ECE/ME/EMA/CS 759
High Performance Computing for Engineering Applications

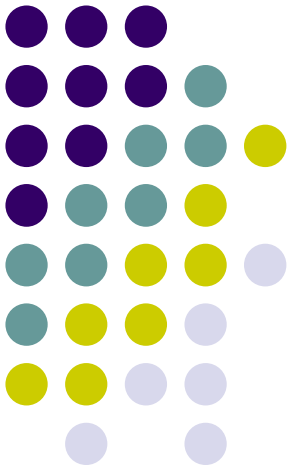
The Shift to Parallel Computing

ILP Wall

Power Wall

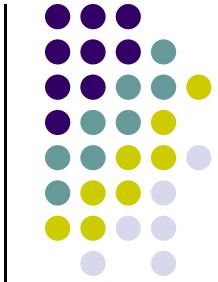
Big Iron HPC

Amdahl's Law



September 18, 2015

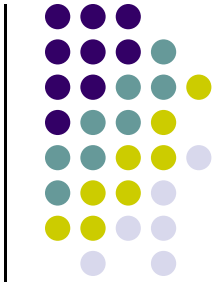
Quote of the Day



“How is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain. Remember when I took that home winemaking course, and I forgot how to drive?”

Homer Simpson

Before We Get Started

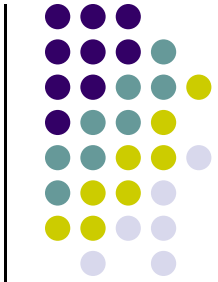


- Issues covered last time:
  - The virtual memory, wrap up.
    - The TLB
  - The shift to parallel computing: three walls to sequential computing
    - The memory wall

- Today's topics
  - The shift to parallel computing: three walls to sequential computing
    - The ILP Wall
    - The Power Wall
  - Big Iron HPC
  - Amdahl's Law

- Assignment:
  - HW03 – posted today and due on September 23 at 11:59 PM
    - No class on Mo and Wd of next week

Instruction Level Parallelism (ILP)



- ILP: a relevant factor in reducing execution times after 1985

- The basic idea:
  - Overlap execution of independent instructions to improve overall performance
  - During the same clock cycle many instructions are being worked upon

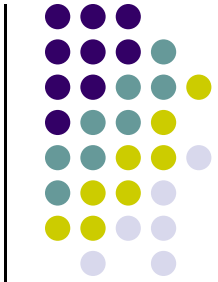
- Two approaches to discovering ILP
  - Dynamic: relies on hardware to discover/exploit parallelism dynamically at run time
    - It is the dominant one in the market
  - Static: relies on compiler to identify parallelism in the code and leverage it (VLIW)

- Examples where ILP expected to improve efficiency

```
for( int=0; i<1000; i++)
    x[i] = x[i] + y[i];
```

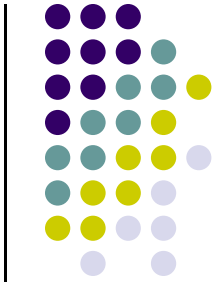
```
1. e = a + b
2. f = c + d
3. g = e * f
```

ILP: Various Angles of Attack

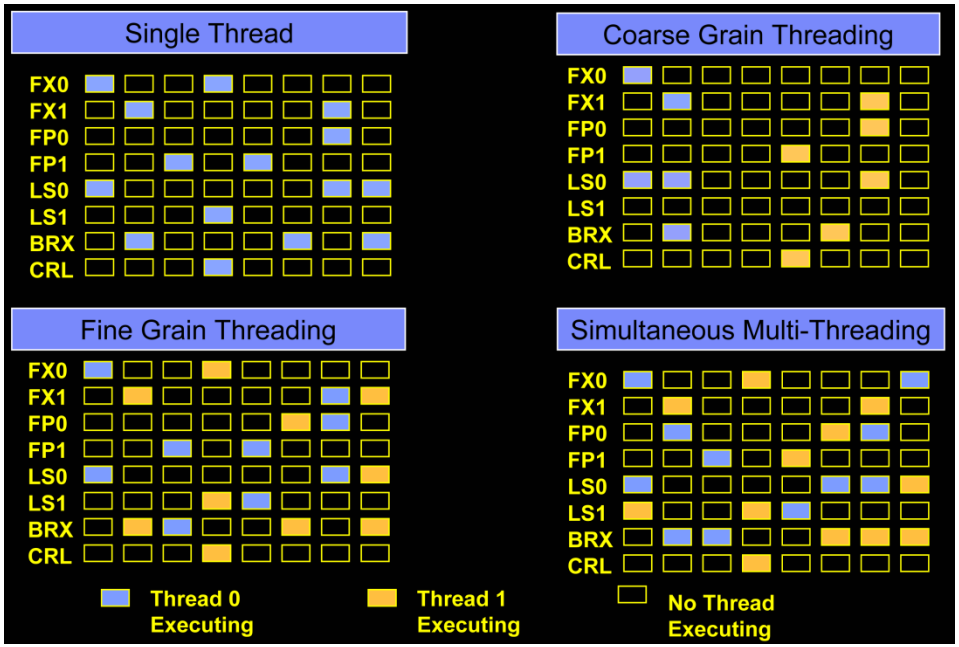


- Instruction pipelining: the execution of multiple instructions can be partially overlapped; where each instructions is divided into series of sub-steps (termed: micro-operations)
- Superscalar execution: multiple execution units are used to execute multiple instructions in parallel
- Out-of-order execution: instructions execute in any order but without violating data dependencies
- Register renaming: a technique used to avoid data hazards and thus lead to unnecessary serialization of program instructions
- Speculative execution: allows the execution of complete instructions or parts of instructions before being sure whether this execution is required
- Branch prediction: used to avoid delays (termed: stalls). Used in combination with speculative execution.

The ILP Journey, Reflected into Instruction Execution

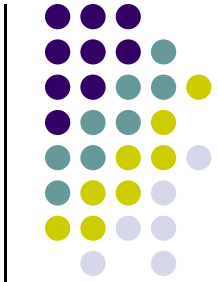


- Squeeze the most out of each cycle…
- Vertical axis: a summary of hardware assets
- Horizontal axis: time



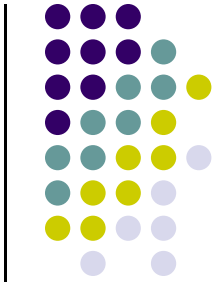
[IBM]→

The ILP Wall



- For ILP to make a dent, you need large blocks of instructions that can be [attempted to be] run in parallel
- Dedicated hardware speculatively executes future instructions before the results of current instructions are known, while providing hardware safeguards to prevent the errors that might be caused by out of order execution
- Branches must be “guessed” to decide what instructions to execute simultaneously
  - If you guessed wrong, you throw away that part of the result
- Data dependencies may prevent successive instructions from executing in parallel, even if there are no branches

The ILP Wall

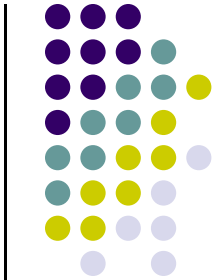


- ILP, the good:
  - Existing programs enjoy performance benefits without any modification
  - Recompiling them is beneficial but entirely up to you as long as you stick with the same ISA (for instance, if you go from Pentium 2 to Pentium 4 you don't have to recompile your executable)

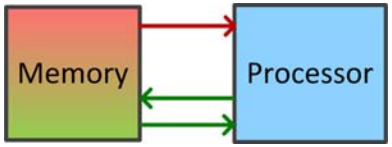
- ILP, the bad:
  - Improvements are difficult to forecast since the "speculation" success is difficult to predict
  - Moreover, ILP causes a super-linear increase in execution unit complexity (and associated power consumption) without linear speedup.

- ILP, the ugly: serial performance acceleration using ILP plateauing because of these effects

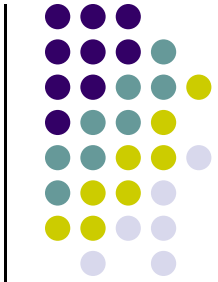
From Simple to Complex: Part 1



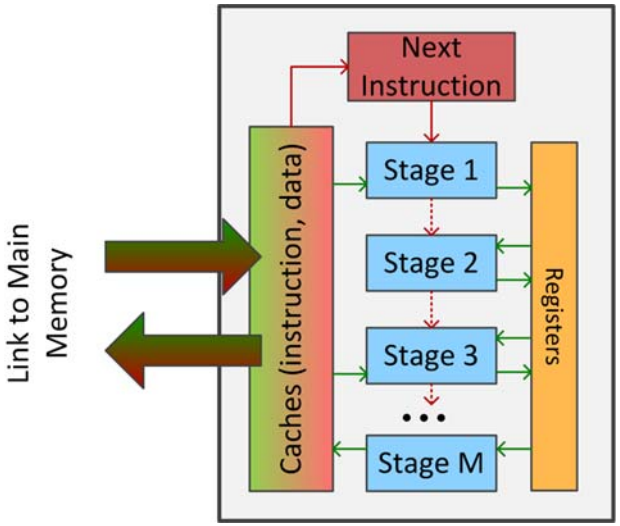
- The von Neumann architecture
  - Red: refers to instructions
  - Green: refers to data



From Simple to Complex: Part 2

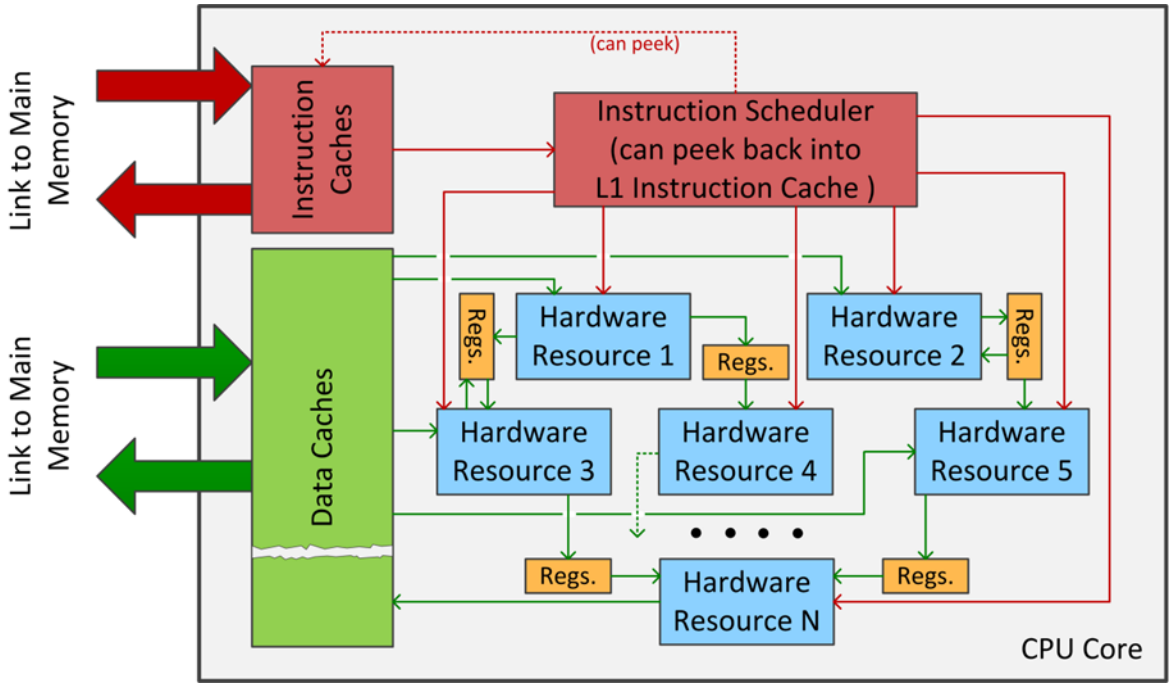


- The architecture of the early to mid 1990s
  - Pipelining was king



From Simple to Complex: Part 3

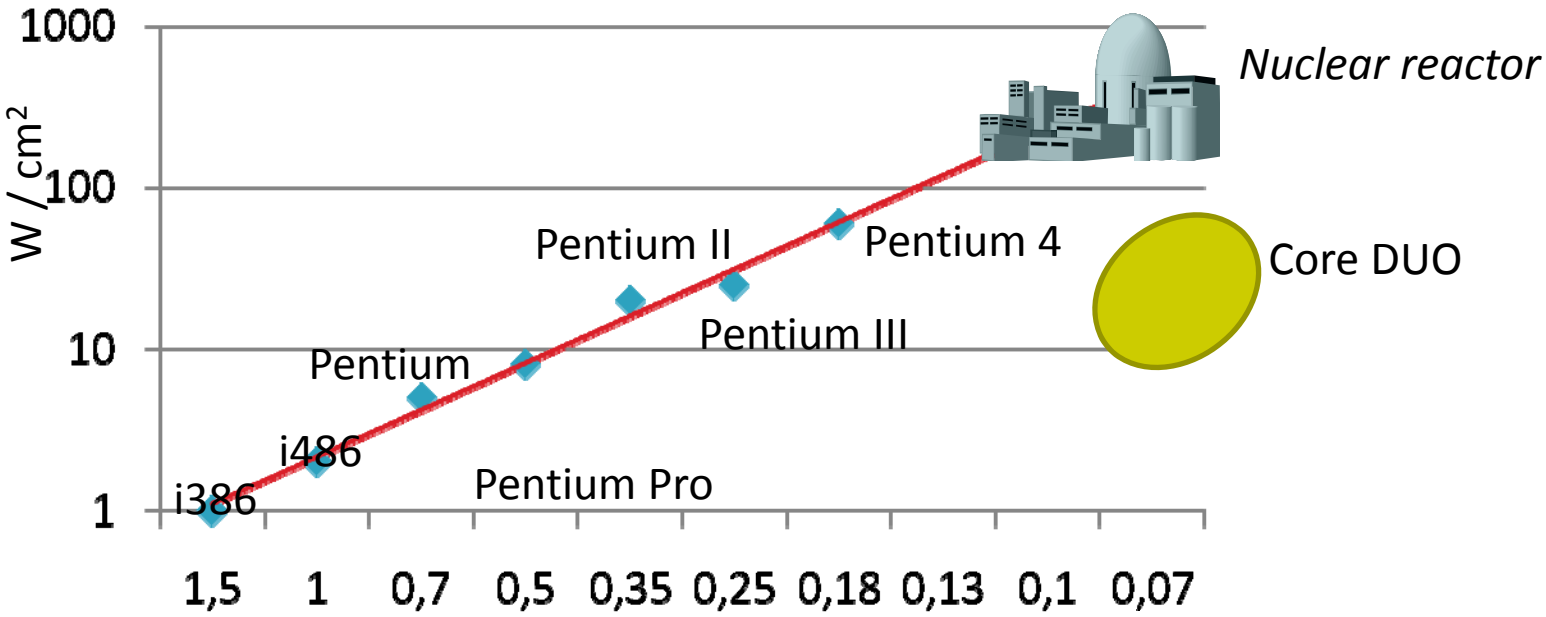
- The architecture of late 1990s, early 2000s
  - ILP-centric



