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

Big Iron HPC

Amdahl's Law

Parallel Computing on the GPU

September 25, 2015

Quote of the Day

"A nickel ain't worth a dime anymore."

Yogi Berra (1925 – Sept. 22, 2015)

Before We Get Started

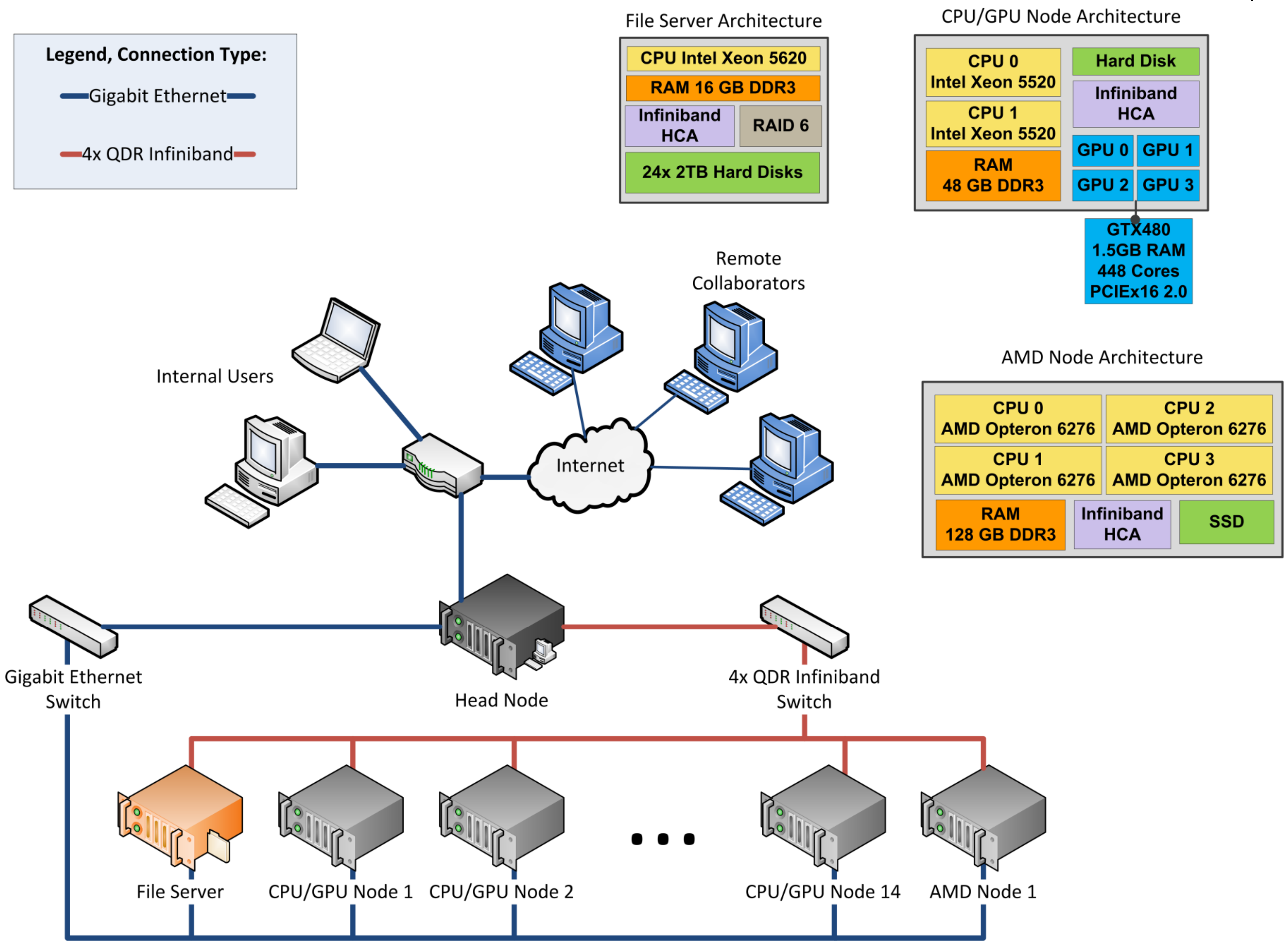
- Issues covered last time:
  - Three walls to sequential computing: memory wall, ILP wall, power wall
  - Moore’s law still holding
    - Many transistors $\rightarrow$ an opportunity to organize it as multiple cores, leads to parallel computing
  - Moore’s law not without issues (Dennard scaling was the secret sauce)

