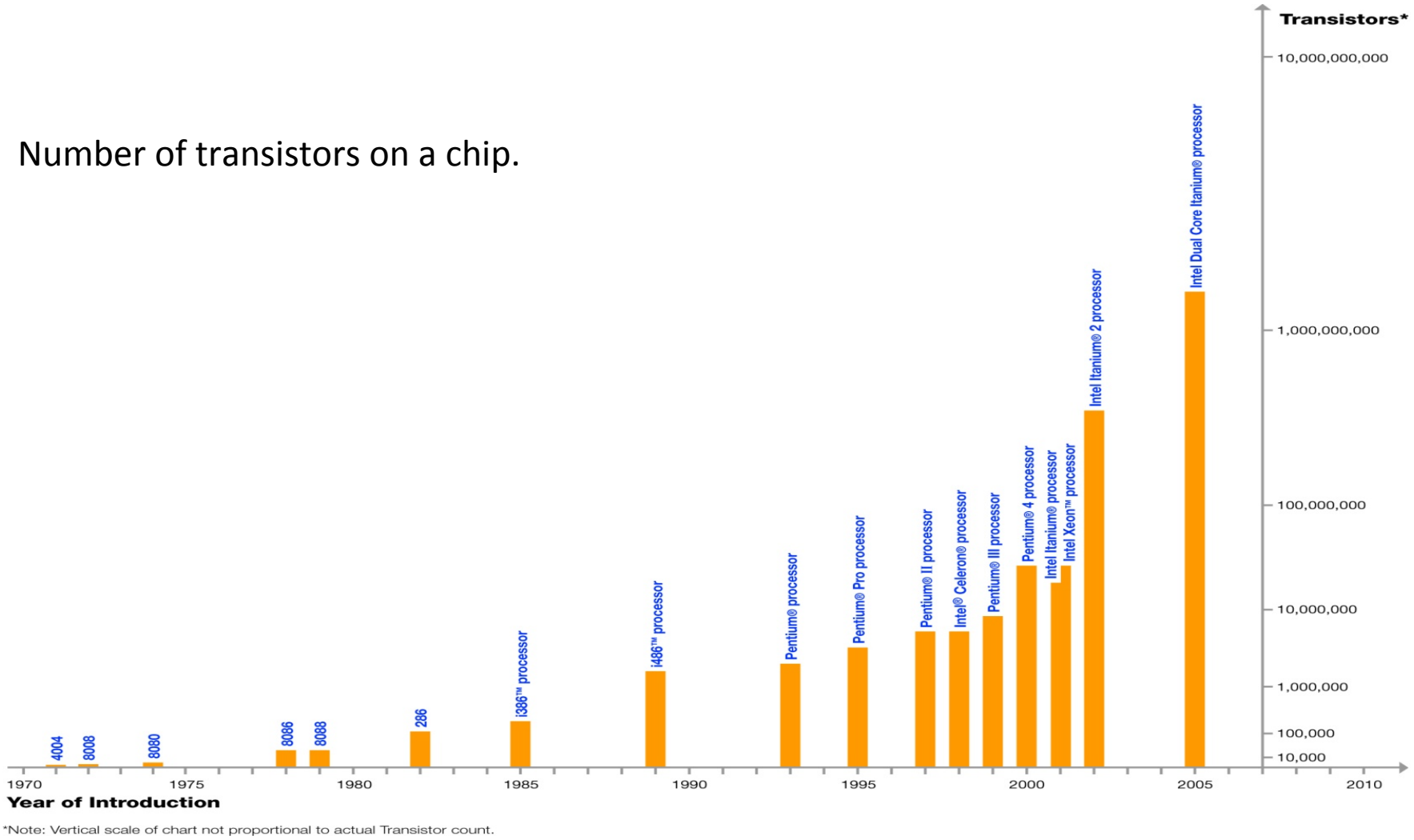
CMOS Inverter Layout



University of Idaho
College of Engineering

Moore's Law

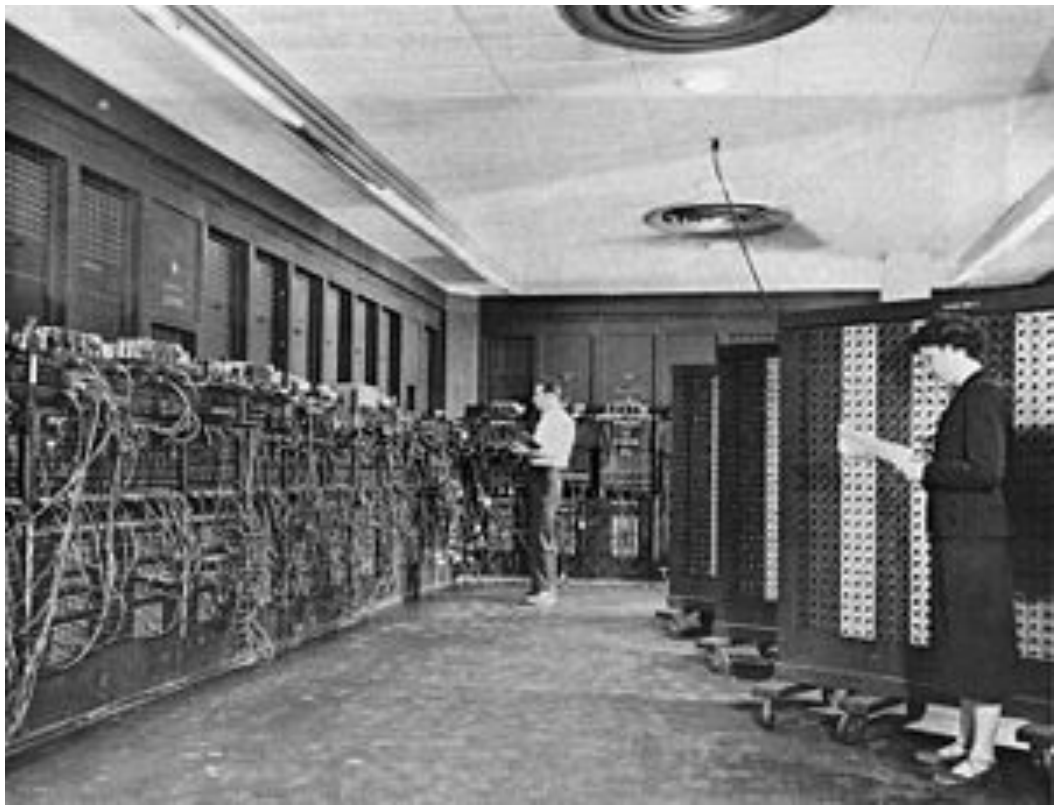
Number of transistors on a chip.



University of Idaho
College of Engineering

Gregory W. Donohoe

Eniac Computer, 1946



- 17,468 vacuum tubes.
- 70,000 resistors.
- 6,000 manual switches.
- Weight: 30 tons.
- Area: 63 m^2.
- Power Consumption: 160 kW.
- 8 Kbits of data memory.
- 5 KHz cycle rate.
- 2 msec cycle time.
- Cost: $500,000.

University of Idaho
College of Engineering

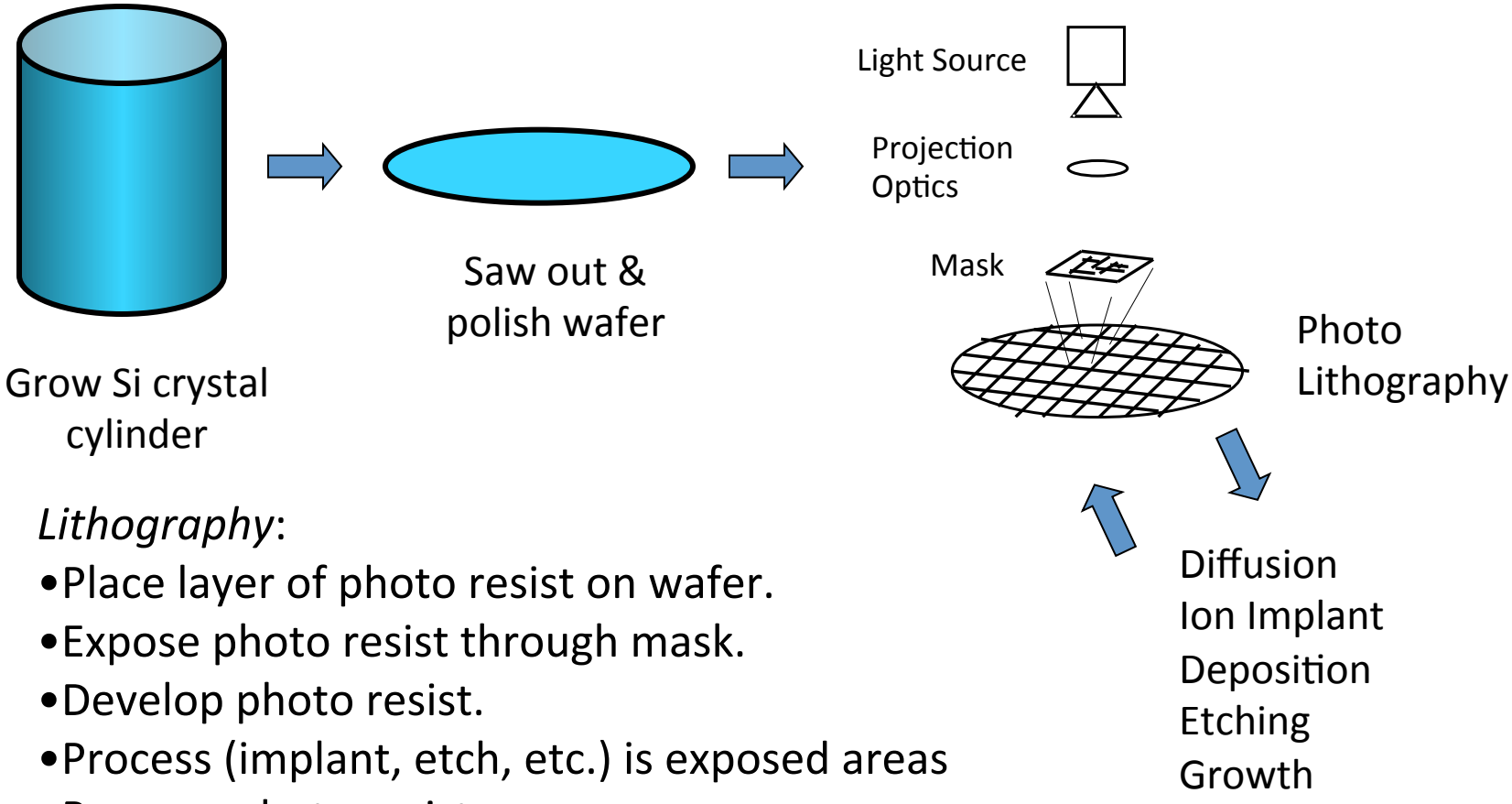
Apple iPad 4, 2012

- 64 Gbytes of memory.
- 1.4 GHz cycle rate, dual core
- .7 nsec cycle time.
- Power consumption ~ 0.25 W
- Weight: 1.44 pounds (652 g)
- Area: 70 in^2 (45,000 mm^2)
- Cost: $700.
- 64 million times more memory than ENIAC.
- 280,000 times faster than ENIAC
- + WiFi, Bluetooth, video camera, GPS, …