The Power Wall

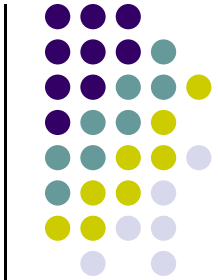
- Power, and not manufacturing, limits traditional general purpose microarchitecture improvements (F. Pollack, Intel Fellow)

- Leakage power dissipation gets worse as gates get smaller, because gate dielectric thicknesses must proportionately decrease



Adapted from F. Pollack (MICRO'99)

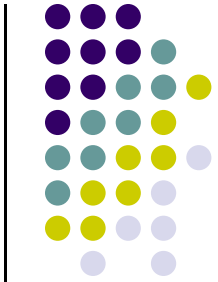
Intel's Perspective



- Intel's "Platform 2015" documentation, see
  http://download.intel.com/technology/computing/archinnov/platform2015/download/RMS.pdf

> First of all, as chip geometries shrink and clock frequencies rise, the transistor leakage current increases, leading to excess power consumption and heat.
> […]
> Secondly, the advantages of higher clock speeds are in part negated by memory latency, since memory access times have not been able to keep pace with increasing clock frequencies.
> […]
> Third, for certain applications, traditional serial architectures are becoming less efficient as processors get faster further undercutting any gains that frequency increases might otherwise buy.

The Power Wall



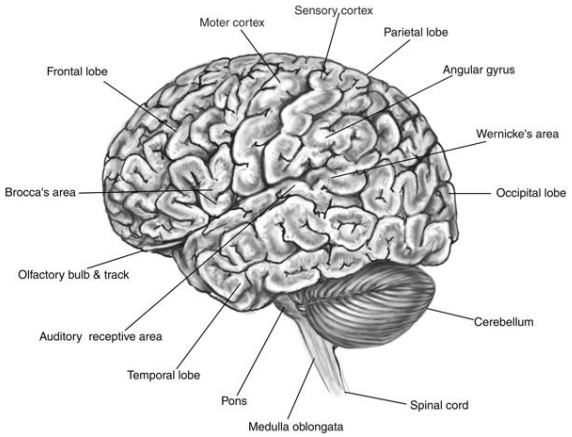
- Power dissipation in clocked digital devices is related to the clock frequency and feature length imposing a natural limit on clock rates
- Significant increase in clock speed without heroic (and expensive) cooling is not possible. Chips would simply melt
- Clock speed increased by a factor of 4,000 in less than two decades
  - The ability of manufacturers to dissipate heat is limited though…
  - Look back at the last ten years, the clock rates are pretty much flat

Trivia

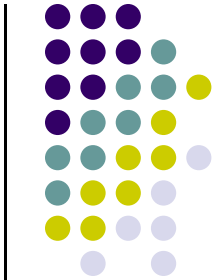
- NVIDIA Tesla K80
  - TDP: 300 Watts

- Intel® Xeon® Processor E7-8890 v3
  - TDP: 165 Watts

- Human Brain
  - 20 W
  - Represents 2% of our mass
  - Burns 20% of all energy in the body at rest
  - Our entire body, ball park: 100 Watts



Conventional Wisdom in Computer Architecture



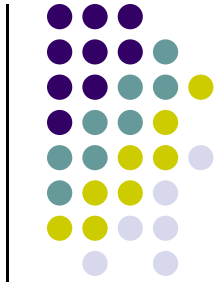
- Old: power is free & transistors expensive
- New: power expensive & transistors free
  (Can put more on chip than can afford to turn on)

- Old: multiplies are slow & memory access is fast
- New: memory slow & multiplies fast ["Memory wall"]
  (400-600 cycles for DRAM memory access, 1 clock cycle for FMA)

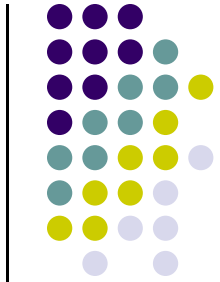
- Old : Increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New: "ILP wall" – diminishing returns on more ILP

- New: Power Wall + Memory Wall + ILP Wall = Brick Wall
  - Old: Uniprocessor performance 2X / 1.5 yrs
  - New: Uniprocessor performance only 2X / 5 yrs?

Credit: D. Patterson, UC-Berkeley



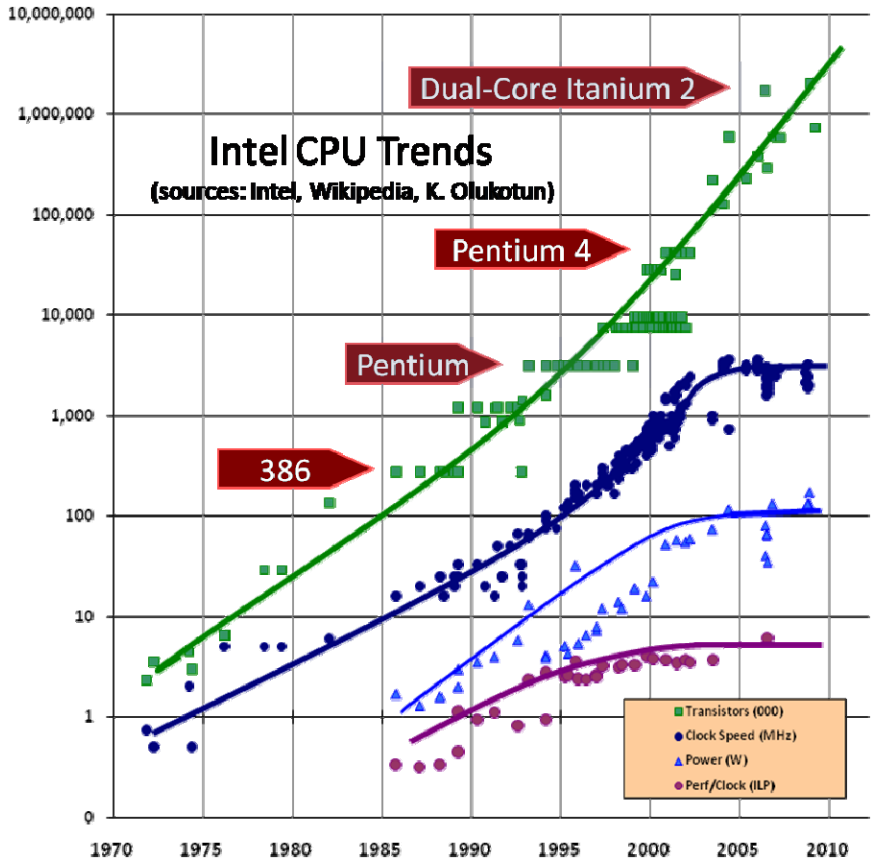
- OK, now what?



- A bunch of bad news…
- … with only one bright spot

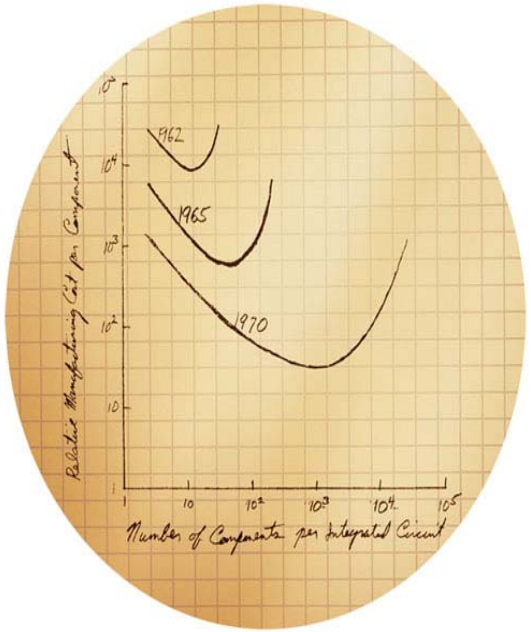
Summarizing It All…

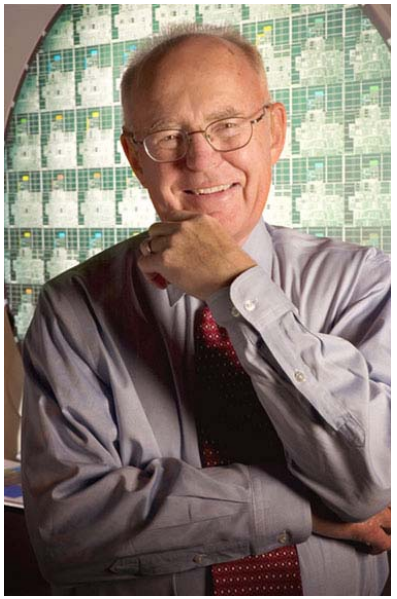
- The sequential execution model is losing steam
- The bright spot: number of transistors per unit area going up and up



Moore's Law

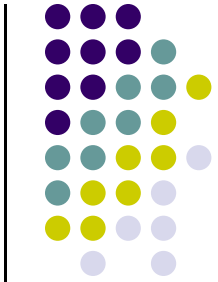
- 1965 paper: Doubling of the number of transistors on integrated circuits every two years
  - Moore himself wrote only about the density of components (or transistors) at minimum cost
- Increase in transistor count is also a rough measure of computer processing performance





http://news.cnet.com/Images-Moores-Law-turns-40/2009-1041_3-5649019.html

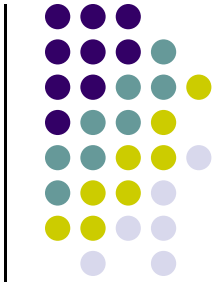
Moore's Law (1965)



- "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year (see graph on next page). Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer."

"Cramming more components onto integrated circuits" by Gordon E. Moore, Electronics, Volume 38, Number 8, April 19, 1965

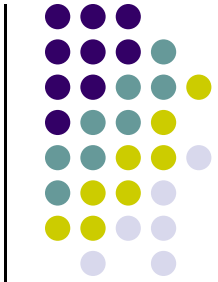
Intel Roadmap



- 2013 – 22 nm Tick: Ivy Bridge – Tock: Haswell
- 2015 – 14 nm Tick: Broadwell – Tock: Skylake
- 2016 – 14nm “Refresh” Kaby Lake
- 2017 – 10 nm Cannonlake (delayed to 2nd Half 2017)
- 2019 – 7 nm
- 2021 – 5 nm
- 2023 – ??? (your turn)

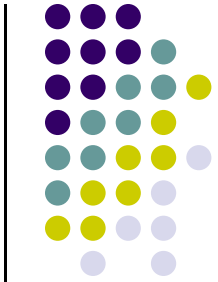
- Moore’s law moving from 18-24 month cycle to 24-30 month cycle for the first time in 50 years

Parallel Computing: Some Black Spots



- More transistors = More computational units
- 2015 Vintage:
  - 18-core Xeon Haswell-EX E7 8890 V3 – 5.6 billion transistors ($7200)
- Black silicon: because of high density and power leaks, we might not be able to power these chips…
  - Black silicon: transistors that today don't get used and are dead weight
  - Dennard's scaling started to break down at the end of last decade
  - Dennard's law is the secrete sauce for Moore's law

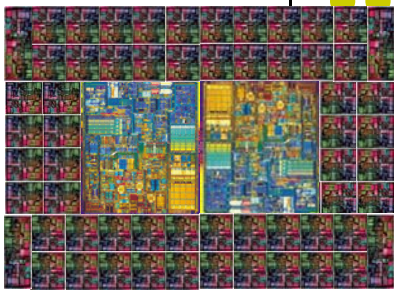
The Ox vs. Chickens Analogy



Seymour Cray: "If you were plowing a field, which would you rather use: Two strong oxen or 1024 chickens?"