- Today’s topics
  - Big Iron HPC
  - Amdahl's Law
  - Start parallel computing on GPU cards

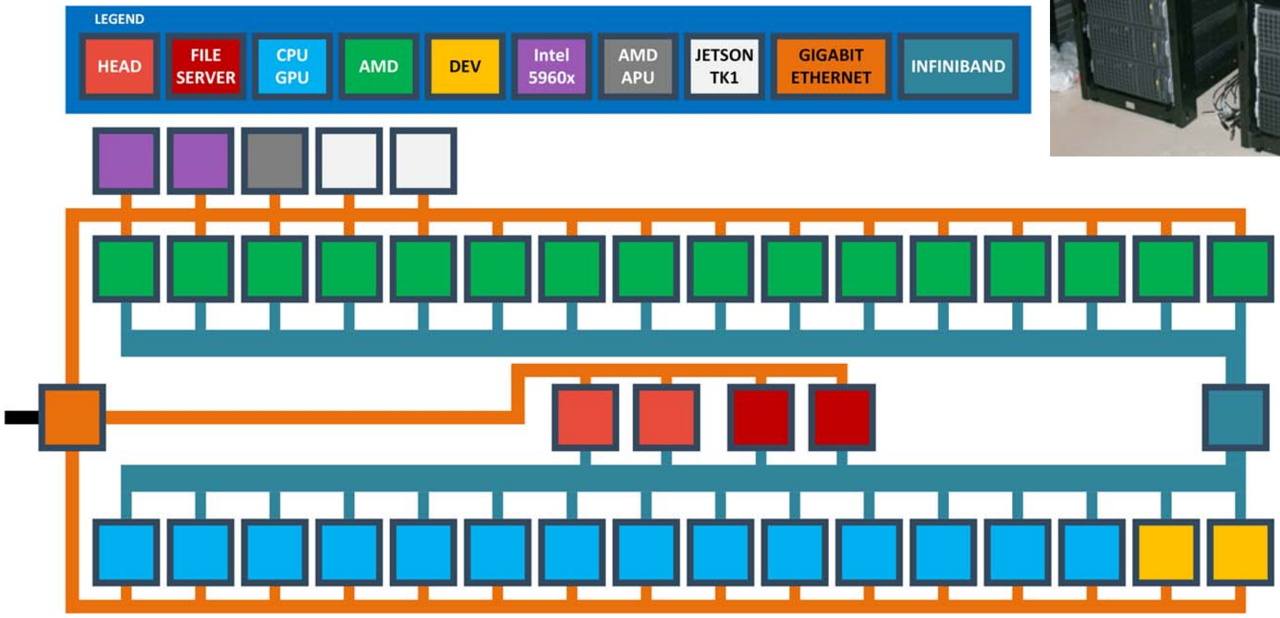
- Assignment:
  - HW03 –due on September 30 at 11:59 PM