University of Idaho
College of Engineering

Gregory W. Donohoe

Integrated Circuit Fabrication



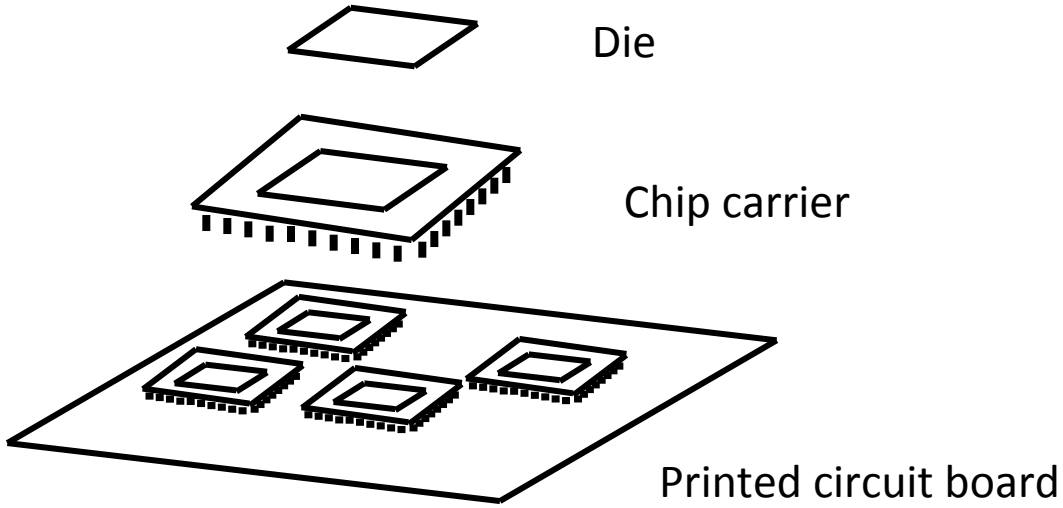
Lithography:

- Place layer of photo resist on wafer.
- Expose photo resist through mask.
- Develop photo resist.
- Process (implant, etch, etc.) is exposed areas
- Remove photo resist.

University of Idaho
College of Engineering

Chips to Systems

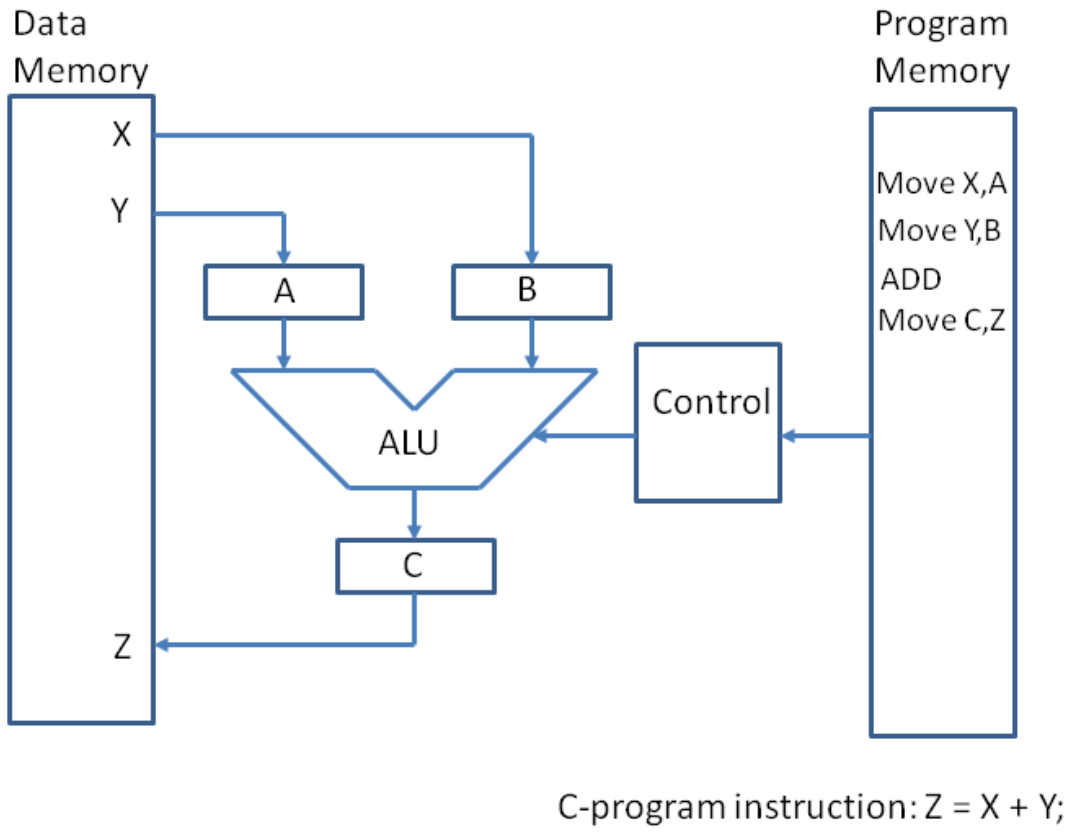
- Die – monolithic silicon substrate
- Chip carrier – substrate to which die is glued, I/O pads wirebonded to package pins
- Printed Circuit Board – chip packages surface mounted or through-hole mounted, interconnected with etched copper "lands"
- System – backplane or system-area-network



Gregory W. Donohoe

University of Idaho
College of Engineering

Sequential Stored Program Computer Architecture



Von Neumann and Harvard models

- Fetch and execute instructions sequentially.

University of Idaho
College of Engineering

Ubiquity of the Stored Program Computer

Generality: Church-Turing thesis

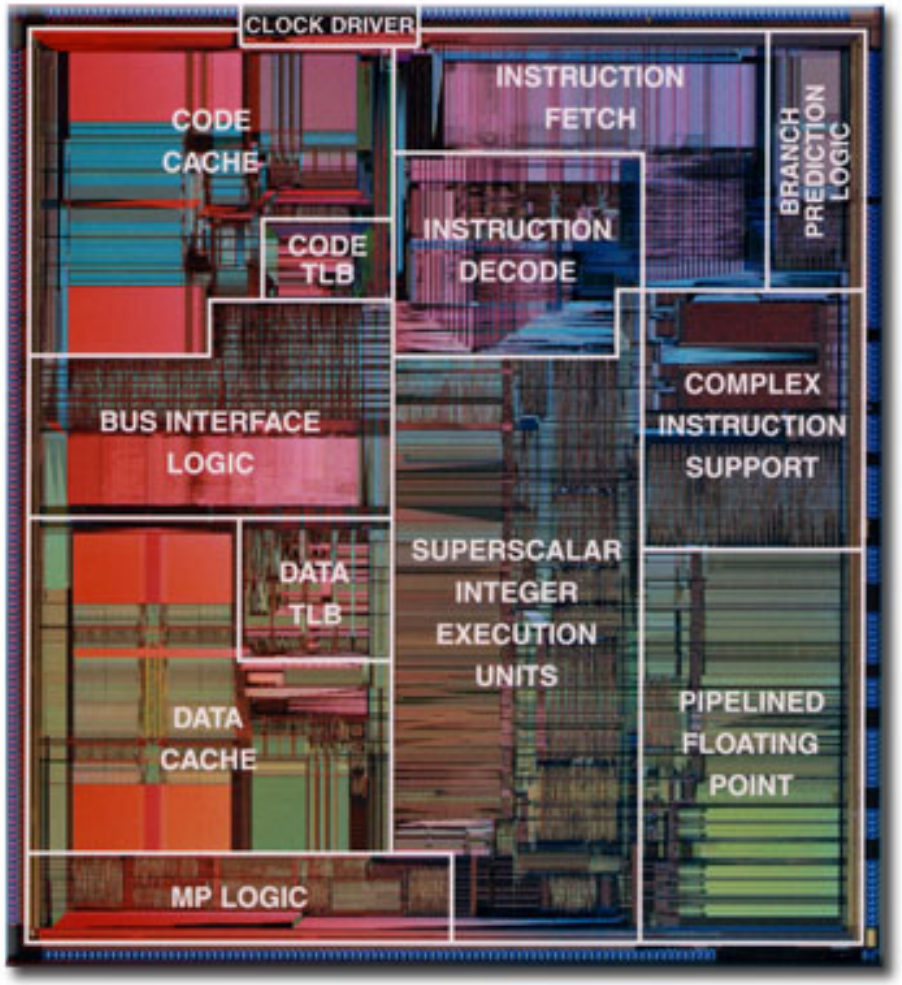
- Performs all the actions we call computing, one step at a time
- Bounded only by time and memory
- Spreadsheets, Internet search engines, aircraft landing systems…