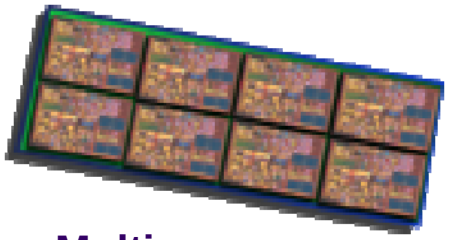
- Chicken is gaining momentum nowadays:
  - For certain classes of applications, you can run many cores at lower frequency and come ahead at the speed game
- Example:
  - Scenario One: one-core processor w/ power budget W
    - Increase frequency by 20%
      - Substantially increases power, by more than 50%
      - But, only increase performance by 13%
  - Scenario Two: Decrease frequency by 20% with a simpler core
    - Decreases power by 50%
    - Can now add another dumb core (one more chicken…)

Intel's Vision: Evolutionary Configurable Architecture



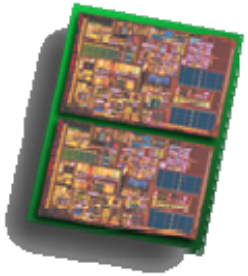
Scalar plus many core for highly threaded workloads

Large, Scalar cores for high single-thread performance



Multi-core array

- **CMP with ~10 cores**



Dual core

- **Symmetric multithreading**

Many-core array

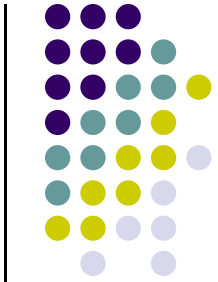
- **CMP with 10s-100s low power cores**
- **Scalar cores**
- **Capable of TFLOPS+**
- **Full System-on-Chip**
- **Servers, workstations, embedded…**

Evolution

CMP = "chip multi-processor"

Presentation Paul Petersen, Sr. Principal Engineer, Intel

Two Examples of Parallel HW:
How Billions of Transistors are Put to Good Use

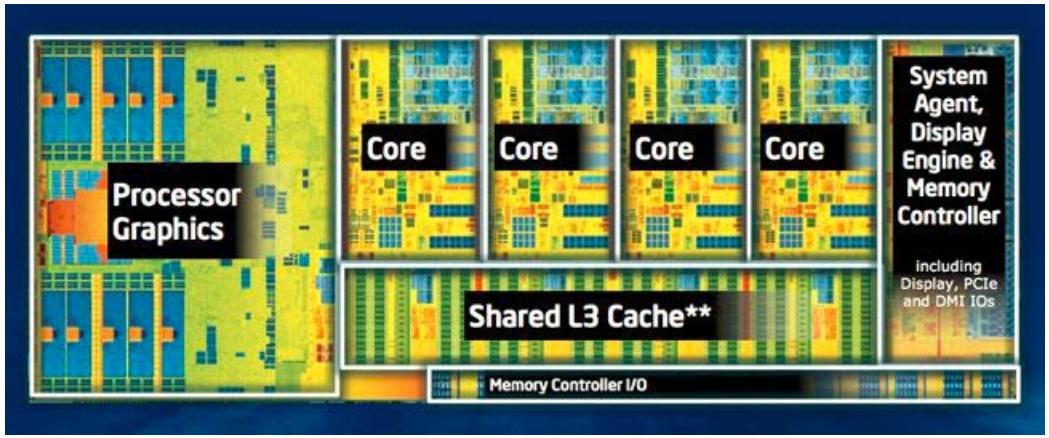


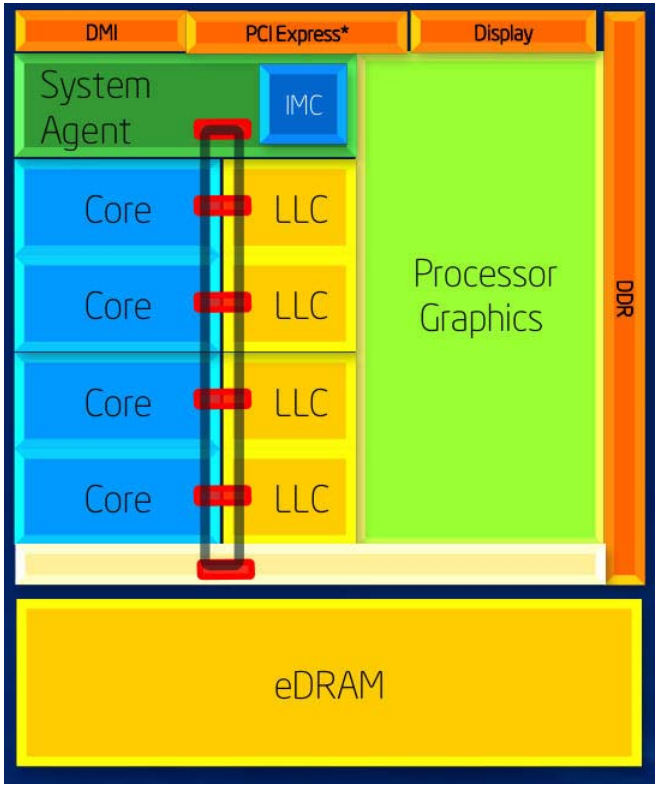
- Intel Haswell
  - Multicore architecture

- NVIDIA Fermi
  - Large number of scalar processors (“shaders”)

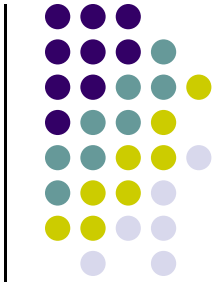
Intel Haswell

- June 2013
- 22 nm technology
- 1.4 billion transistors
- 4 cores, hyperthreaded
- Integrated GPU
- System-on-a-chip (SoC) design



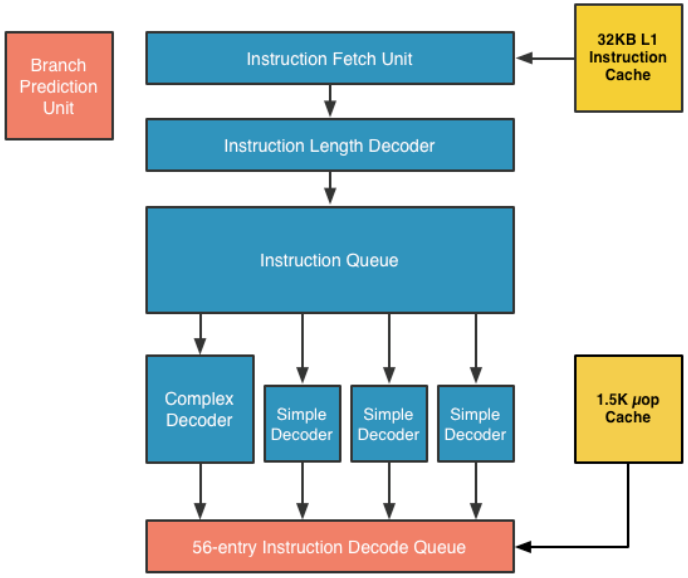


Intel Haswell: Front End and Back End

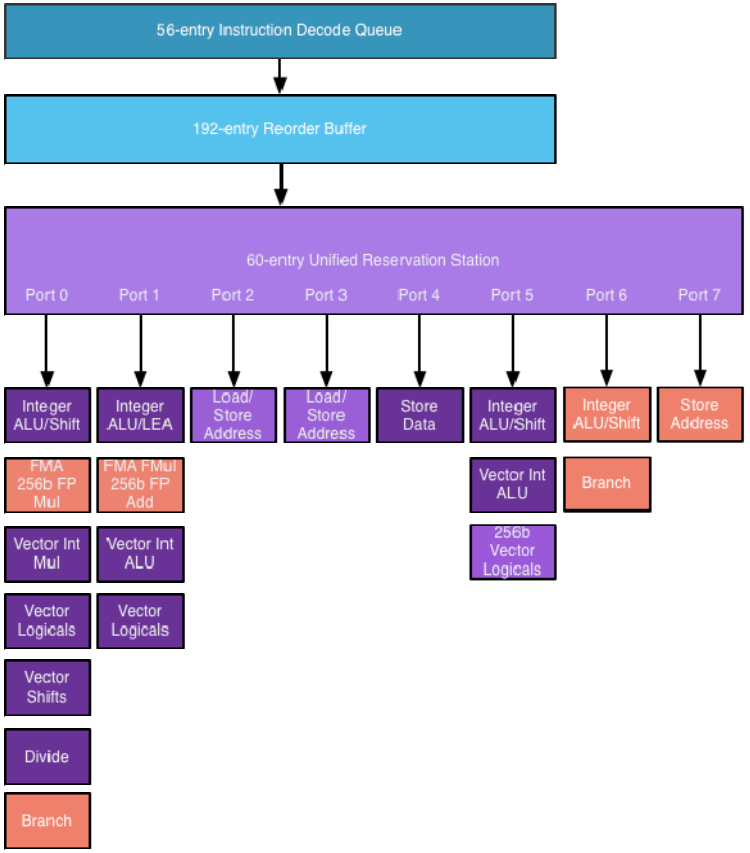


- A high level organization:
  - Decoding
  - Scheduling
  - Execution

Intel Haswell Front End



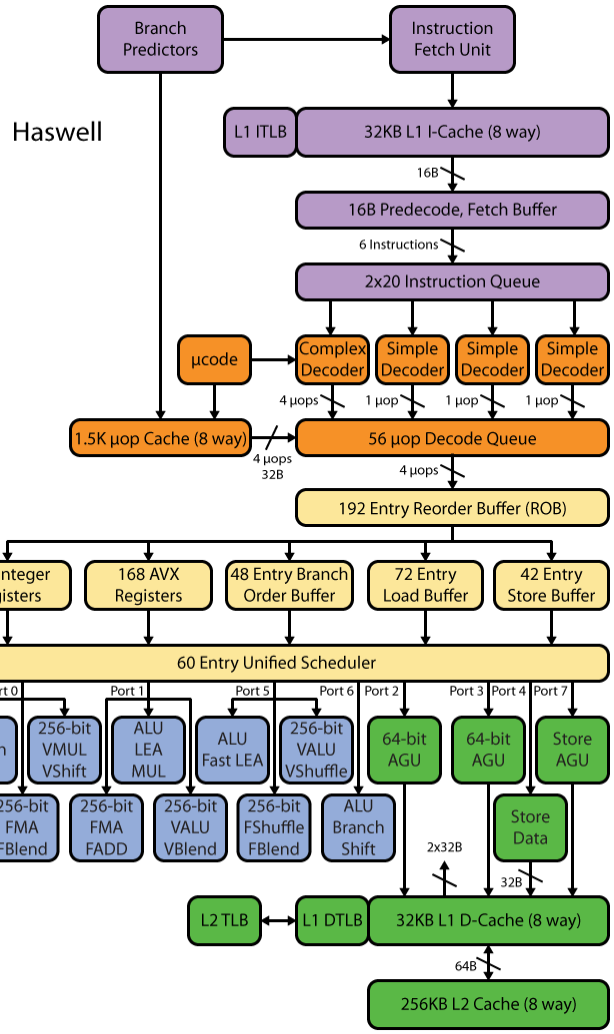
Intel Haswell Execution Engine



[http://www.realworldtech.com/haswell-cpu/]→

Intel Haswell: Overall Perspective

- At the right: complete schematic of microarchitecture
- More info: see online primer

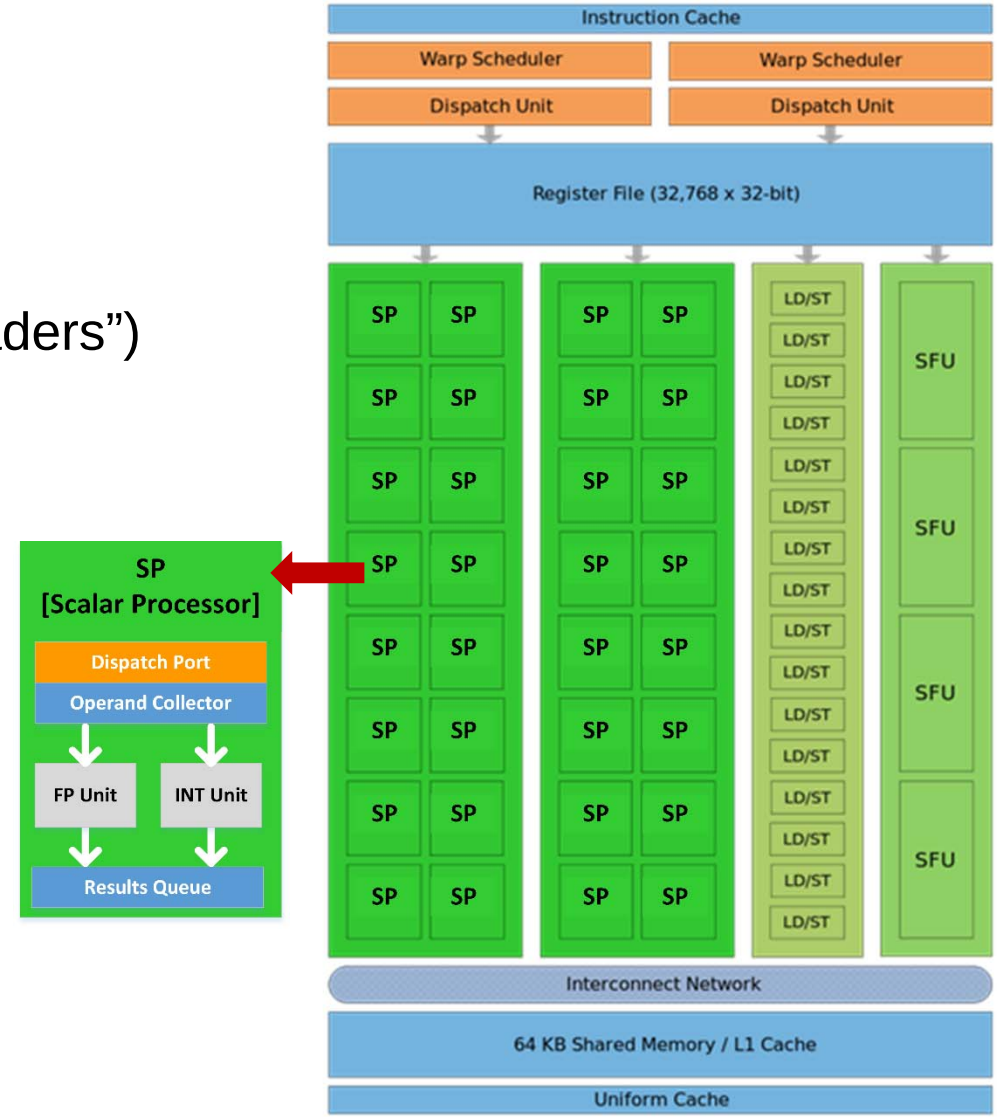


[http://www.realworldtech.com/haswell-cpu/]→

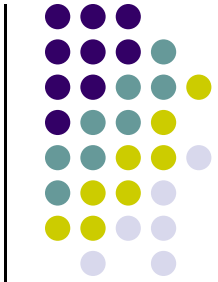
The Fermi Architecture

[Schematic of a NVIDIA Stream Multiprocessor (SM)]