“Big Iron” Parallel Computing

Euler: CPU/GPU Heterogeneous Cluster ~ Hardware Configuration ~

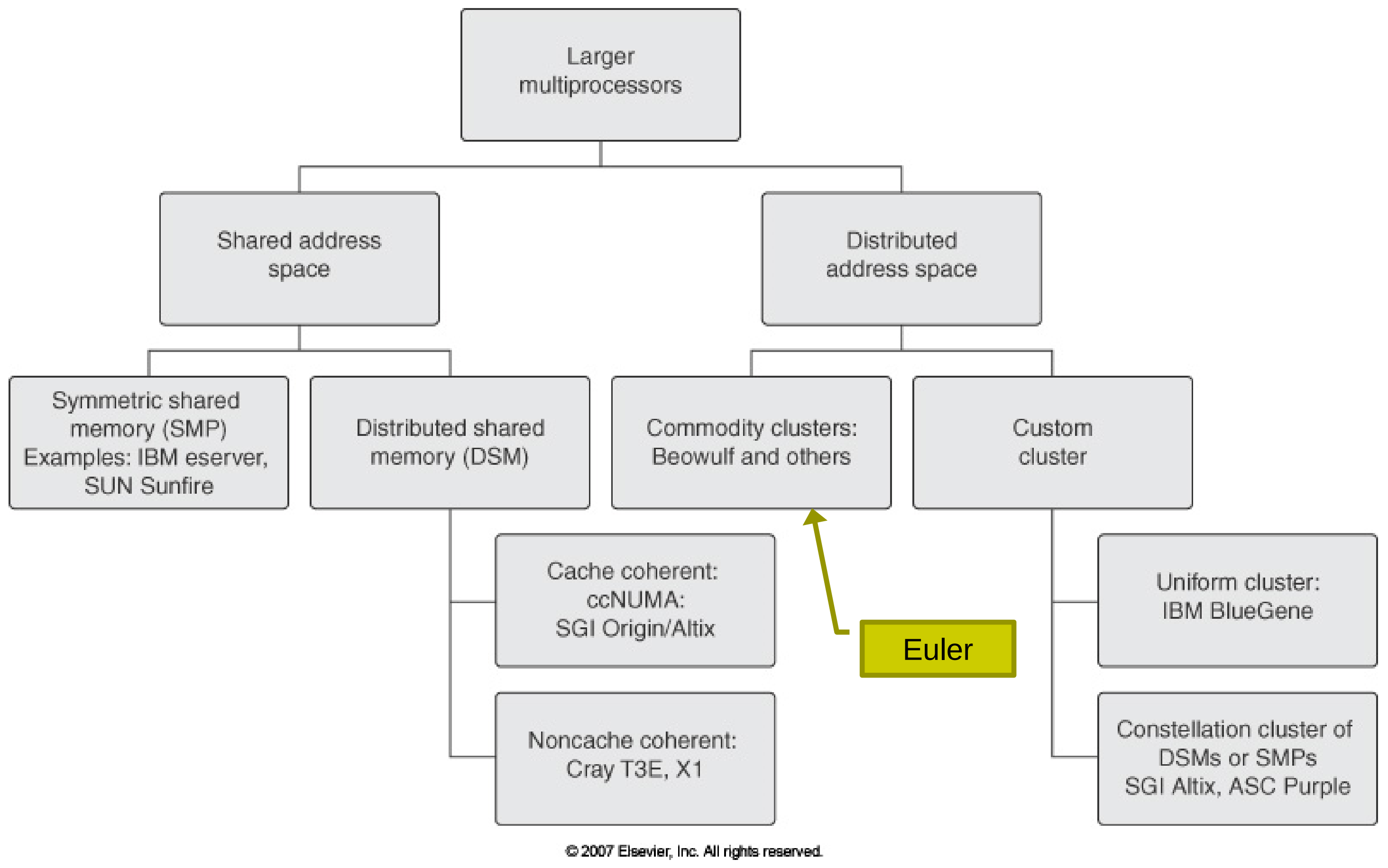


Background: Lab's Research Cluster



EULER - Heterogeneous Research Cluster.

Overview of Large Multiprocessor Hardware Configurations (“Big Iron”)


Larger multiprocessors
Shared address space
Distributed address space
Symmetric shared memory (SMP) Examples: IBM eserver, SUN Sunfire
Distributed shared memory (DSM)
Commodity clusters: Beowulf and others
Custom cluster
Cache coherent: ccNUMA: SGI Origin/Altix
Noncache coherent: Cray T3E, X1
Uniform cluster: IBM BlueGene
Constellation cluster of DSMs or SMPs SGI Altix, ASC Purple
Euler
© 2007 Elsevier, Inc. All rights reserved.