Design Reuse

- Vast repertoire of program specification in source code libraries
- Huge programmer knowledge base
- Development tools

Intel Pentium Processing Unit

A High Performance "Uniprocessor"

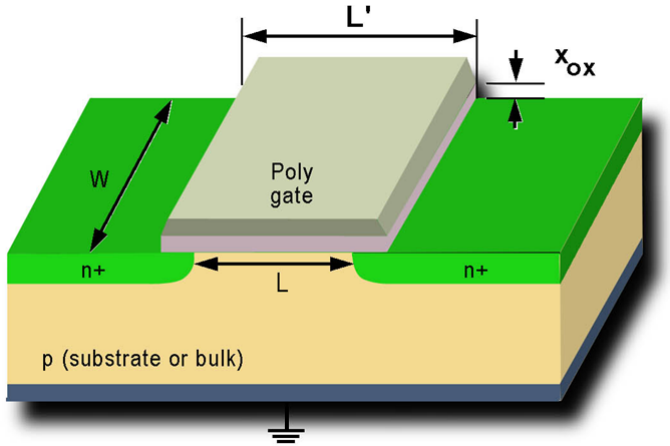


Software environment:
C, C++, Java, 4GLs,…

Operating system support
- CPU Scheduling
- Memory Management
- Input/Output
- Multithreading

University of Idaho
College of Engineering

The Shrinking Transistor – The End of Moore's Law?



1980's: L = 100 micrometers
2000's: L < 100 nanometers
2010's: L < 10 nanometers

Reaching the limits of lithography?

Features < than wavelength of light

Reliability issues

- Gate oxide only a few atoms thick
- Internal field intensities very strong
- Power, heat density too high

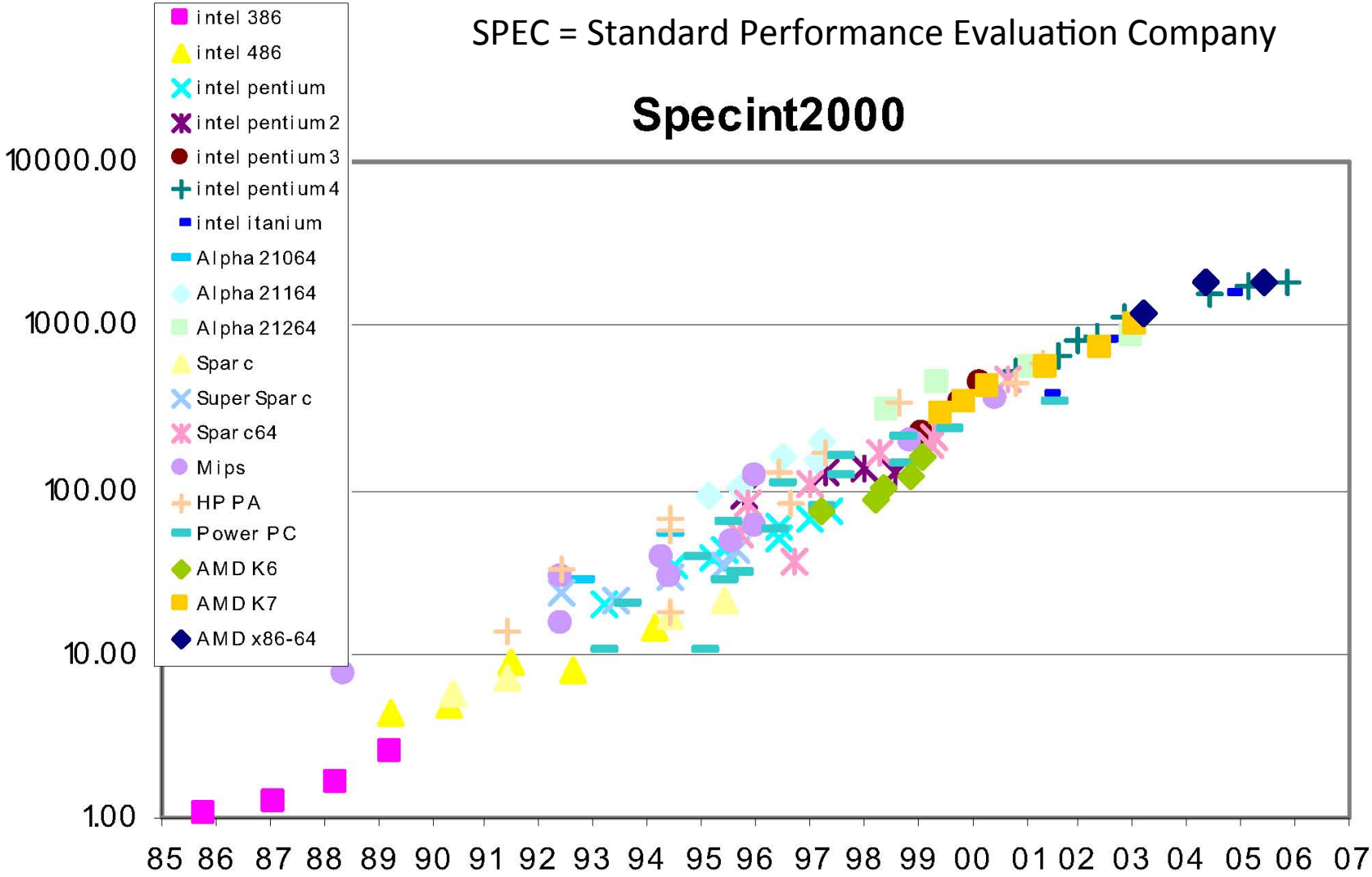
Reaching the limits of frequency scaling…

- We can make transistors smaller, but we can't make 'em faster.
- Memory & interconnect latencies not keeping up with gates

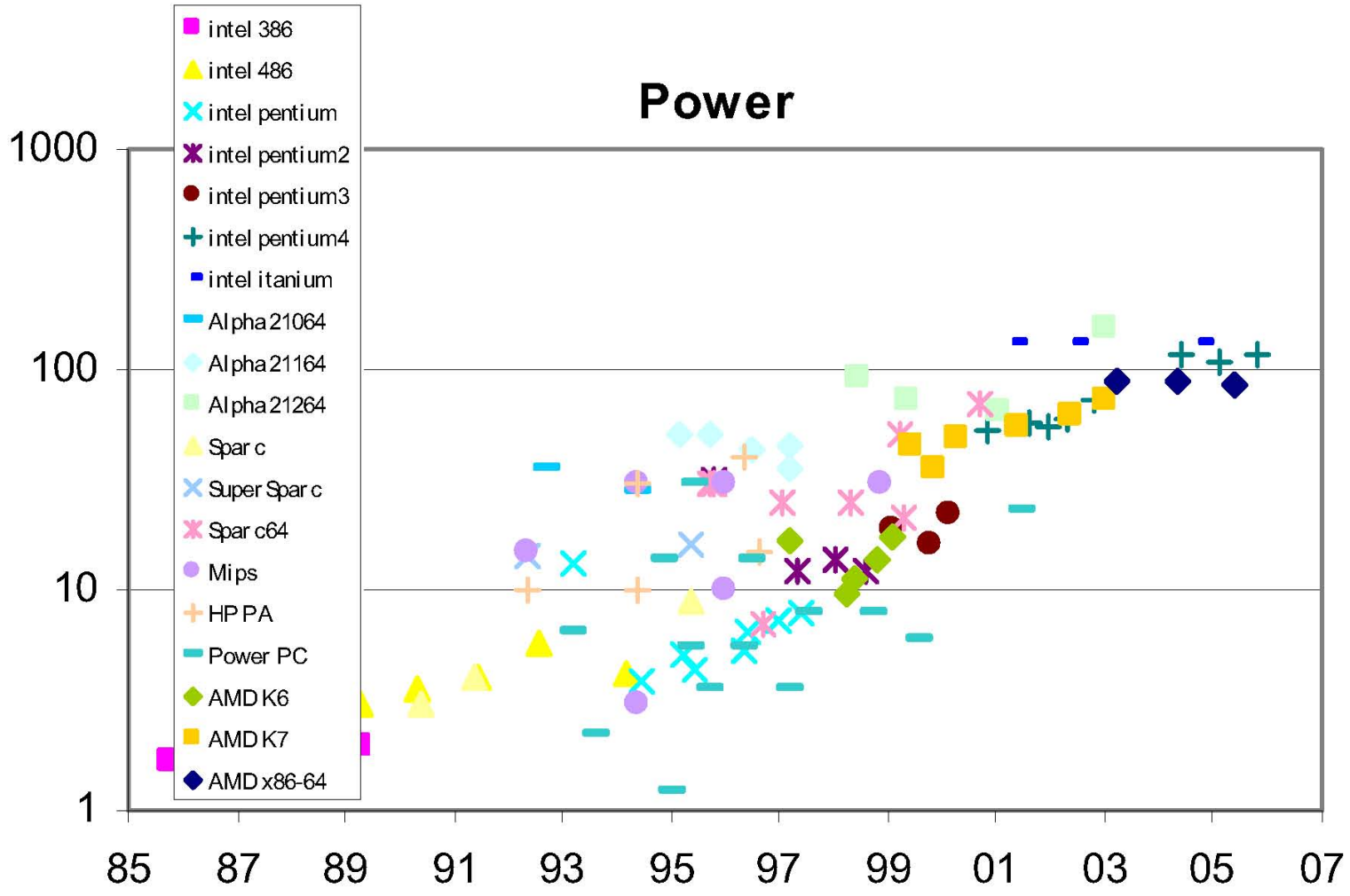
University of Idaho
College of Engineering

The March to Multicore: Uniprocessor Performance (SPECint)