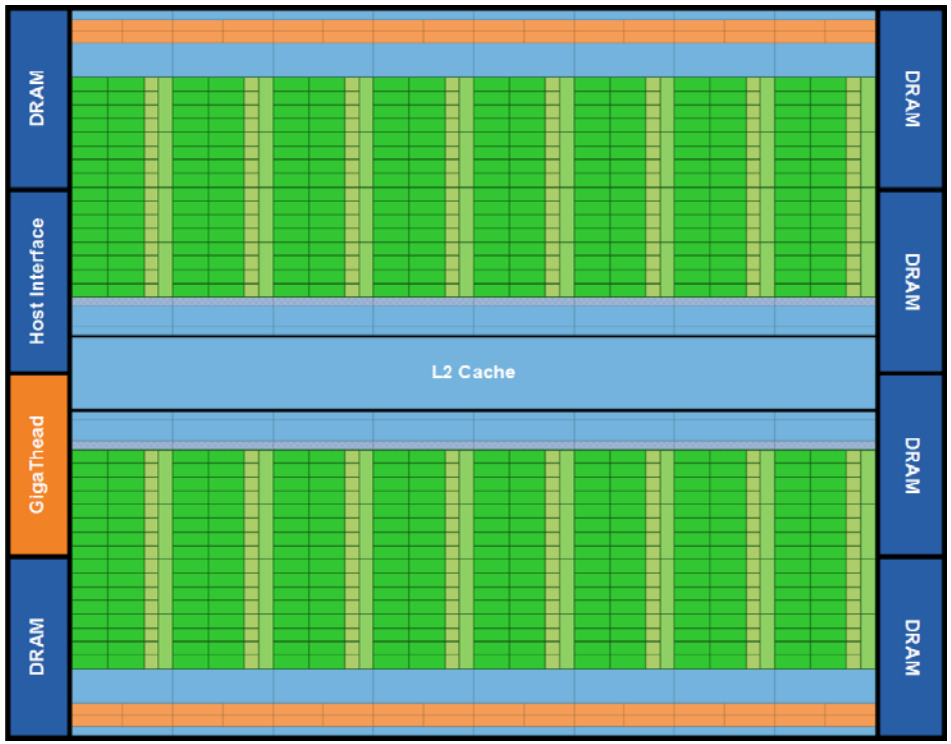
- Late 2009, early 2010
- 40 nm technology
- Three billion transistors
- 512 Scalar Processors (SP, “shaders”)
- L1 cache
- L2 cache
- 6 GB of global memory
- Operates at low clock rate
- High bandwidth (close to 200 GB/s)



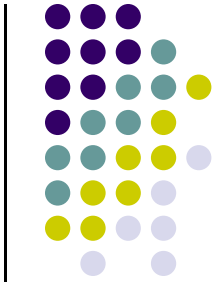
Fermi: 30,000 Feet Perspective

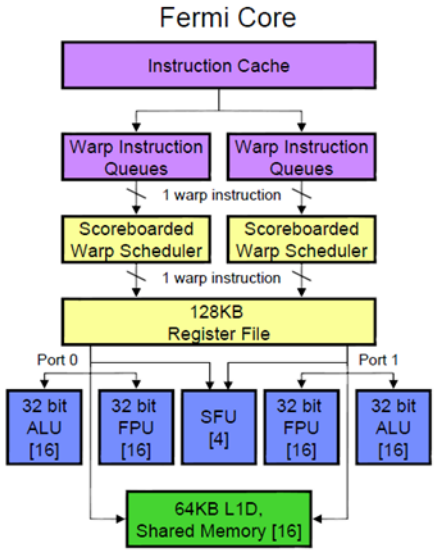


- Lots of ALU (green), not much of CU (orange)
- Explains why GPUs are fast for high arithmetic intensity applications
- Arithmetic intensity: high when many operations performed per word of memory

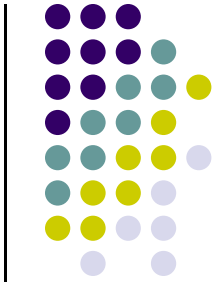


A Fermi Core
(or SM – Streaming Multiprocessor)

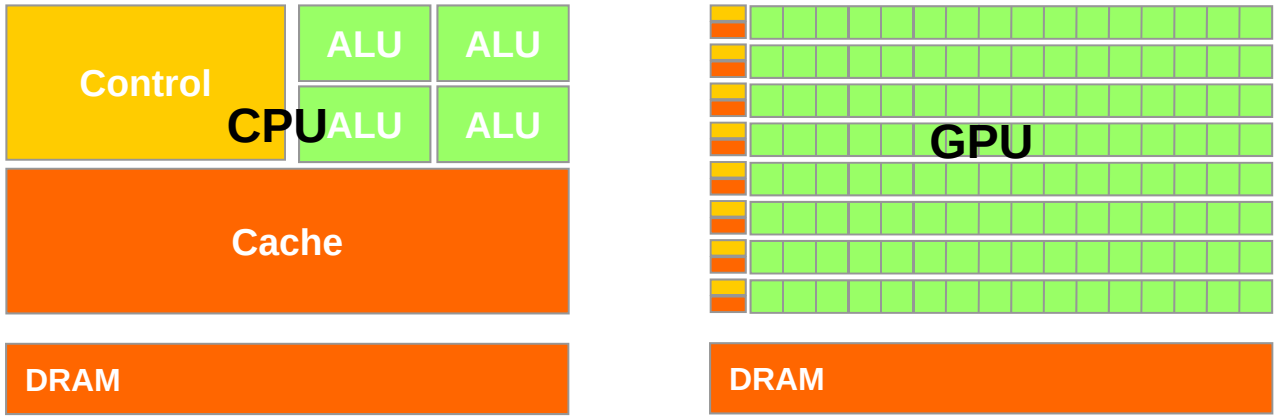




Why is the GPU Fast?

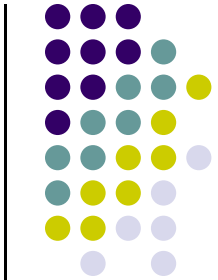


- The GPU is specialized for compute-intensive, highly data parallel computation (owing to its graphics rendering origin)
  - More transistors devoted to data processing rather than data caching and control flow
  - Where are GPUs good: high arithmetic intensity (the ratio between arithmetic operations and memory operations)



- The large video game industry exerts strong economic pressure that forces constant innovation in GPU computing

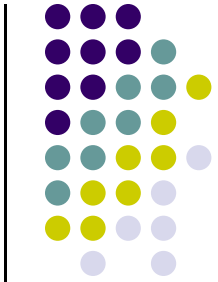
Key Parameters GPU, CPU



	GPU – NVIDIA Tesla K80 (Dual GPU card) $4400	CPU – Intel® Xeon® Processor E7-8890 v3 $7200
Processing Cores	2496x2 (4992 total)	18 cores (36 threads)
Memory	12GBx2 (24GB)	- 64 KB L1 cache / core - 256 KB L2 (I&D) cache / core - 45 MB L3 (I&D) shared by all cores
Clock speed	562 MHz each	2.5 GHz
Memory bandwidth	240 GB/s per GPU	102 GB/s
Floating point operations/s	2910×10^9 Double Precision 8740×10^9 Single Precision	532×10^9 Double Precision
TDP	300 Watts	165 Watts

https://en.wikipedia.org/wiki/Nvidia_Tesla
http://www.techpowerup.com/cpudb/1819/xeon-e7-8890-v3.html
https://software.intel.com/en-us/articles/improve-server-application-performance-with-intel-advanced-vector-extensions-2

Shift in Perception, Left to Right



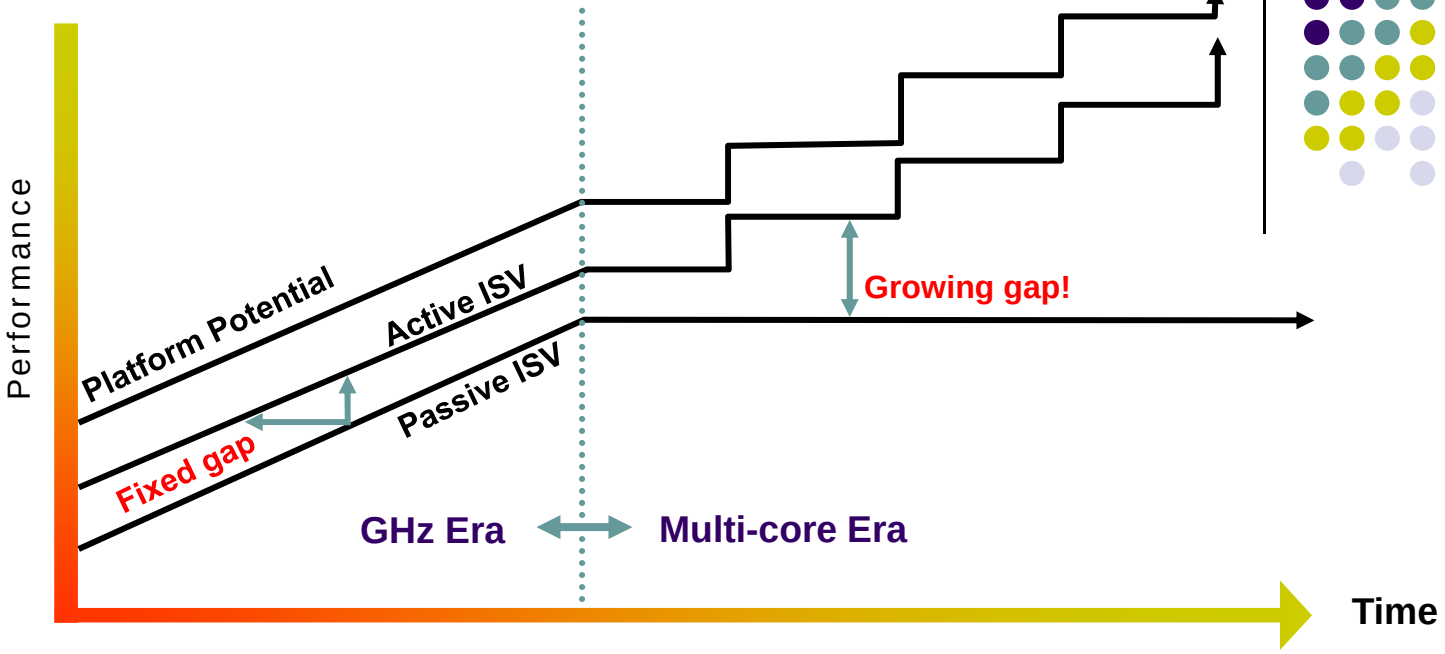
- Increasing clock frequency is primary method of performance improvement

- Processors parallelism is primary method of performance improvement

- Less than linear scaling for a multiprocessor is failure

- Given the switch to parallel hardware, even sub-linear speedups are beneficial as long as you beat the sequential on a watt-to-watt basis