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

Nomenclature Issues

- 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

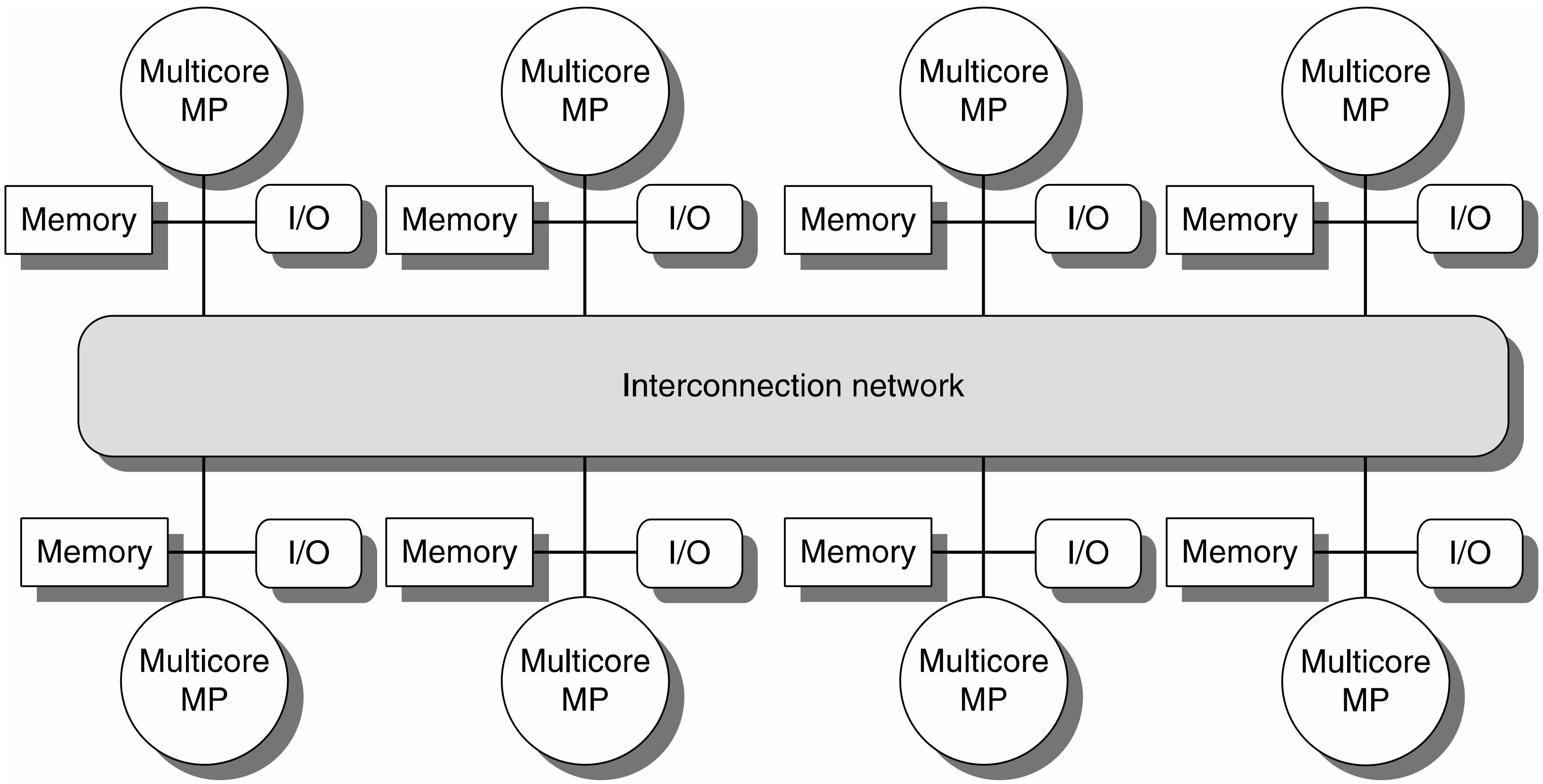
- Distributed addressed space: the opposite of the above

- 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). Raises 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 (MP) architecture
  - Euler, for instance