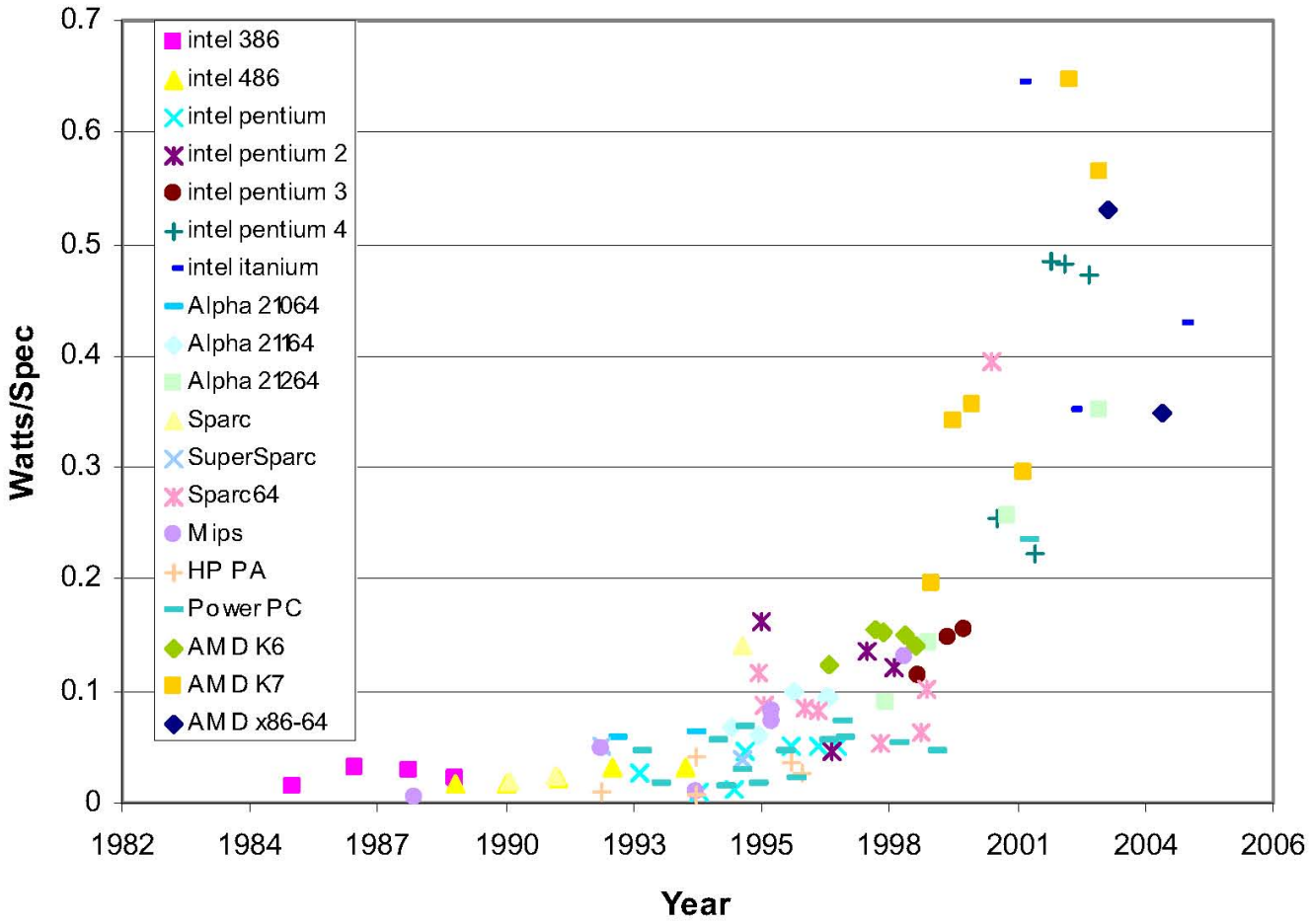
SPEC = Standard Performance Evaluation Company



Power Consumption (watts)



In Power Efficiency (watts/spec)



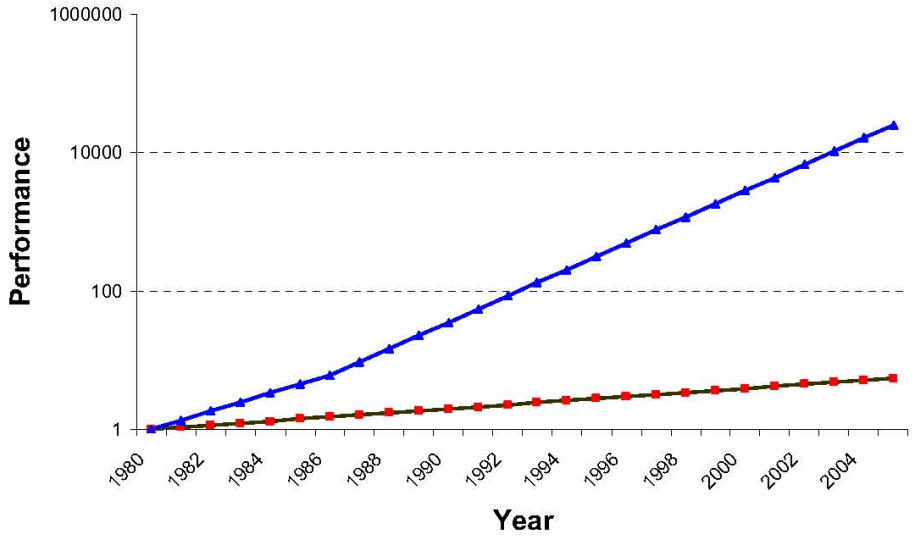
DRAM Access Latency

- Access times are a speed of light issue
- Memory technology is also changing
  - SRAM are getting harder to scale
  - DRAM is no longer cheapest cost/bit
- Power efficiency is an issue here as well

Images removed due to copyright restrictions.

µProc
60%/yr.
(2X/1.5yr)

DRAM
9%/yr.
(2X/10 yrs)



First Casualty: Frequency Scaling

We can no longer make processors faster by frequency scaling

- But we can still make transistors smaller
- Put more stuff on a chip
- Look for speed through parallelism

Some approaches

- **Design a custom processor for the task at hand**
  - Takes years, costs millions of dollars
  - Not flexible – specs change, start over
- **Programmable system-on-a chip (Xiliinx, Altera)**
  - CPU's – programmable in C
  - Field Programmable Gate Array configurable logic fabric
    - Programmed with a hardware description language
  - Custom structures like chains of Digital Signal Processor blocks
    - Mulitiply, add, shift, scale in a pipeline

This class – adding more parallelism to sequential computers

Exploiting Parallelism

Basic Forms of Parallelism

1. Data-Level Parallelism (DLP). Many *data items* processed at the same time.
2. Task-Level Parallelism (TLP). Can perform independent *tasks* at the same time.

Parallelism in Processor Architecture

- Instruction Level Parallelism. Instruction pipelining. 10X increase.
- Graphics Processing Units (GPUs), Vector Processors.
  - A single instruction applied to a collection of data, simultaneously.
- Thread-level Parallelism.
  - Data or task level, in parallel execution threads on multiple processors.
- Request-level Parallelism.
  - Like thread level, but tasks scheduled asynchronously, as in a request-response or client-sever system.

Source: *Computer Architecture: A Quantitative Approach*, 5th Ed., John Hennessey & David Patterson, 2012.

Instruction Pipelining for a RISC Processor

Instruction Execution Steps

IF – Instruction Fetch

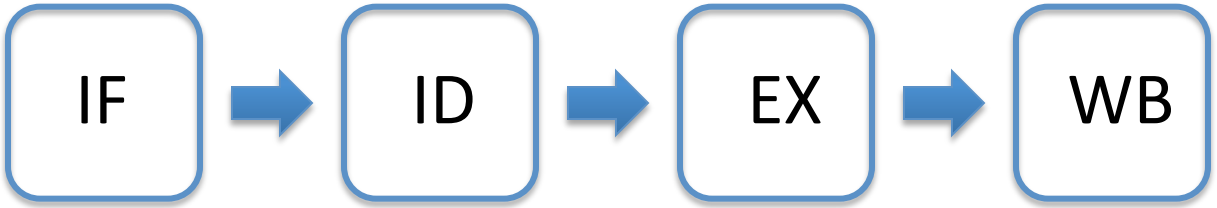
ID – Instruction Decode

EX – Instruction execute

- Memory reference: MEM
- Register , Register, ALU
- Register, Immediate, ALU
- Branch

WB – Write results back to register

Build a pipeline



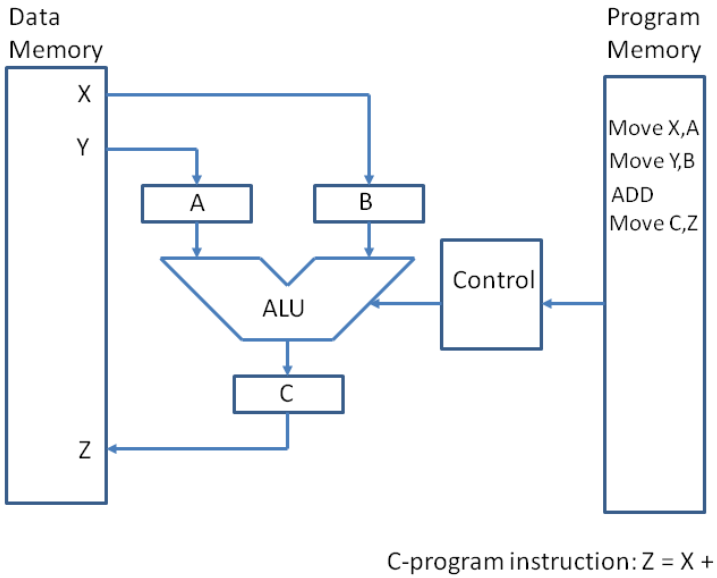
Can also build parallel execution units

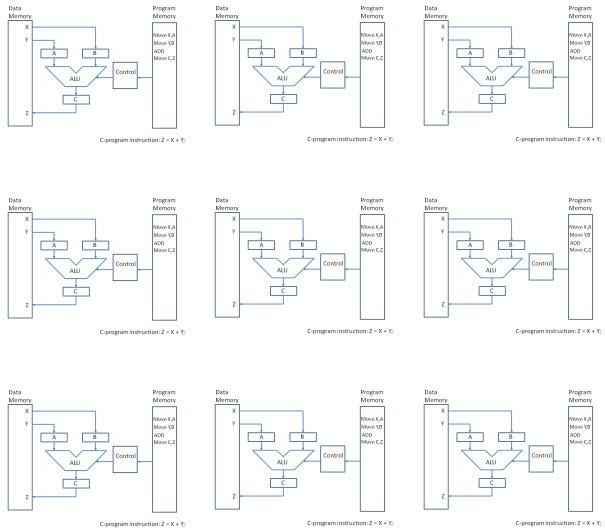
Run multiple pipelines in parallel

University of Idaho
College of Engineering

Multiprocessing

Multiple von Neumann or similar execution unit or “cores”