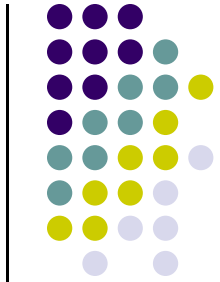
Implications in the Software Business



ISV: Independent Software Vendors

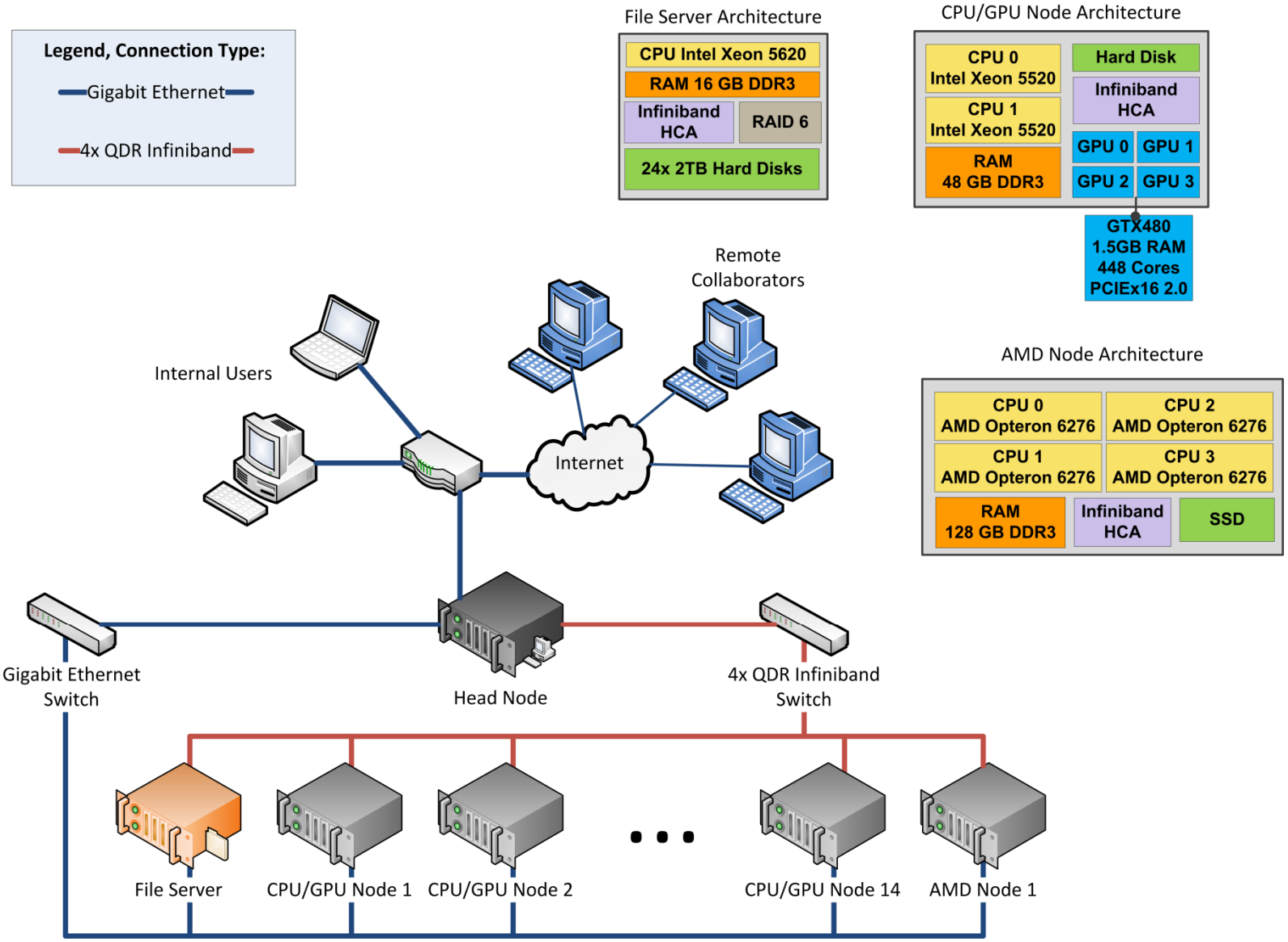
- "Parallelism for Everyone"
- Parallelism changes the game
  - A large percentage of people who provide applications are going to have to care about parallelism in order to match the capabilities of their competitors.

competitive pressures = demand for parallel applications

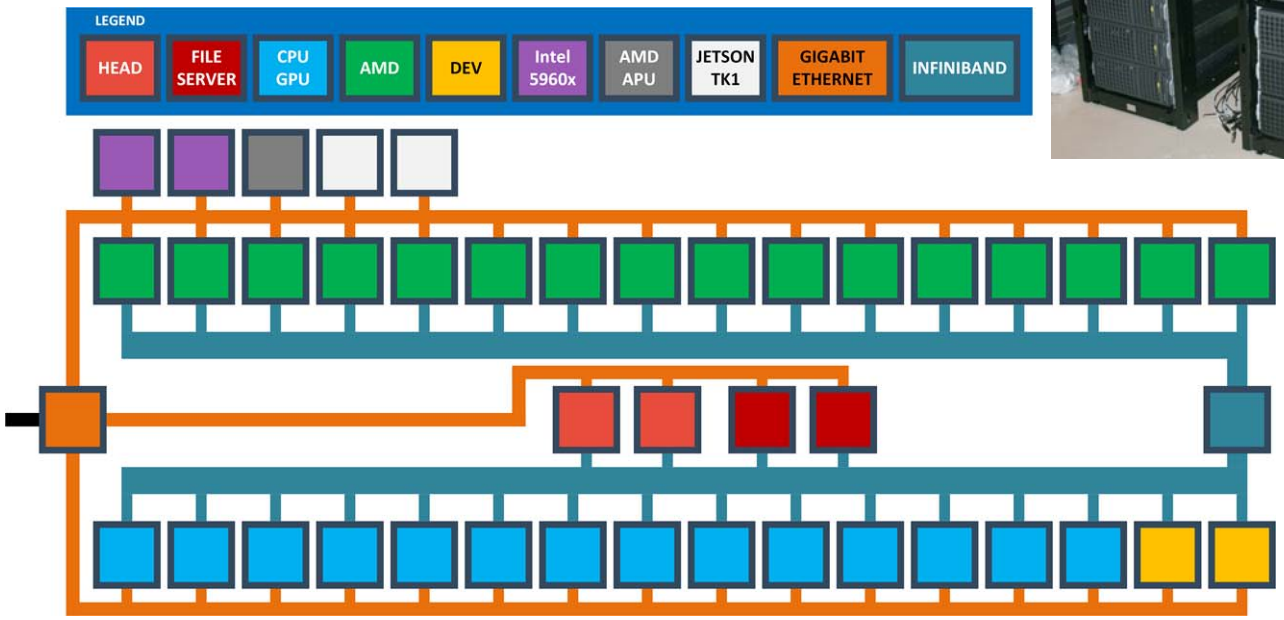


“Big Iron” Parallel Computing

Euler: CPU/GPU Heterogeneous Cluster
~ Hardware Configuration ~

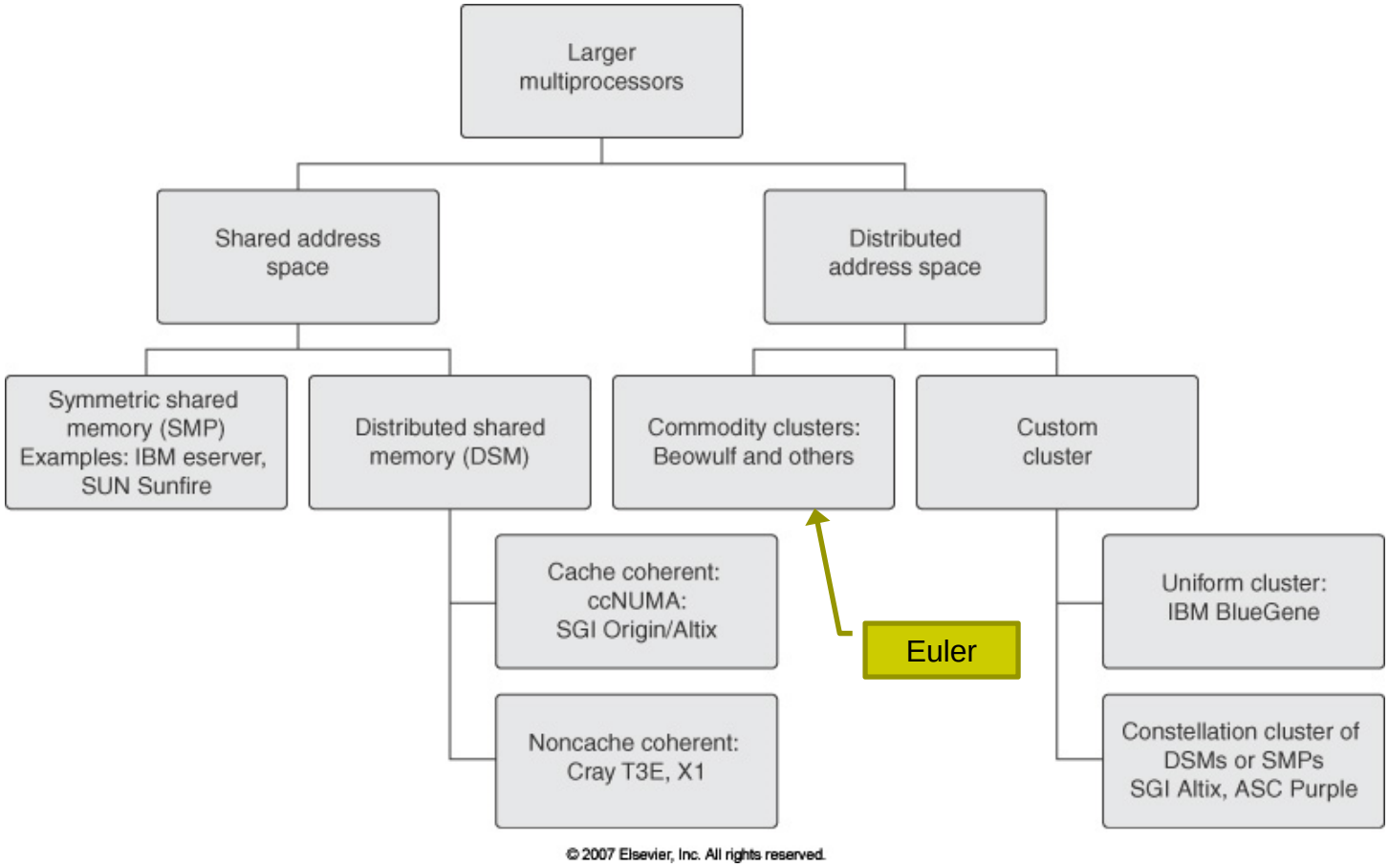


Background: Lab's Research Cluster



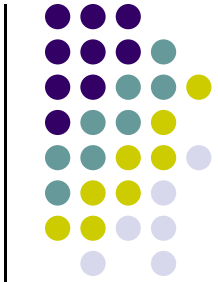
EULER - Heterogeneous Research Cluster.

Overview of Large Multiprocessor Hardware Configurations ("Big Iron")



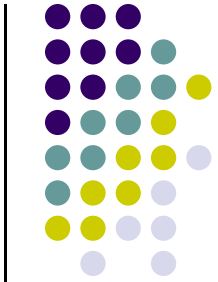
Courtesy of Elsevier, Computer Architecture, Hennessey and Patterson, fourth edition

Some Nomenclature…

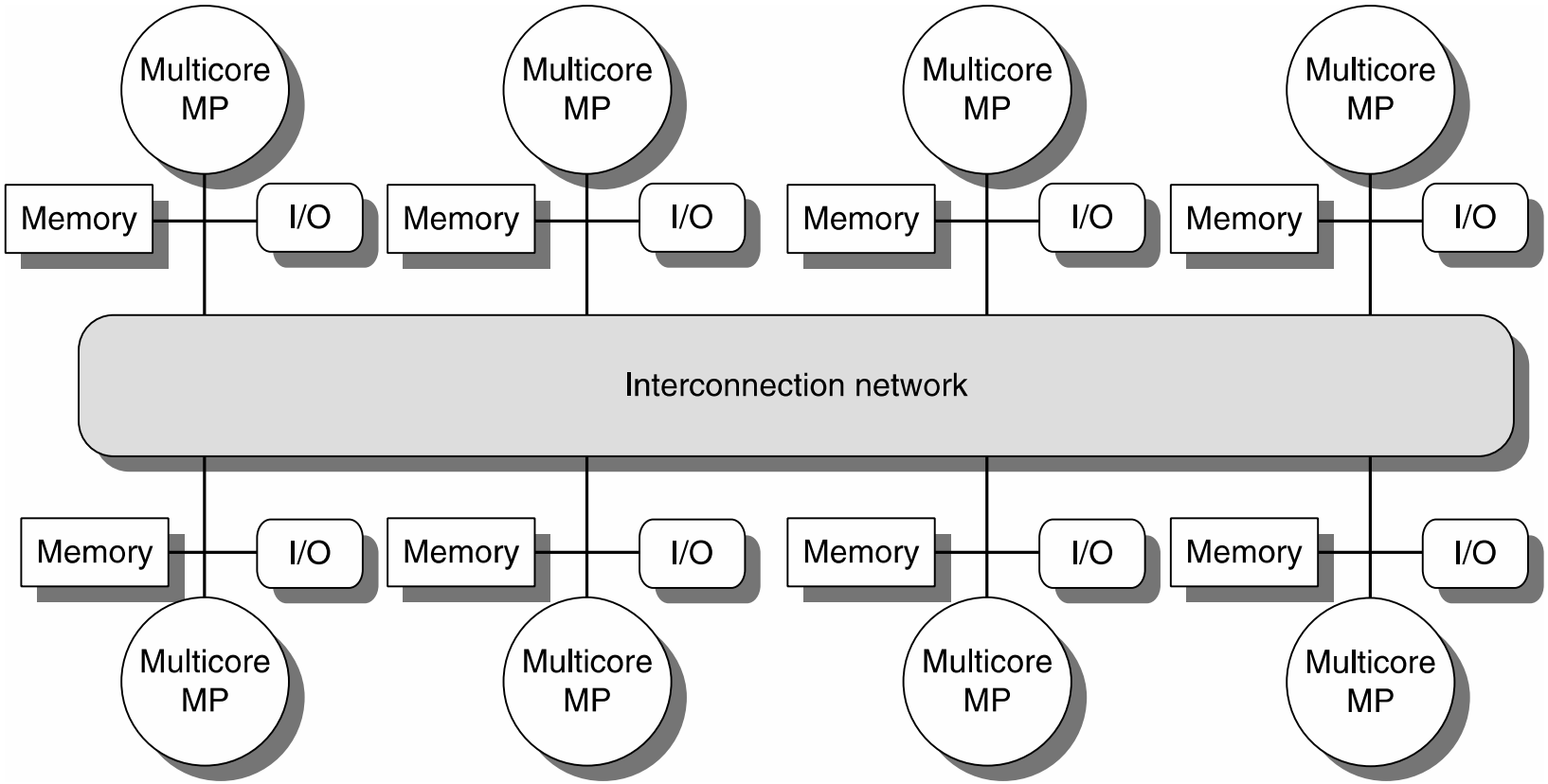


- Shared addressed space: when you invoke address “0x0043fc6f” on one machine and then invoke “0x0043fc6f” on a different machine they actually point to the same global memory space
  - Issues: memory coherence
    - Fix: software-based or hardware-based
- Distributed addressed space: the opposite of the above
- Symmetric Multiprocessor (SMP): you have one machine that shares amongst all its processing units a certain amount of memory (same address space)
  - Mechanisms should be in place to prevent data hazards (RAW, WAR, WAW). Brings back the issue of memory coherence
- Distributed shared memory (DSM) – aka distributed global address space (DGAS)
  - Although physically memory is distributed, it shows as one uniform memory
  - Memory latency is highly unpredictable