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

Comments, distributed-memory multiprocessor architecture

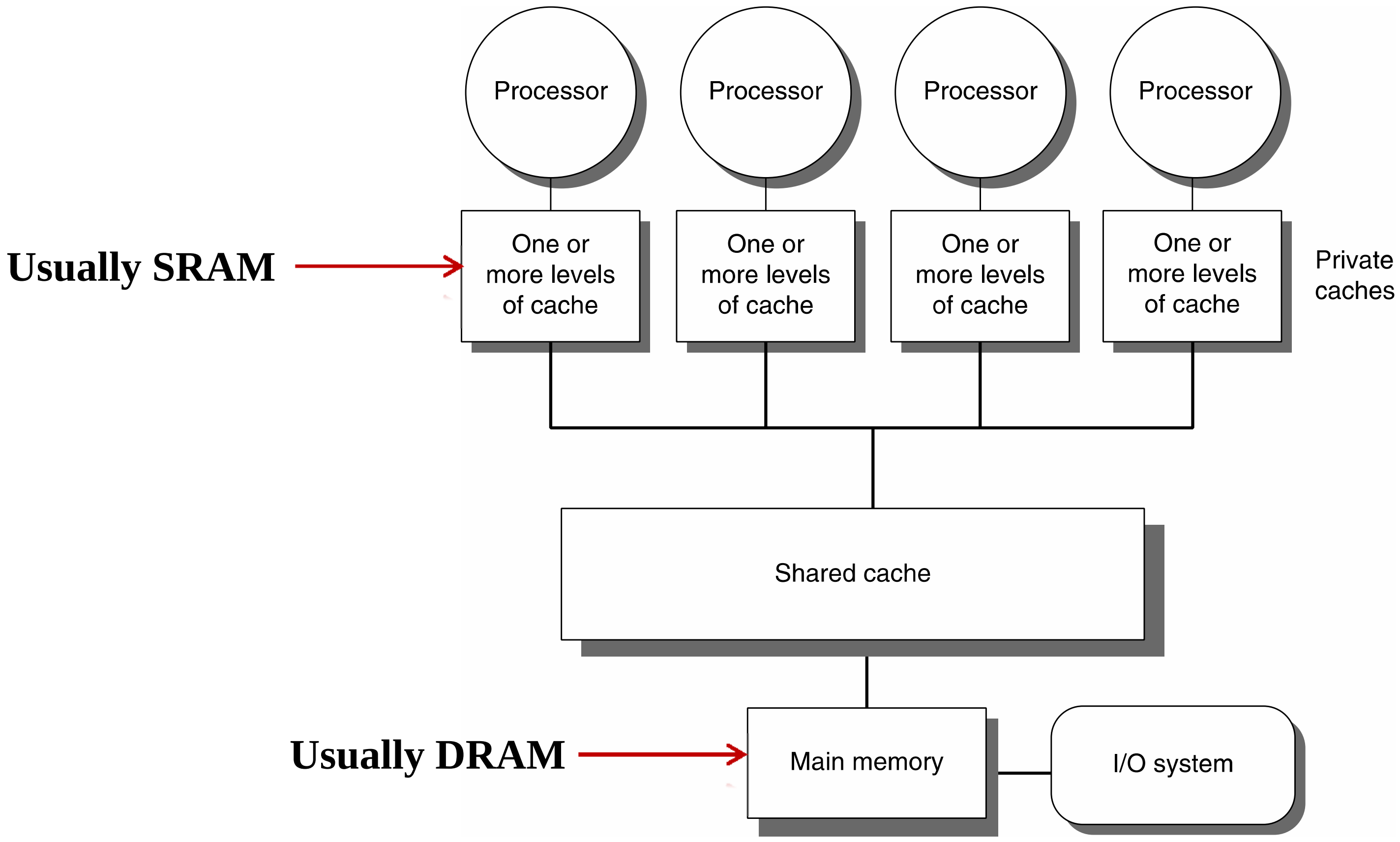
- Basic architecture consists of nodes containing a processor, some memory, typically some I/O, and an interface to an interconnection network that connects all nodes

- 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); i.e., high
  - Requires special network cards: HCA – “Host Channel Adaptor”

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

- Shared-Memory Multiprocessor Architecture


Processor
Processor
Processor
Processor
Usually SRAM
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Private caches
Shared cache
Usually DRAM
Main memory
I/O system