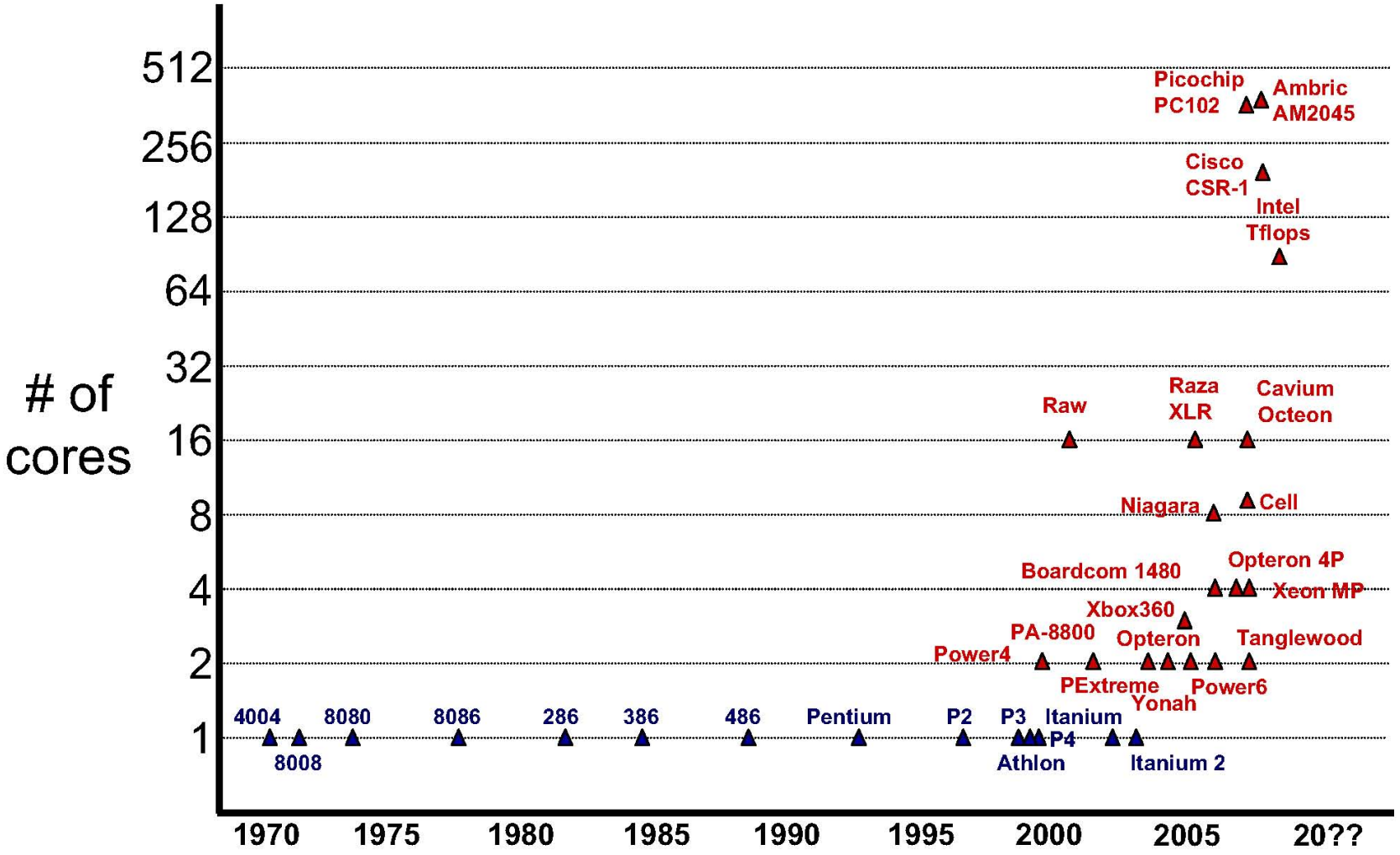


- Multiple cores on single chip: multicore
- Network or cluster of computers: distributed computing

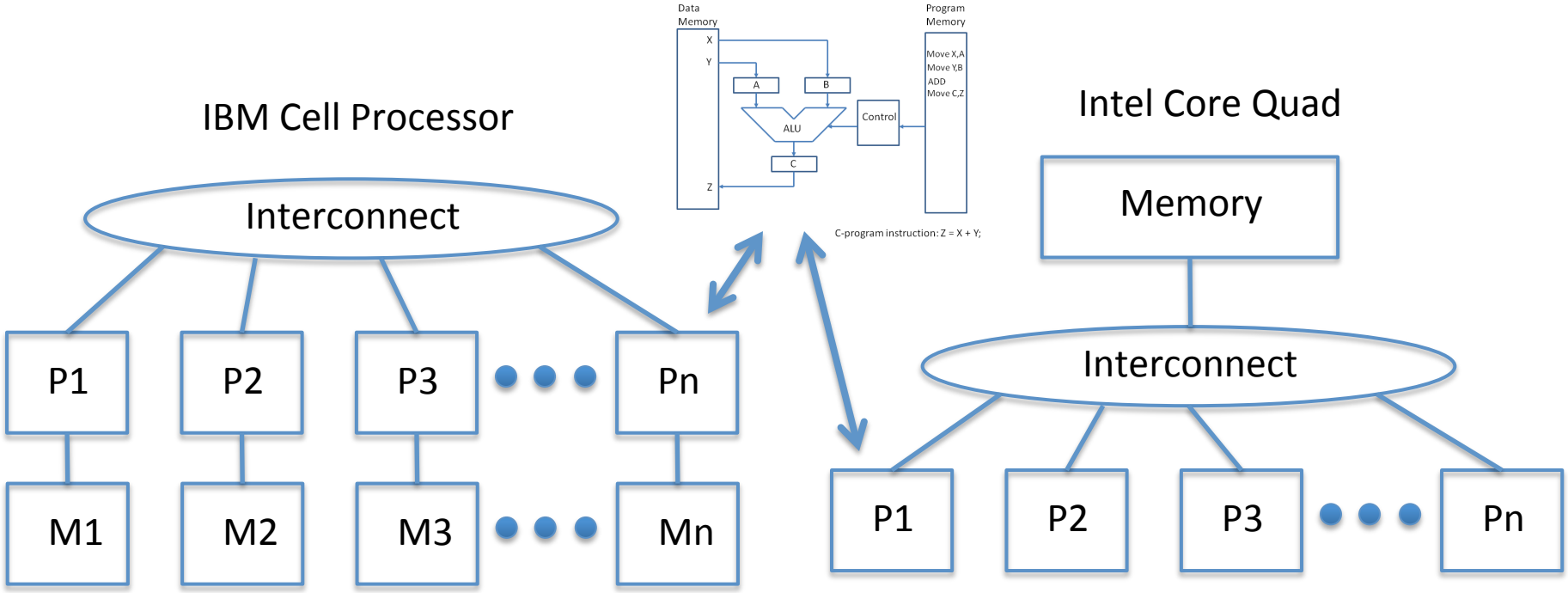
University of Idaho
College of Engineering

Gregory W. Donohoe

Multicores are Here



Basic Multicore Architectures



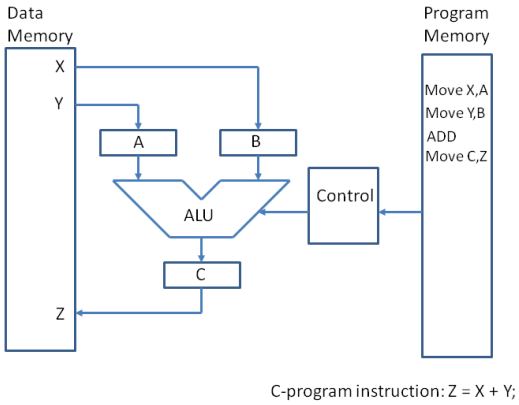
Local Memory
Each processor has its own memory
•Need to communicate explicitly between cores (e.g. message passing)

Shared Memory
Many processors share memory
•Need atomization, locking, synchronizaton