Example

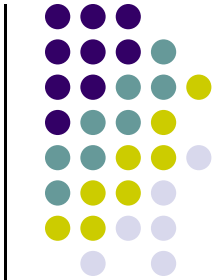


- Distributed-memory multiprocessor architecture (Euler, for instance)



Courtesy of Elsevier, Computer Architecture, Hennessey and Patterson, fifth edition

Comments, distributed-memory multiprocessor architecture



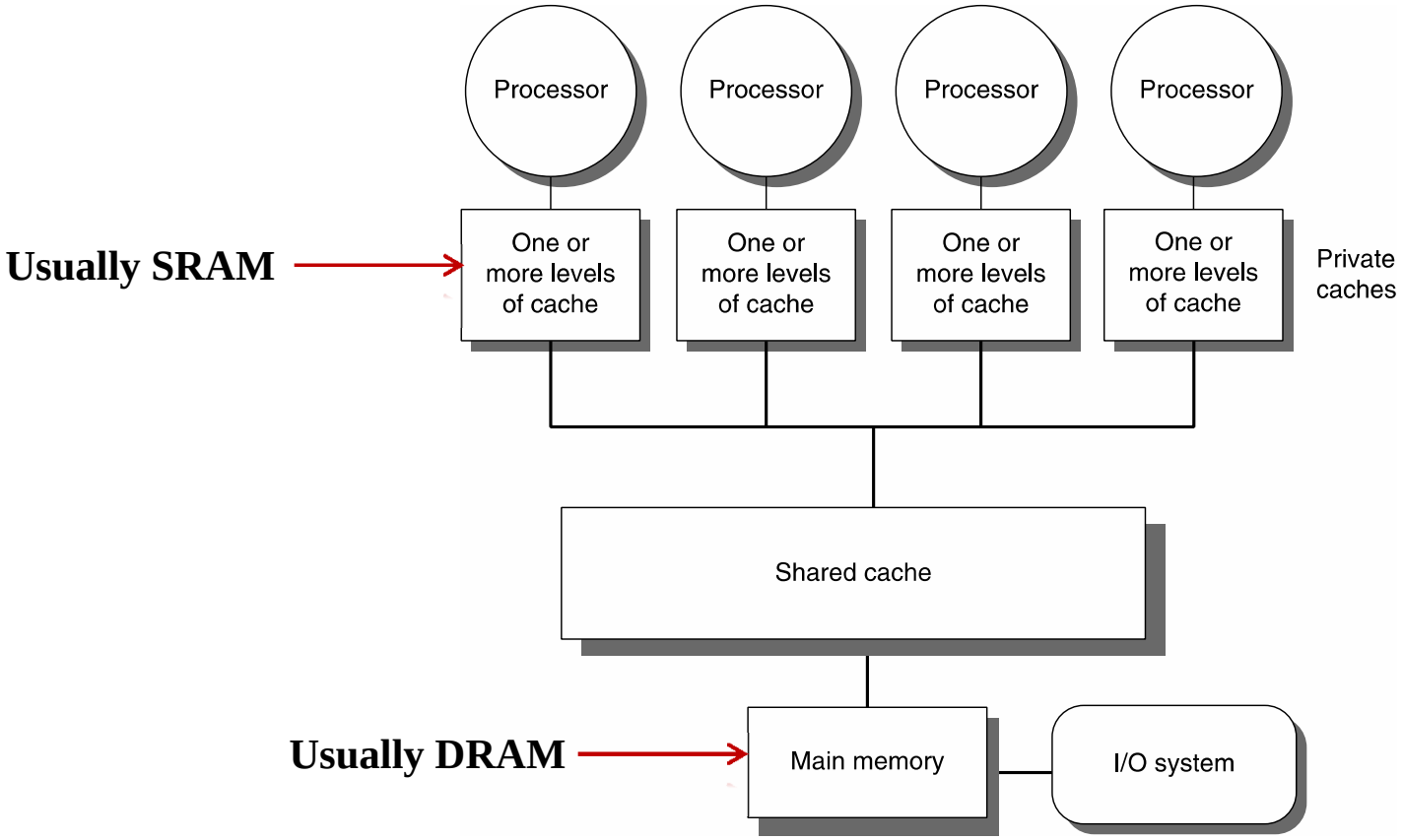
- Basic architecture consists of nodes containing a processor, some memory, typically some I/O, and an interface to an interconnection network that connects all nodes

- Individual nodes may contain a small number of processors, which may be interconnected by a small bus or a different interconnection technology, which is less scalable than the global interconnection network

- Popular interconnection network: Mellanox and Qlogic InfiniBand
  - Bandwidth range: 1 through 50 Gb/sec (about 6 GB/s)
  - Latency: in the microsecond range (approx. 1E-6 seconds)
  - Requires special network cards: HCA – “Host Channel Adaptor”

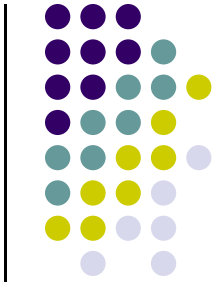
Example, SMP
[This is not “Big Iron”, rather a desktop nowadays]

- Shared-Memory Multiprocessor Architecture



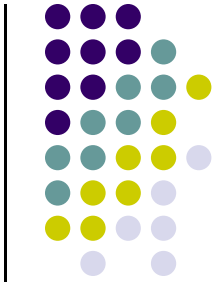
Courtesy of Elsevier, Computer Architecture, Hennessey and Patterson, fifth edition

Comments, SMP Architecture



- Multiple processor-cache subsystems share the same physical off-chip memory
- Typically connected to this off-chip memory by one or more buses or a switch
- Key architectural property: uniform memory access (UMA) time to all of memory from all the processors
  - This is why it's called symmetric

Examples…

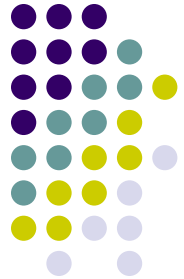


- Shared-Memory
  - Intel Xeon Phi available as of 2012
    - Packs 61 cores, which are on the basic (unsophisticated) side
  - AMD Opteron 6200 Series (16 cores: Opteron 6276) – Bulldozer architecture
  - Sun Niagara

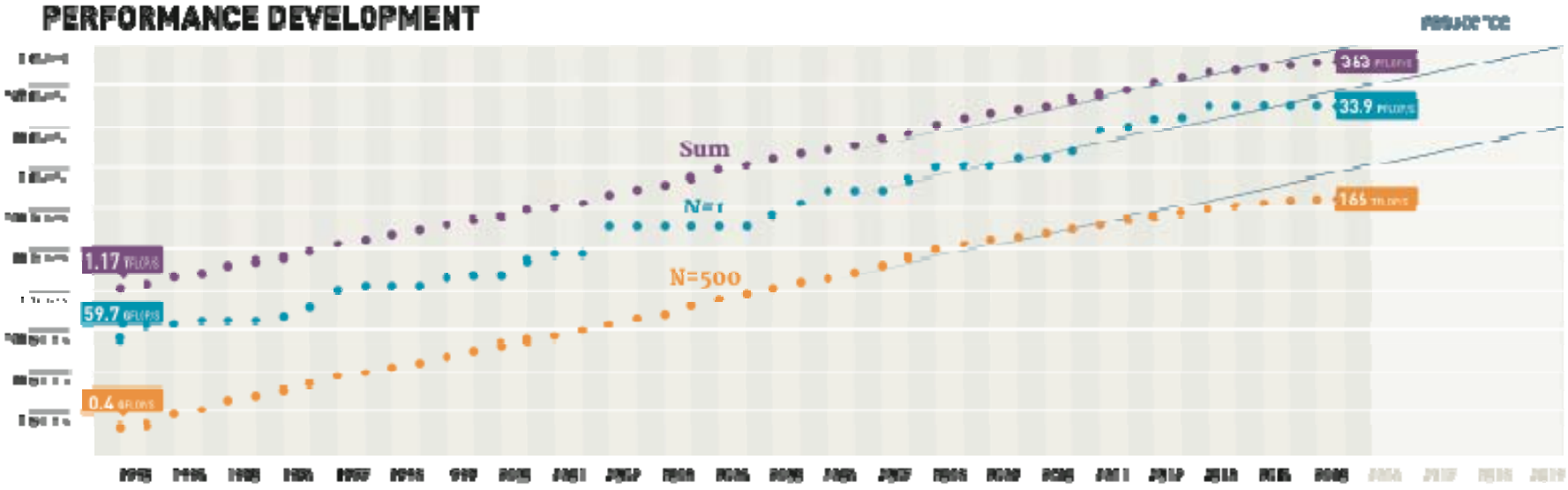
- Distributed-Memory
  - IBM BlueGene/L
  - Cell (see http://users.ece.utexas.edu/~adnan/vlsi-07/hofstee-cell.ppt)

Big Iron: Where Are We Today?

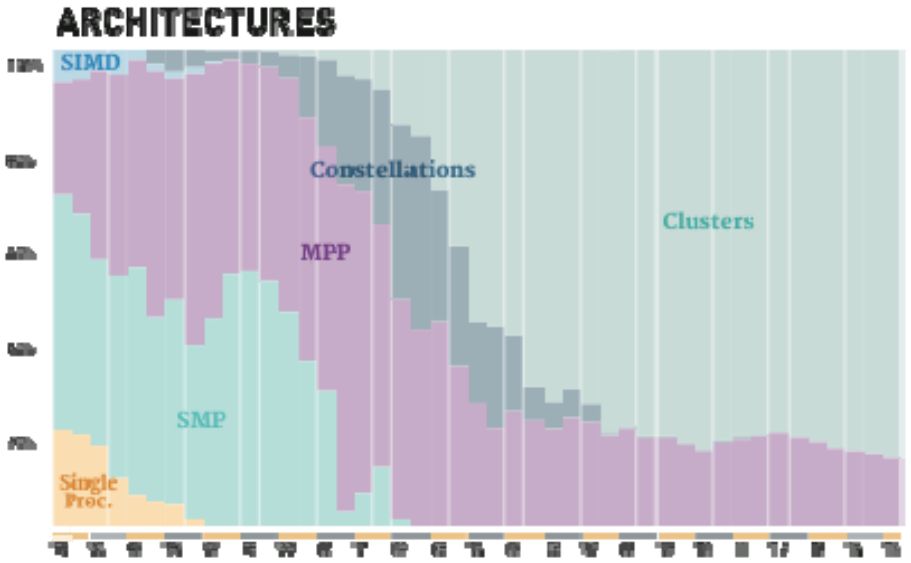
[Info lifted from Top500 website: http://www.top500.org/]

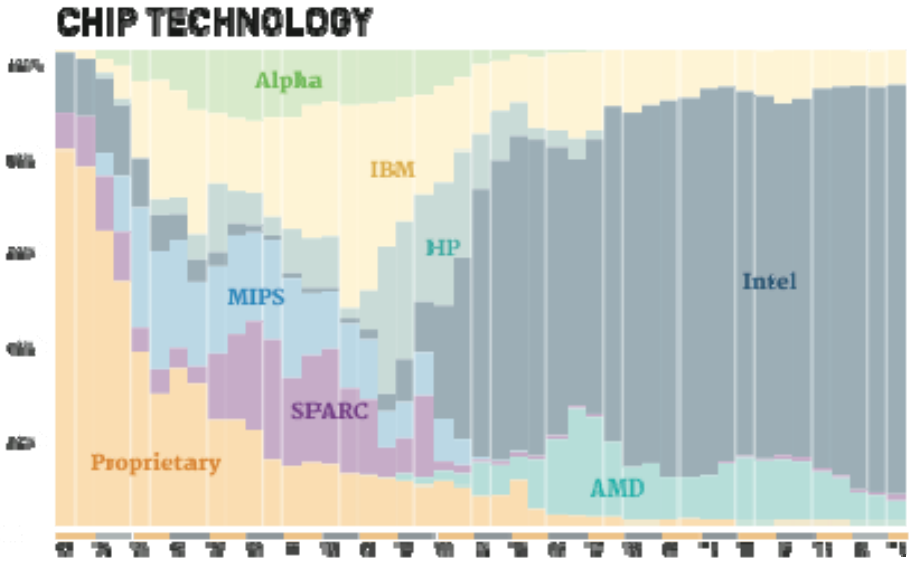


1	Tianhe-2 (Milkyway-2)	Intel Ivy Bridge (12C 2.2 GHz) & Xeon Phi (57C 1.1 GHz), Custom interconnect	NUDT	China	3,120,000	33.9	17.8
2	Titan	Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect	DOE/SC/ORNL	USA	560,640	17.6	8.2
3	Sequoia	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	17.2	7.9
4	K computer	Fujitsu SPARC64 VIIIfx (8C 2.0 GHz), Custom interconnect	RIKEN AICS	Japan	705,024	10.5	12.7
5	Mira	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/SC/ANL	USA	786,432	8.59	3.95



Big Iron: Where Are We Today?
[Cntd.]