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

Comments, SMP Architecture

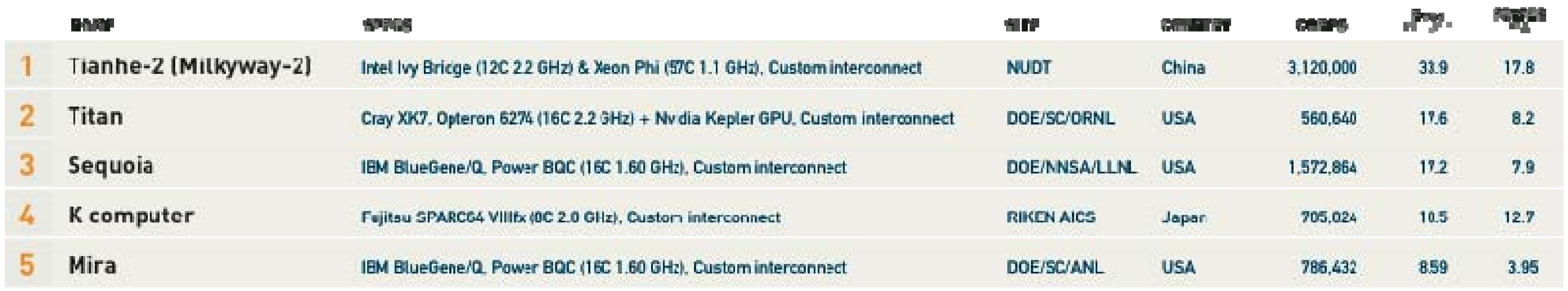
- Multiple processor-cache subsystems share the same physical off-chip memory
- Typically connected to this off-chip memory by one or more buses or a switch
- Key architectural property: uniform memory access (UMA) time to all of memory from all the processors
  - This is why it's called symmetric

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)

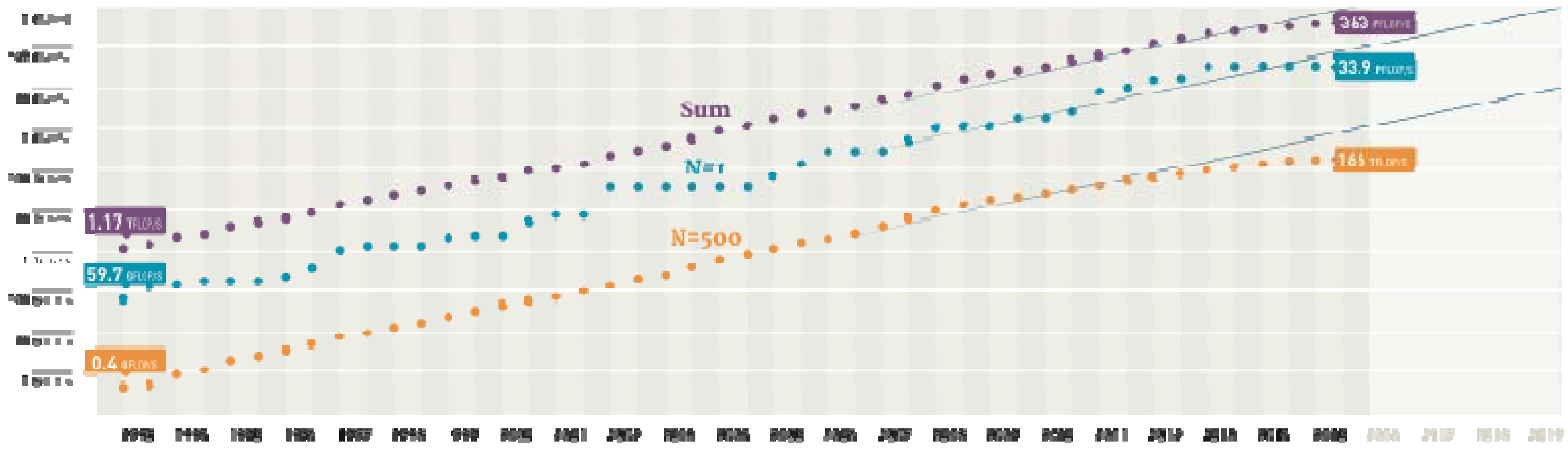
Big Iron: Where Are We Today?