University of Idaho
College of Engineering

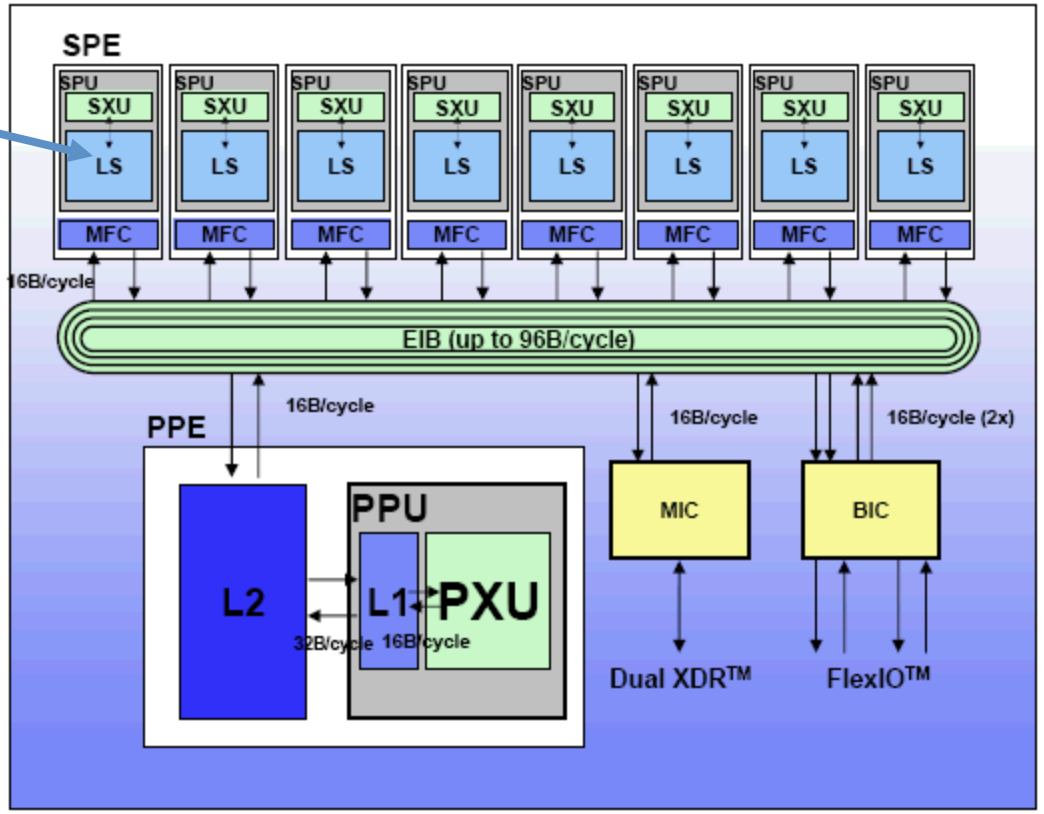
Gregory W. Donohoe

IBM Cell Processor



C-program instruction: Z = X + Y;