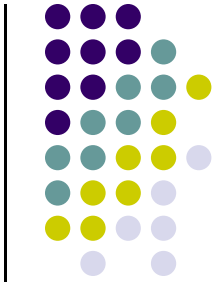




- Abbreviations/Nomenclature
  - MPP – Massively Parallel Processing
  - Constellation – subclass of cluster architecture envisioned to capitalize on data locality
  - MIPS – “Microprocessor without Interlocked Pipeline Stages”, a chip design of the MIPS Computer Systems of Sunnyvale, California
  - SPARC – “Scalable Processor Architecture” is a RISC instruction set architecture developed by Sun Microsystems (now Oracle) and introduced in mid-1987
  - Alpha - a 64-bit reduced instruction set computer (RISC) instruction set architecture developed by DEC (Digital Equipment Corporation was sold to Compaq, which was sold to HP) – adopted by Chinese chip manufacturer (see primer)

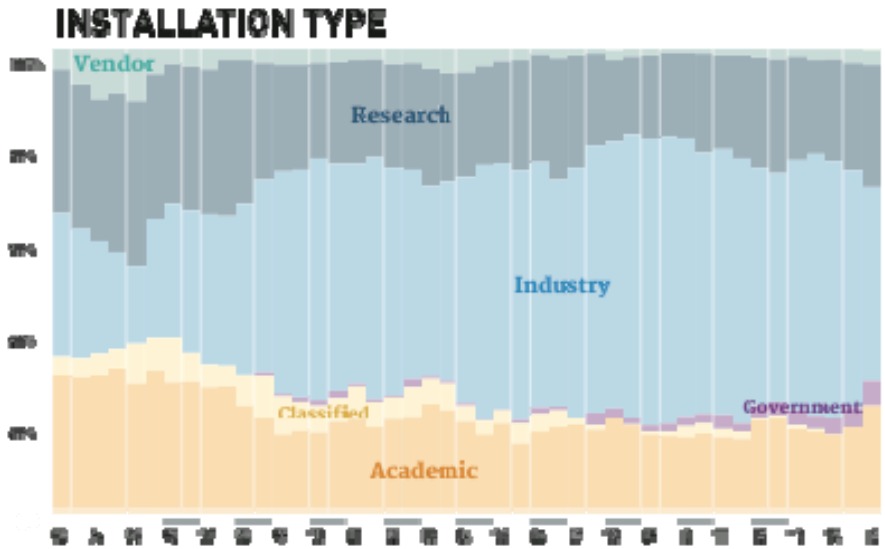
Short Digression: Massively Parallel Processing

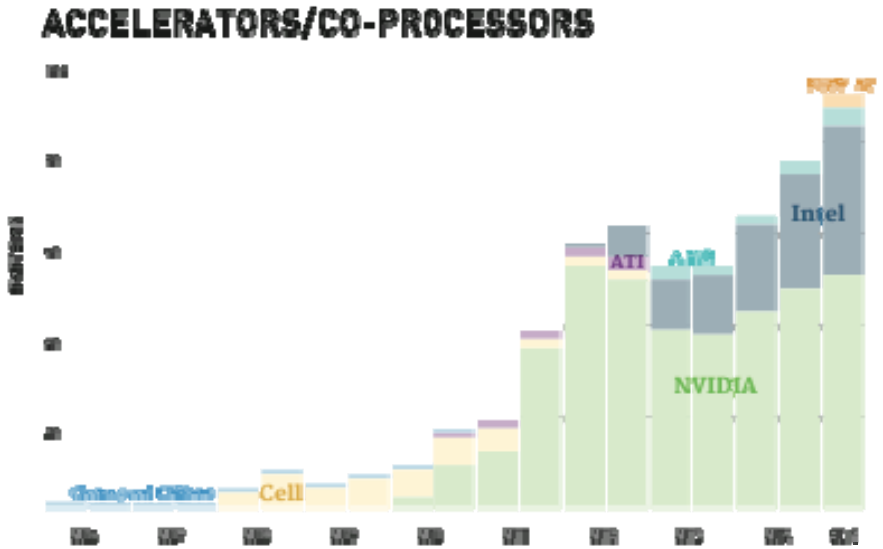
What is a MPP?



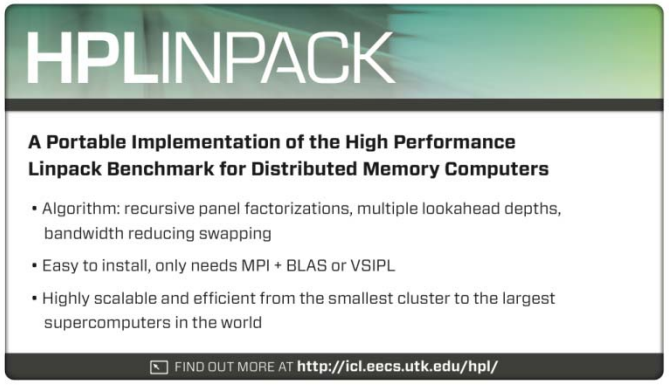
- A very large-scale computing system with commodity processing nodes interconnected with a custom-made high-bandwidth low-latency interconnect
- Memories are physically distributed
- Nodes often run a microkernel
- Rather blurred line between MPPs and clusters
- Example:
  - Euler (our machine) is a cluster
  - An IBM BG/Q machine is a MPP (because of the interconnect)

Big Iron: Where Are We Today?
[Cntd.]

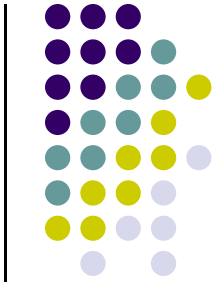




- How is the speed measured to put together the Top500?
  - Basically reports how fast you can solve a dense linear system

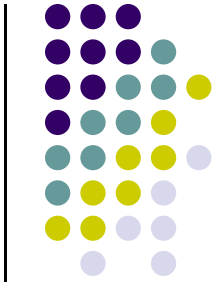


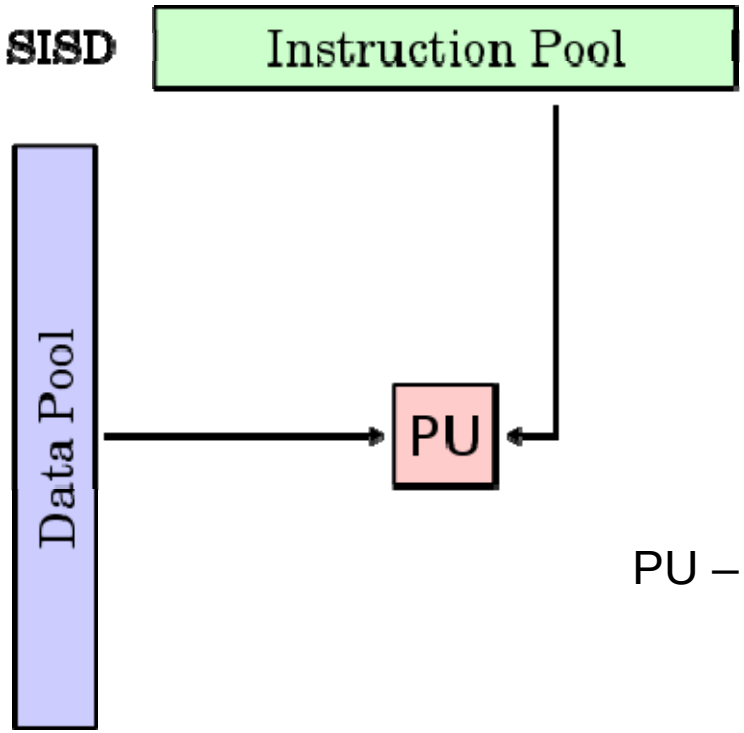
Flynn's Taxonomy of Architectures



- There are several ways to classify architectures (we just saw one based on how memory is organized/accessed)
- Below, classification based on how instructions are executed in relation to data
    - SISD - Single Instruction/Single Data
    - SIMD - Single Instruction/Multiple Data
    - MISD - Multiple Instruction/Single Data
    - MIMD - Multiple Instruction/Multiple Data

Single Instruction/Single Data Architectures





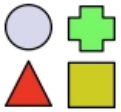
PU – Processing Unit

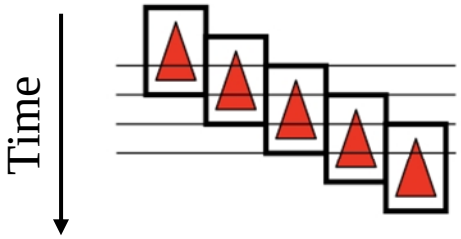
Your desktop, before the spread of dual core CPUs

Slide Source: Wikipedia, Flynn's Taxonomy

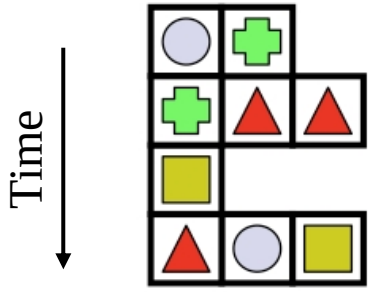
Increasing Throughput of SISD

Instructions:



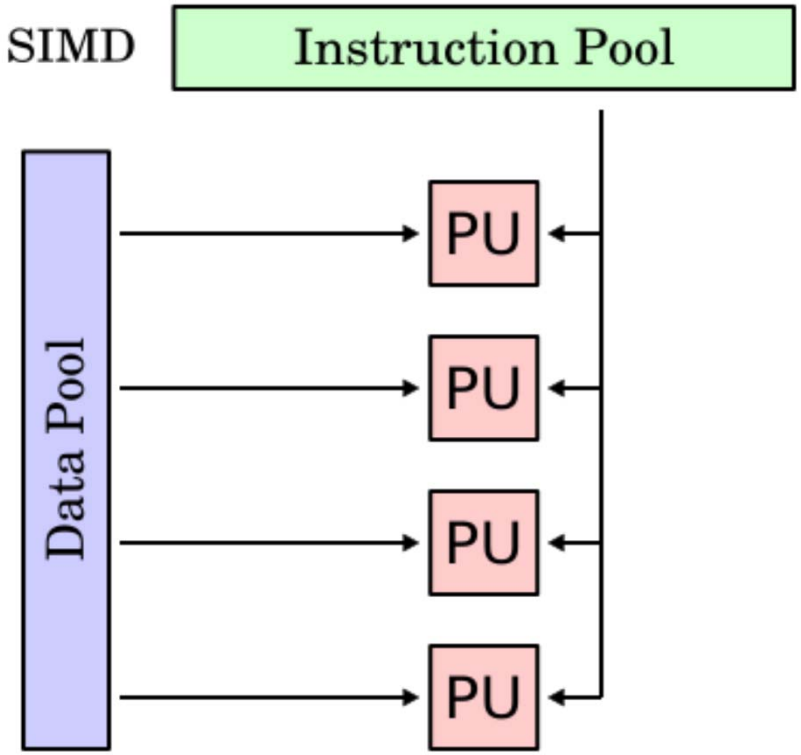


Pipelining



Multiple Issue

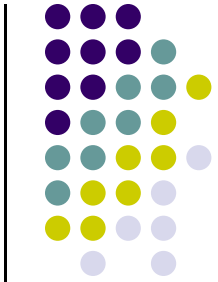
Single Instruction/Multiple Data Architectures



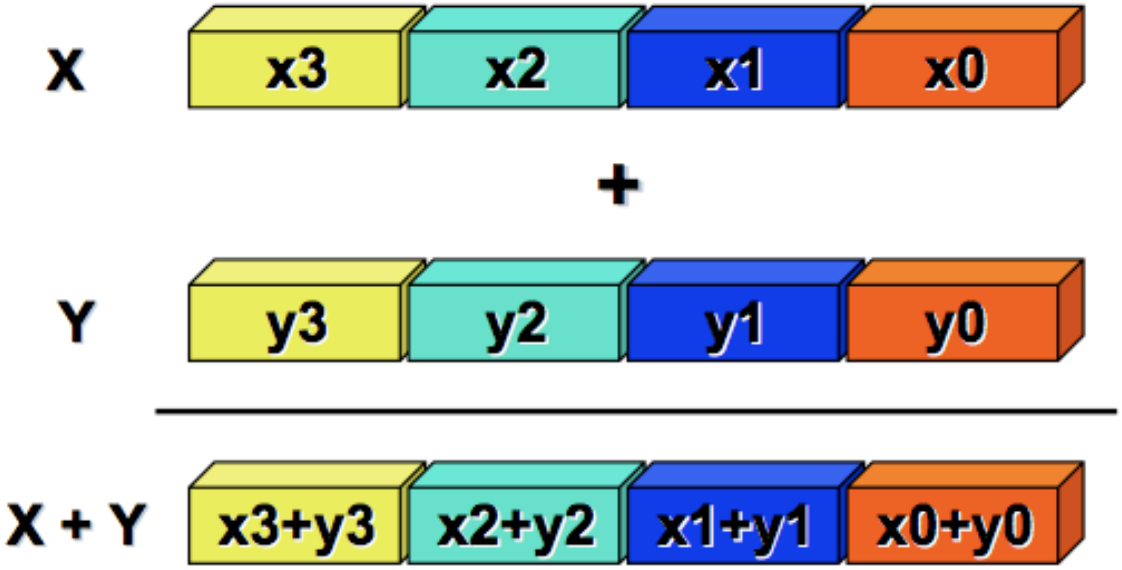
Processors that execute same instruction on multiple pieces of data: NVIDIA GPUs

Slide Source: Wikipedia, Flynn's Taxonomy

Single Instruction/Multiple Data [Cntd.]