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

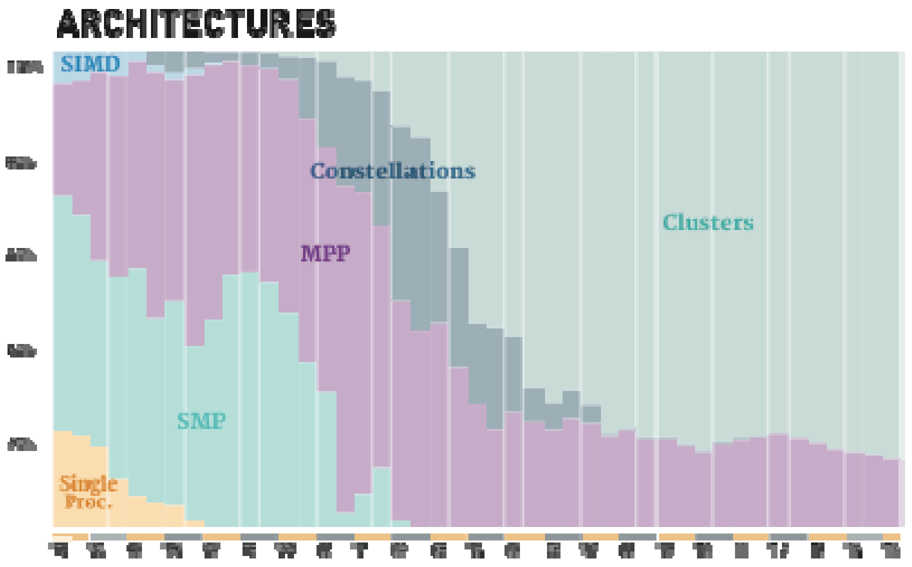


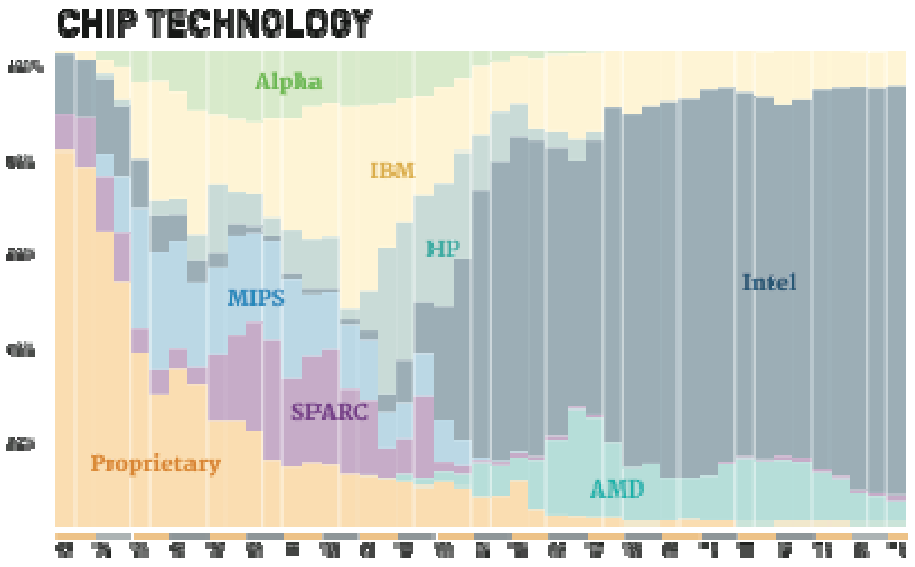
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
1	Tianhe-2 (Milkyway-2)	Intel Ivy Bricge (12C 2.2 GHz) & Xeon Phi (57C 1.1 GHz), Custom interconnect	NUDT	China	3,120,000	33.9	17.8
2	Titan	Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvdia Kepler GPU, Custom interconnect	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