- Local memory for each processor
- Network interconnect bus

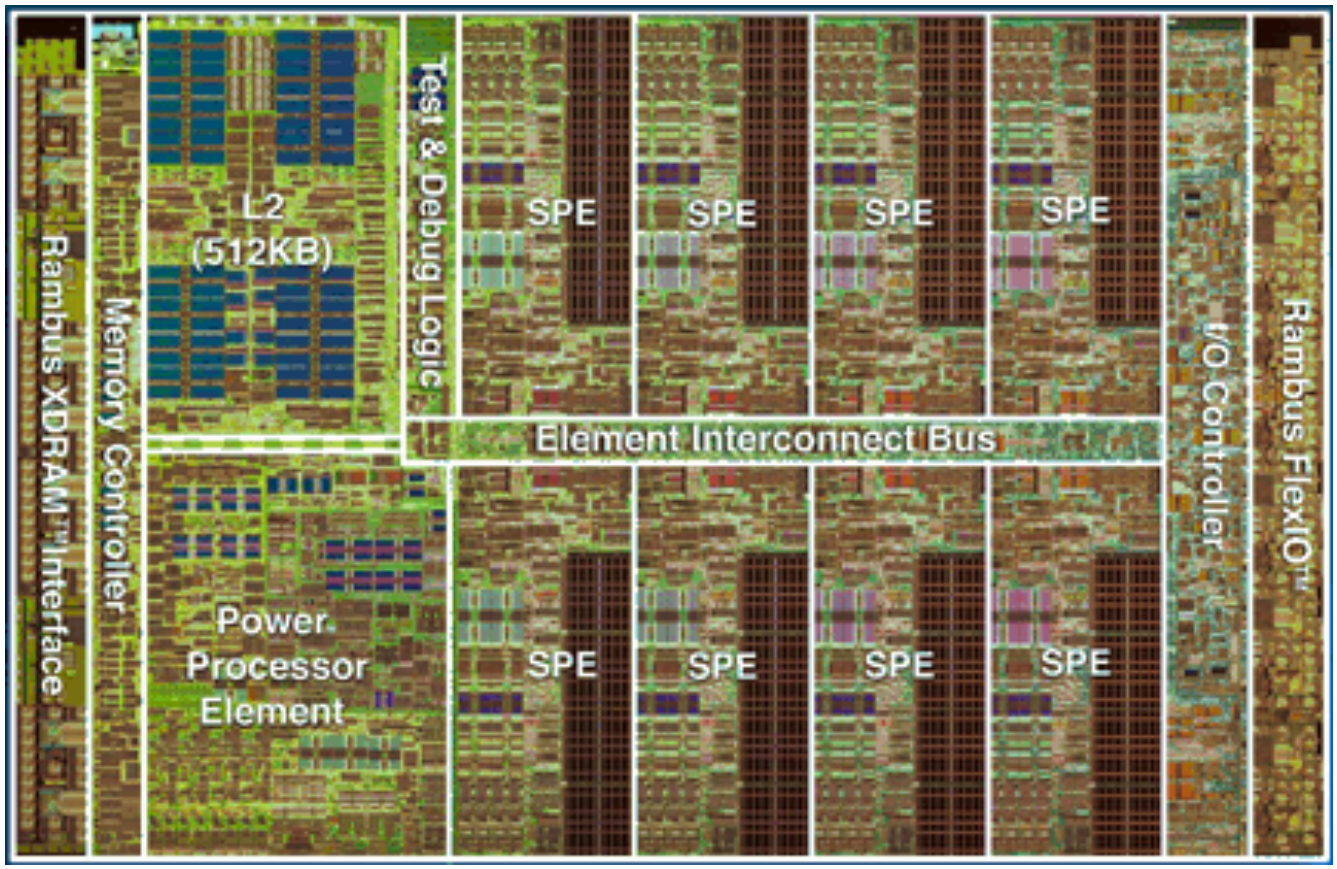


Source: M. Gschwind et al., Hot Chips-17, August 2005

Used in the Sony PlayStation

University of Idaho
College of Engineering

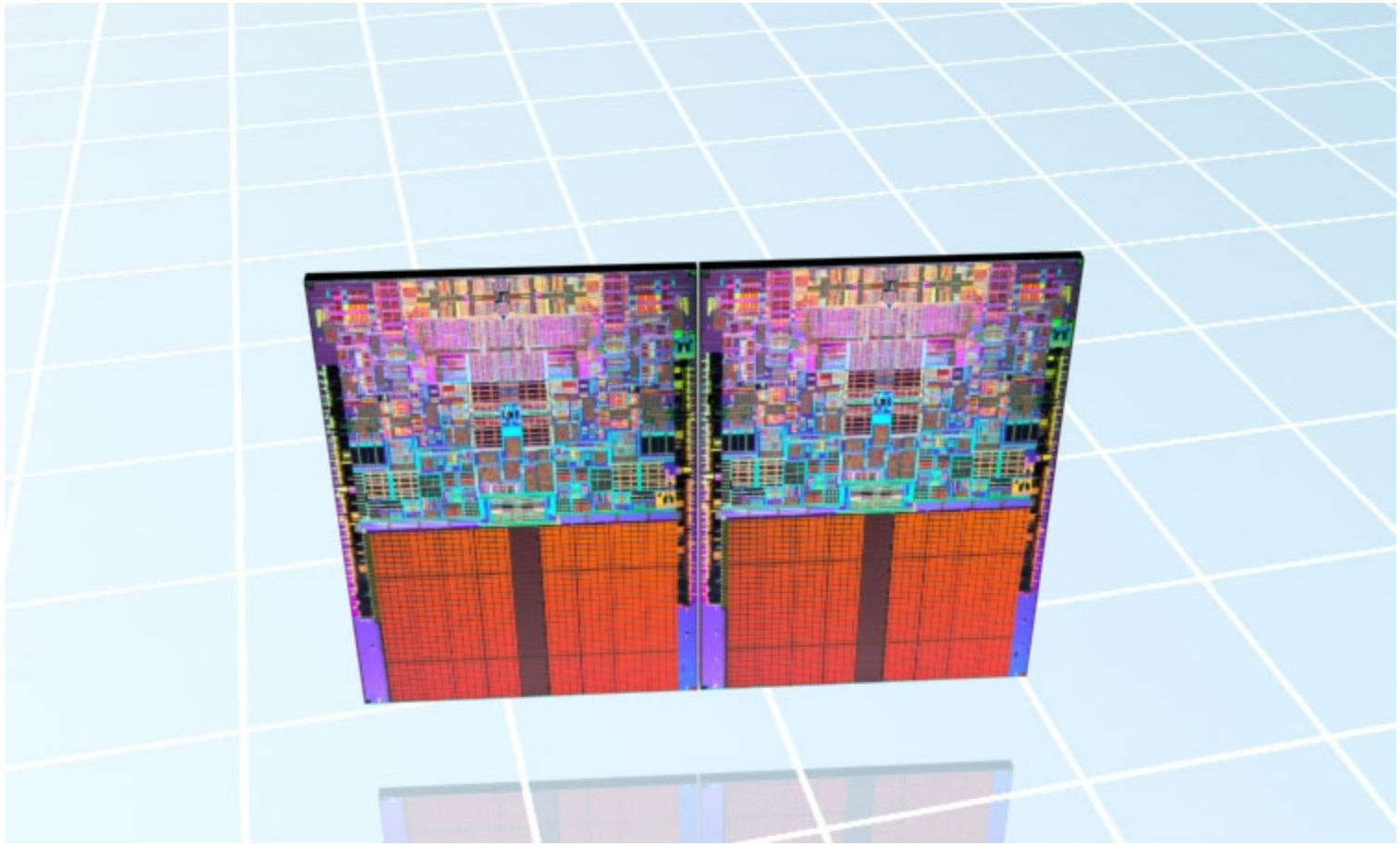
Cell Processor Chip



University of Idaho
College of Engineering

Gregory W. Donohoe

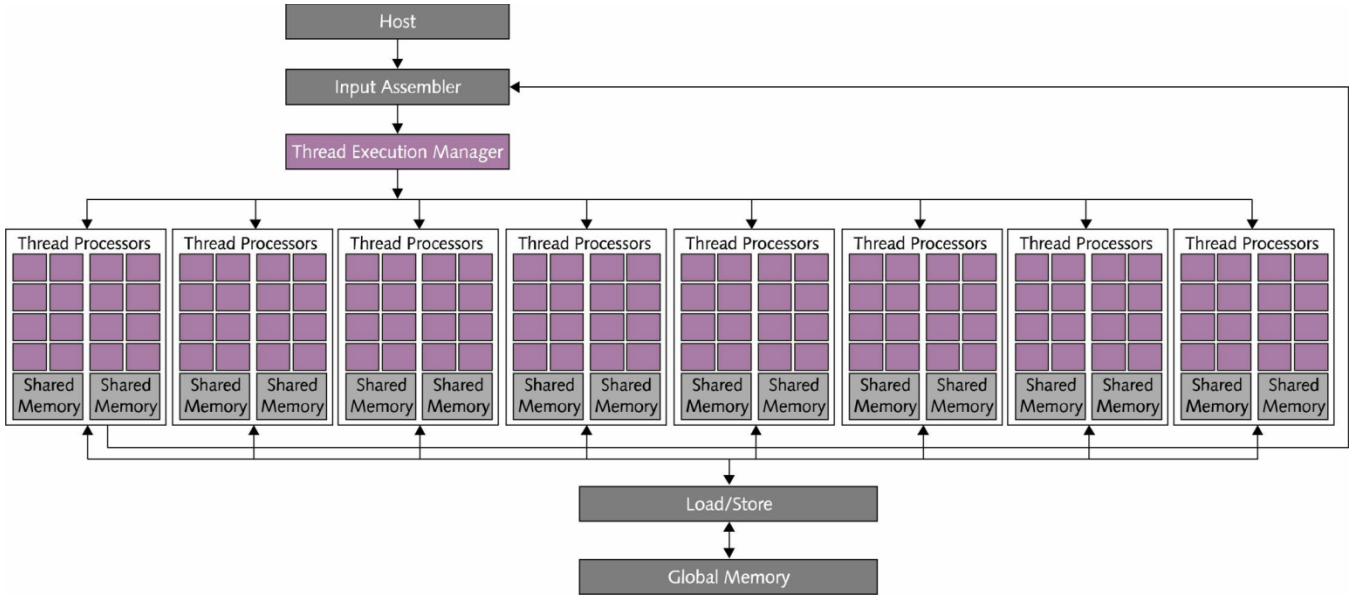
Intel Quad Core



General Purpose Graphics Processing Units (GPGPUs)

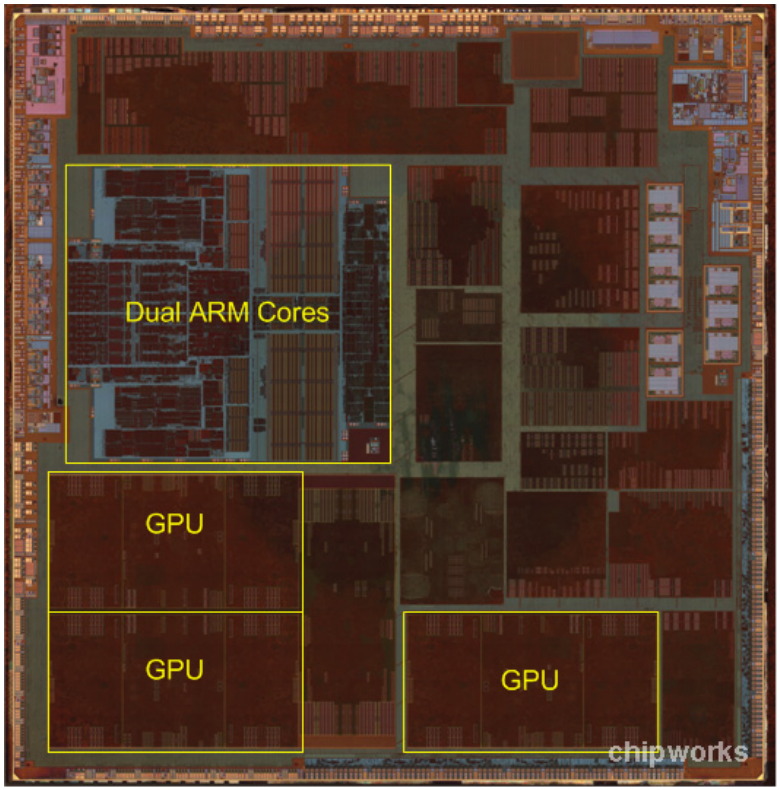
Massively Parallel Many-Core Architectures

- Derived from Graphics Processing Units
- Vector processors under control of GPU
  - Optimized for very-high-speed computations of certain kinds
  - Streaming, SIMD problems associated with graphics, DSP



Nvidia GeForce 8 CPU architecture

Apple A6 Processor



- 32 nm CMOS technology
  - 320x hydrogen atoms
- 2 ARMv7 based cores
- 3 PowerVR SGX 543PM3 GPUs
- ~1.3 GHz clock speed
- 1 GB RAM
- Programming challenge:
  - "Heterogeneous" architecture
  - OpenCL designed for this kind of problem

Intel Many Integrated Core (MIC) Architecture

Experimental code name: Knight's Corner

- Commercial Release: Xeon Phi (2012)
  - Aimed at the high performance computer market
  - 1,000 GigaFLOPS (floating point operations per second)
  - 5 billion, 22 nm CMOS "3D transistors" (200 H atoms)
  - 62 64-bit x86 CPU cores + 512-bit SIMD data unit

Programming Environments

- OpenCL - Threads
- OpenMP – Message Passing
- Intel Cilk™

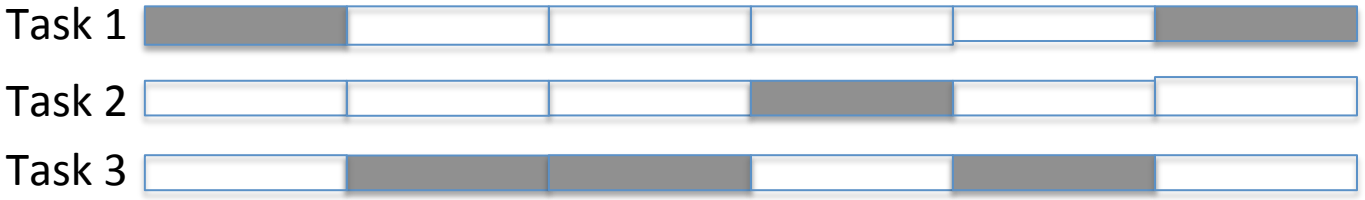
University of Idaho
College of Engineering

Parallelism vs Concurrency

Concurrency

- Separate tasks operate independently
- Communicate via well-defined channels and protocols
- Scheduling is non-deterministic: can't rely on specific sequence

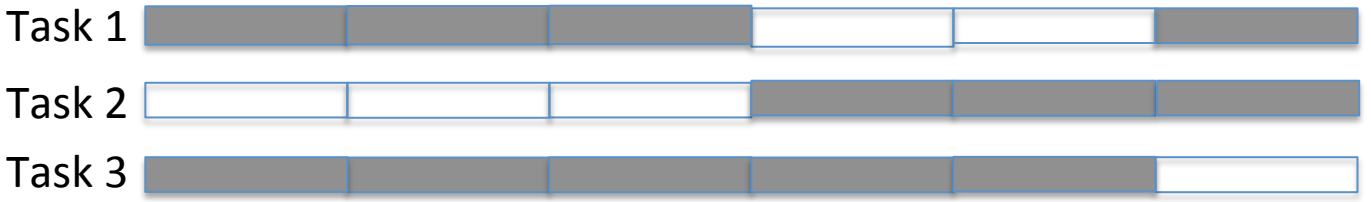
Example: 3 tasks on a unicore system.



Parallelism

- Separate tasks operate independently, communicate via well-defined channels
- Run simultaneously on separate processors
- Example: threads on a multiprocessor system
- Speedup? That depends!

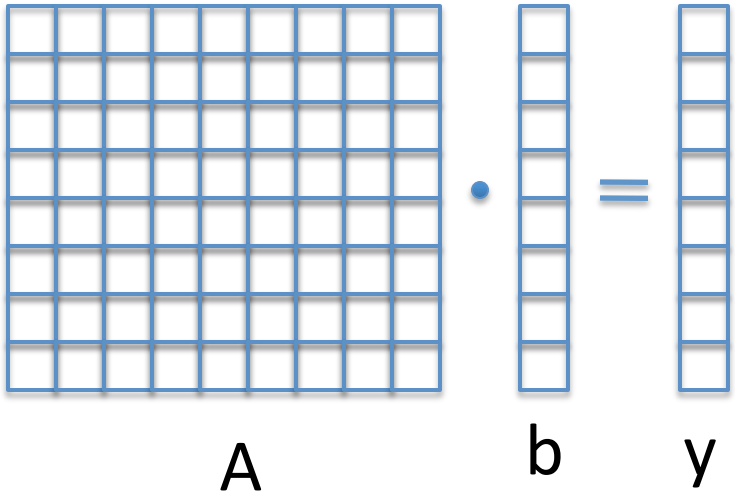
Example: 3 tasks on a dual core system.