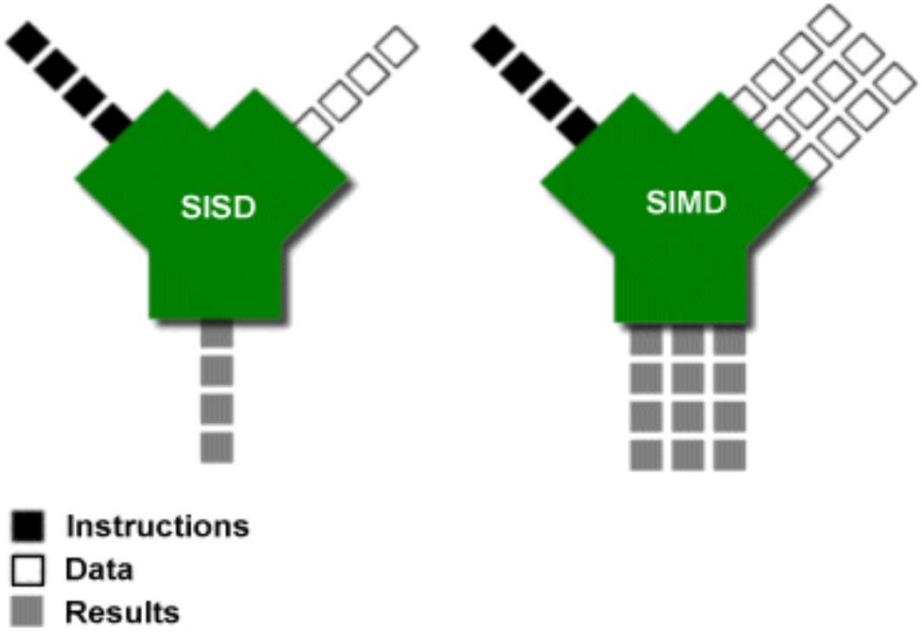


- Each core runs the same set of instructions on different data
- Examples:
  - Graphics Processing Unit (GPU): processes pixels of an image in parallel
  - CRAY's vector processor, see image below



Slide Source: Klimovitski & Macri, Intel

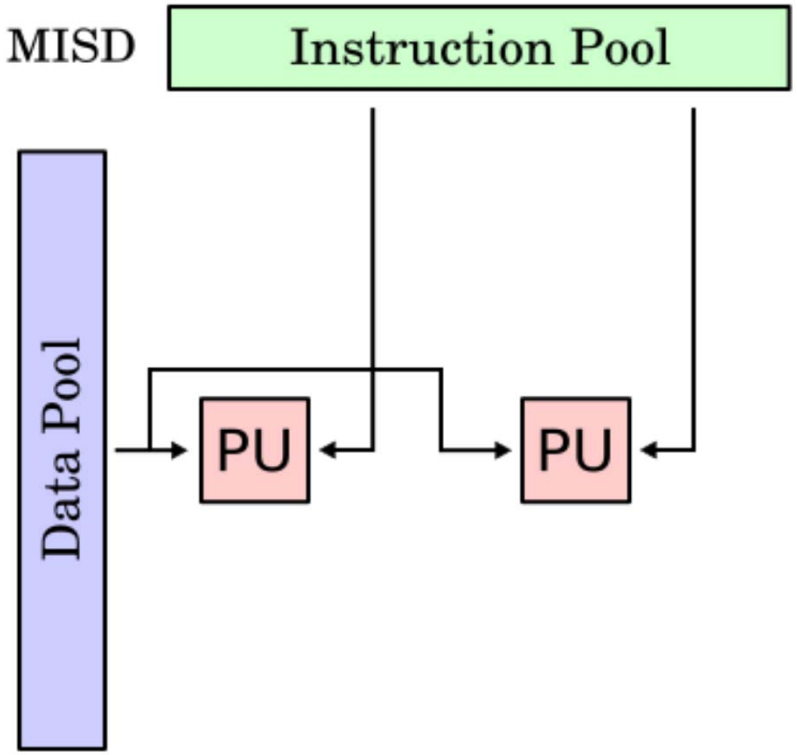
SISD versus SIMD



Writing a compiler for SIMD architectures is difficult
(inter-thread communication complicates the picture...)

Slide Source: ars technica, Peakstream article

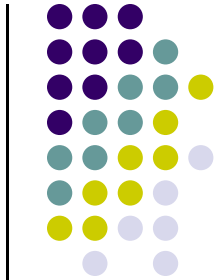
Multiple Instruction/Single Data

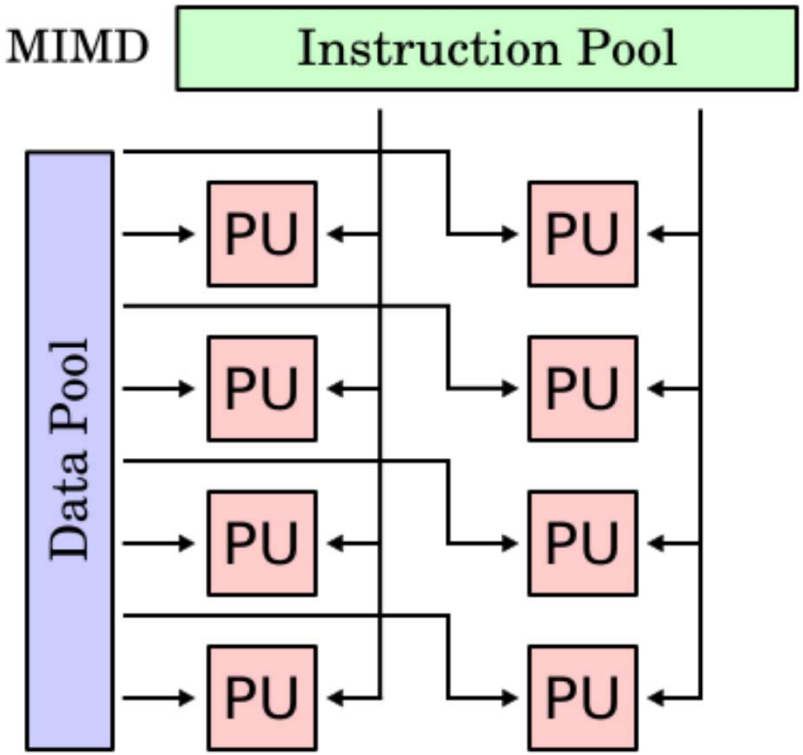


Not useful, not aware of any commercial implementation...

Slide Source: Wikipedia, Flynn's Taxonomy

Multiple Instruction/Multiple Data

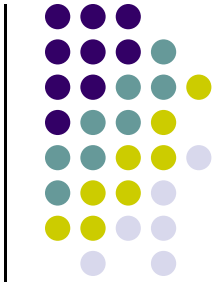




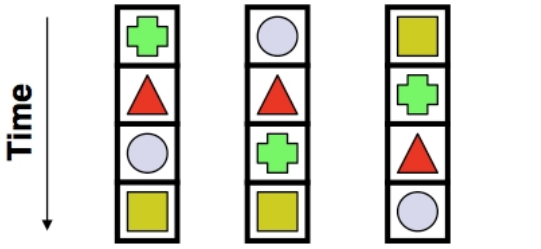
Almost all our desktop/laptop chips are MIMD systems

Slide Source: Wikipedia, Flynn's Taxonomy

Multiple Instruction/Multiple Data

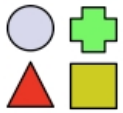


- The sky is the limit: each PU is free to do as it pleases
- Can be of either shared memory or distributed memory categories



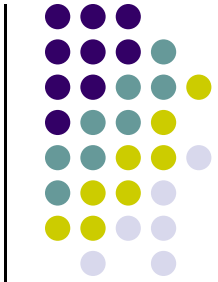
Thread-Level Parallelism (TLP)

Instructions:



Amdahl's Law

Excerpt from “Validity of the single processor approach to achieving large scale computing capabilities,” by Gene M. Amdahl, in Proceedings of the “AFIPS Spring Joint Computer Conference,” pp. 483, 1967

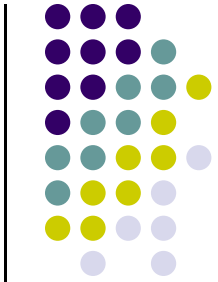


“A fairly obvious conclusion which can be drawn at this point is that the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude”

- Let r_s capture the amount of time that a program spends in components that can only be run sequentially
- Let r_p capture the amount of time spent in those parts of the code that can be parallelized.
- Assume that r_s and r_p are normalized, so that $r_s + r_p = 1$
- Let n be the number of threads used to parallelize the part of the program that can be executed in parallel
- The “best case scenario” speedup S is

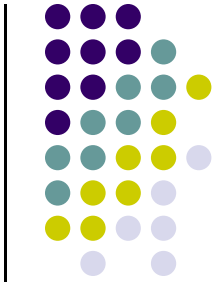
$$S = \frac{T_{old}}{T_{new}} = \frac{r_s + r_p}{r_s + \frac{r_p}{n}} = \frac{1}{r_s + \frac{r_p}{n}}$$

Amdahl's Law
[Cntd.]



- Sometimes called the law of diminishing returns
- In the context of parallel computing used to illustrate how going parallel with a part of your code is going to lead to overall speedups
- The art is to find for the same problem an algorithm that has a large r_p
  - Sometimes requires a completely different angle of approach for a solution
- Nomenclature
  - Algorithms for which r_p=1 are called "embarrassingly parallel"

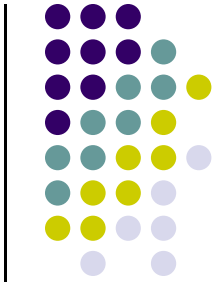
Example: Amdahl's Law



- Suppose that a program spends 60% of its time in I/O operations, pre and post-processing
- The rest of 40% is spent on computation, most of which can be parallelized
- Assume that you buy a multicore chip and can throw 6 parallel threads at this problem. What is the maximum amount of speedup that you can expect given this investment?
- Asymptotically, what is the maximum speedup that you can ever hope for?

A Word on “Scaling”

[important to understand]

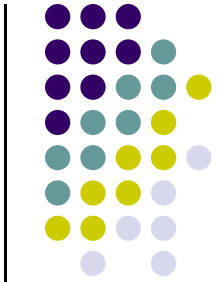


- **Algorithmic Scaling** of a solution algorithm
  - You only have a mathematical solution algorithm at this point
  - Refers to how the effort required by the solution algorithm scales with the size of the problem
  - Examples:
    - Naïve implementation of the N-body problem scales like $O(N^2)$, where N is the number of bodies
    - Sophisticated algorithms scale like O(N logN)
    - Gauss elimination scales like the cube of the number of unknowns in your linear system

- **Implementation Scaling** of a solution on a certain architecture
  - **Intrinsic Scaling**: how the execution time changes with an increase in the size of the problem
  - **Strong Scaling**: how the execution time changes when you increase the processing resources
  - **Weak Scaling**: how the execution time changes when you increase the problem size but also the processing resources in a way that basically keeps the ratio of problem size/processor constant

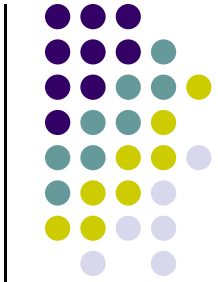
A Word on “Scaling”

[important to understand]



- Two follow up comments

1. Worry about this: Is the Intrinsic Scaling similar to the Algorithmic Scaling?
   - If Intrinsic Scaling significantly worse than Algorithmic Scaling you then probably memory transactions are dominating the implementation
2. If the problem doesn’t scale can be an interplay of several factors
   - The intrinsic nature of the problem at hand
   - The attributes (organization) of the underlying hardware
   - The algorithm used to solve the problem; i.e., the amount of parallelism it exposes



End: Intro Part

Beginning: GPU Computing, CUDA Programming Model