Data Parallel Example: Vector Machine

$$A \cdot \bar{b} = \bar{y}$$

$m \times n$ Matrix-Vector Multiplication



Thread 1: $[a_{11}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_1$

Thread 2: $[a_{21}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_2$

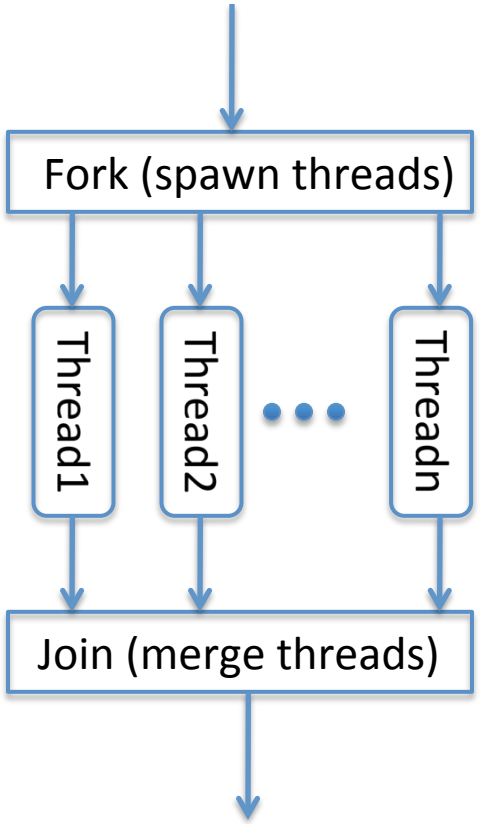
Thread 3: $[a_{31}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_3$

Thread n: $[a_{n1}\text{-}a_{nn}][b_1\text{-}b_n]^T = y_n$

We can assign one *process thread* to compute each element of y.

Each process needs:

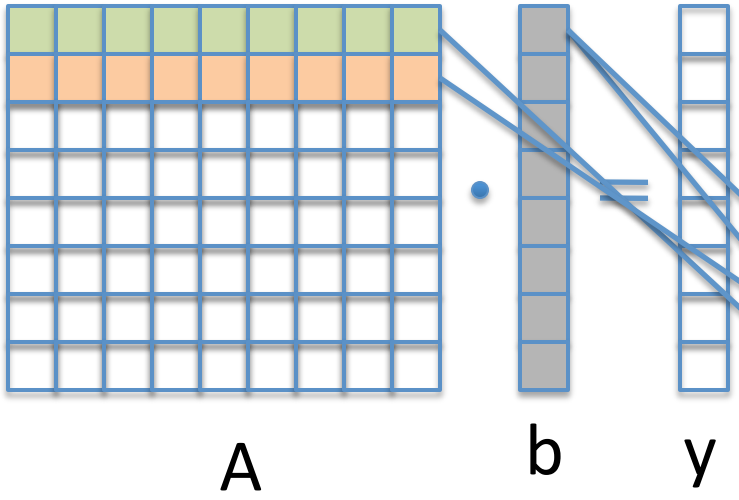
- Access to (a copy of) a row of A
- Access to (or a copy of) of b



Concurrency Example

$A \cdot \bar{b} = \bar{y}$

$m \times n$ Matrix-Vector Multiplication



Thread 1: $[a_{11}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_1$

Thread 2: $[a_{21}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_2$

Thread 3: $[a_{31}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_3$

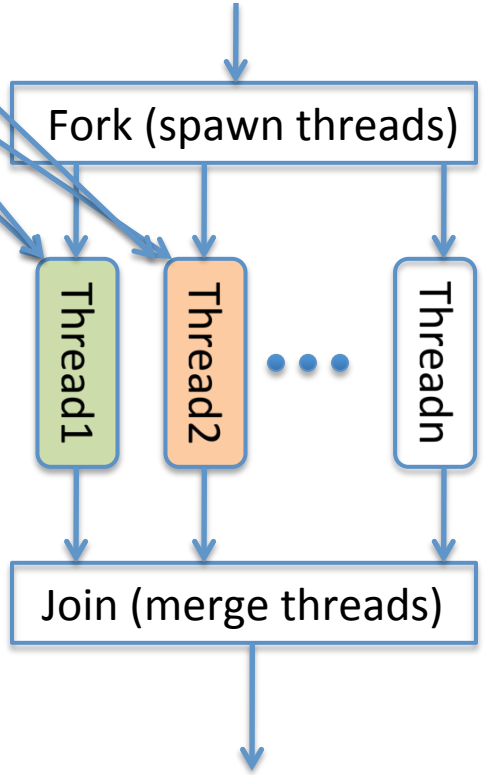
Thread n: $[a_{n1}\text{-}a_{nn}][b_1\text{-}b_n]^T = y_n$

Get it going:

- In Pthreads:
- for (i=1; i<=n; i++) { create_thread(…) }

How do we know when we are done?

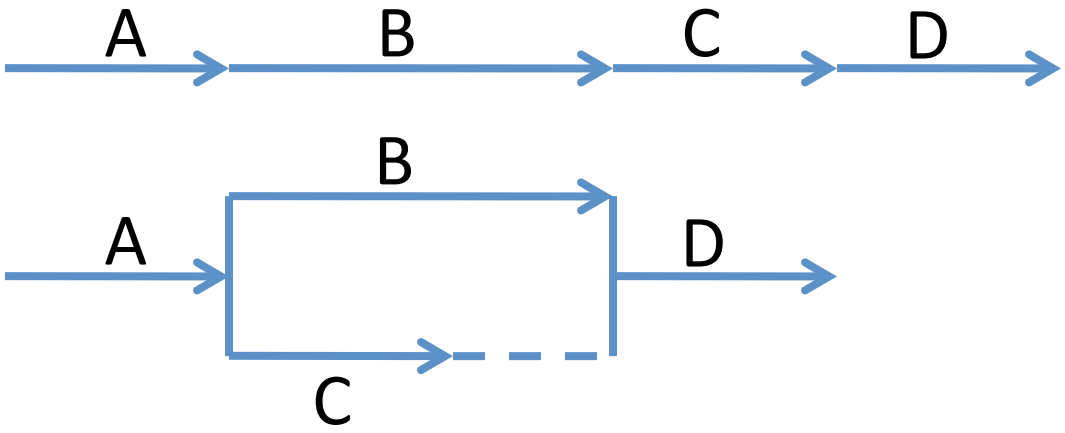
- When all of the threads have completed
- In Pthreads: pthread_join()



Why is Parallel Design Difficult?

1. **Concurrent reasoning** – we're not good at it.
   It's a skill most of us need to learn
   - Play clarinet: one note at a time
   - Play counterpoint on piano: multiple related "sequences"

2. **Structure of the algorithm**– where is the potential concurrency?
   How much time can we really save?

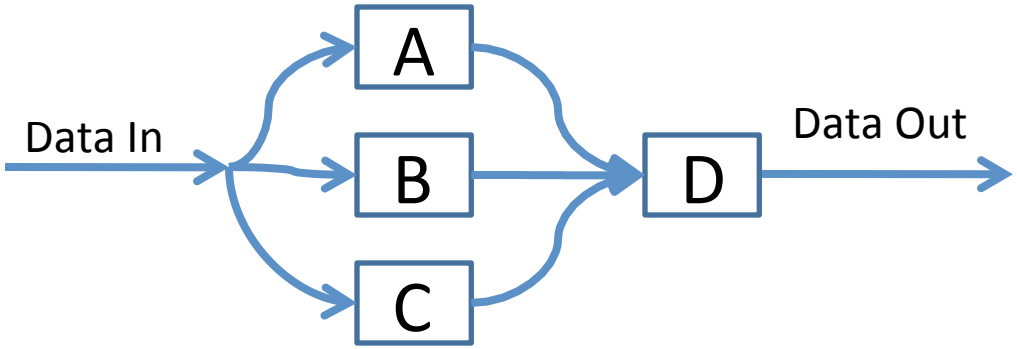


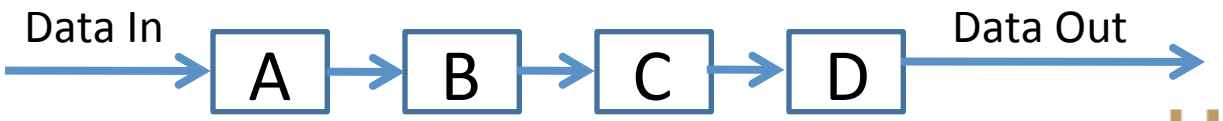
Amdahl's Law: *maximum speedup limited by the parts that have to be done sequentially.*

University of Idaho
College of Engineering

Why is Parallel Design Difficult?

3. **Understanding and partitioning the problem.** At what level can concurrency in the problem be exploited?
   - **Thread level** – different sequential instruction "threads" can execute simultaneously on different cores
     - Write code using threads
     - Scheduling occurs at runtime: depends on O/S
   - **Datapath level** – "loop parallelism"
     - Parallel or systolic pipeline; good for streaming apps





This decision crosses system boundaries

•Outside the domain of a compiler or synthesizer

University of Idaho
College of Engineering

Why is Parallel Design Difficult?

4. **Resource contention.** Access to busses, memory, I/O channels
   - Some tasks must wait. Slows things down.
   - Flush the cache: worse performance than unicore!

5. **Synchronizing independent tasks**
   - Race conditions: result depends on who gets there first
     - E.g Two parallel tasks balancing a checking account
       - Both read balance and modify, then both write. Second write overwrites the first. Wrong data!
   - Deadlock: each task waits for the other to go first

6. **Debugging challenge:** "Heisenbugs" depend on race outcome
   - Now you see 'em, now you don't!

7. **Software.** Lack of mature software support to bring all these parts together

Conclusions

Frequency scaling stalled means we may no longer see the "easy", exponential increase in computer performance.

- Could have serious social and economic consequences.

Solutions:

1. Materials & devices researchers looking for the next super-fast transistor to restore frequency scaling, e.g. 3D
2. Non-von Neumann (non-von) architectures
   - Heterogeneous System on a Chip
     - Huge programming challenge: nonstandard paradigms
3. Parallel multiple von Neumann machines
   - Multicore – multiple processors on a chip
   - Distributed processing – networked clusters
     - Can use standard programming paradigms

Conclusions II

Increased performance requires understanding the problems from many perspective

1. Use cases: how the system will be used
2. Hardware architecture
   - Processor systems
   - Memory systems
   - Interconnect
   - Overall system perspective: how parts interact
3. Algorithms
4. Software
   - Design
   - Programming
   - Operating systems, scheduling