PERFORMANCE DEVELOPMENT



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





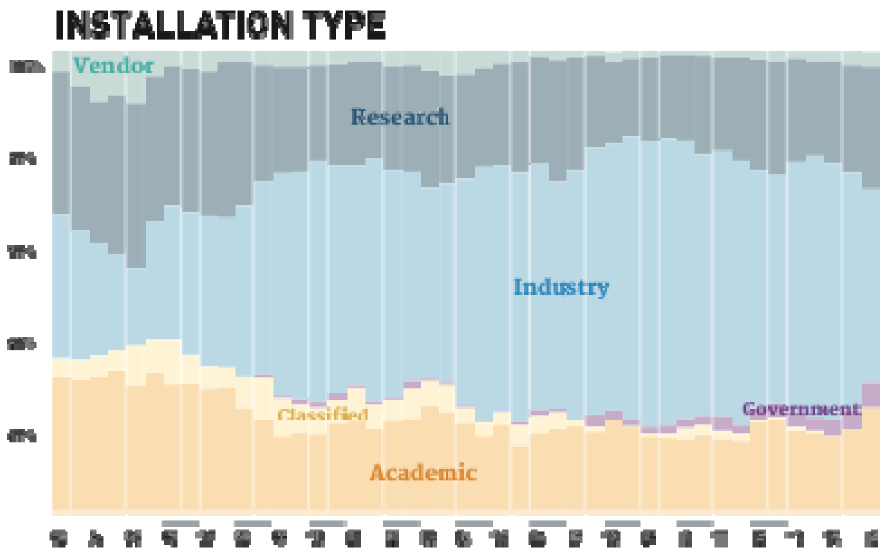
- Abbreviations/Nomenclature
  - MPP – Massively Parallel Processing
  - Constellation – subclass of cluster architecture envisioned to capitalize on data locality
  - MIPS – “Microprocessor without Interlocked Pipeline Stages”, a chip design of the MIPS Computer Systems of Sunnyvale, California
  - 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)

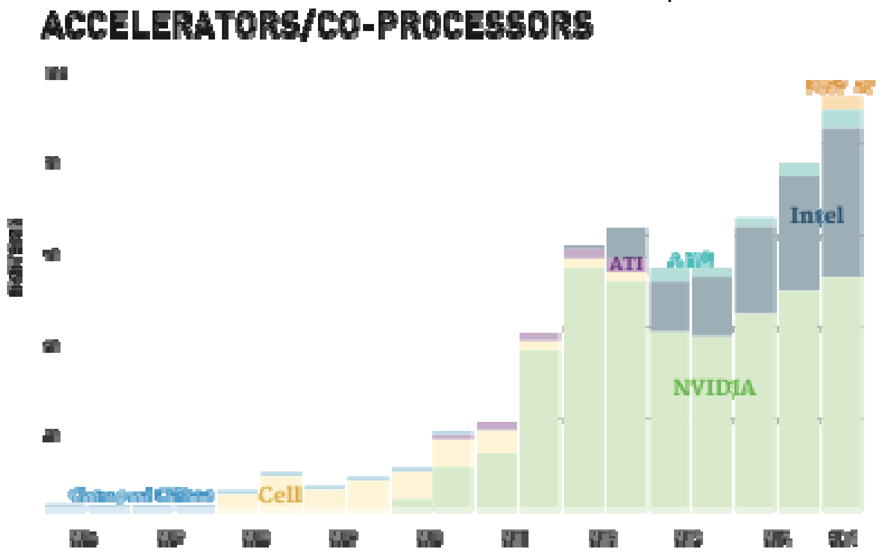
Short Digression: Massively Parallel Processing

What is a MPP?

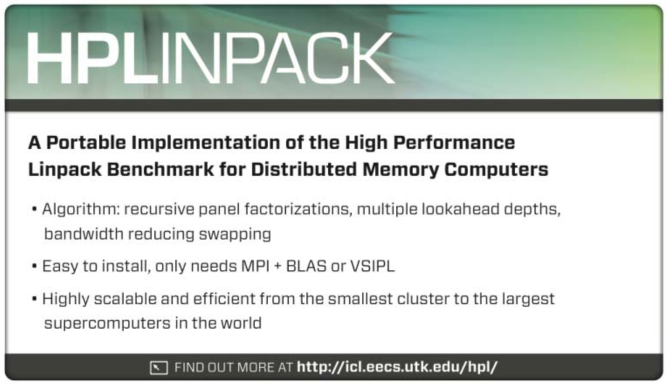
- A very large-scale computing system with commodity processing nodes interconnected with a custom-made high-bandwidth low-latency interconnect
- Memories are physically distributed
- Nodes often run a microkernel
- Rather blurred line between MPPs and clusters
- Example:
  - Euler (our machine) is a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



HPLINPACK

A Portable Implementation of the High Performance Linpack Benchmark for Distributed Memory Computers

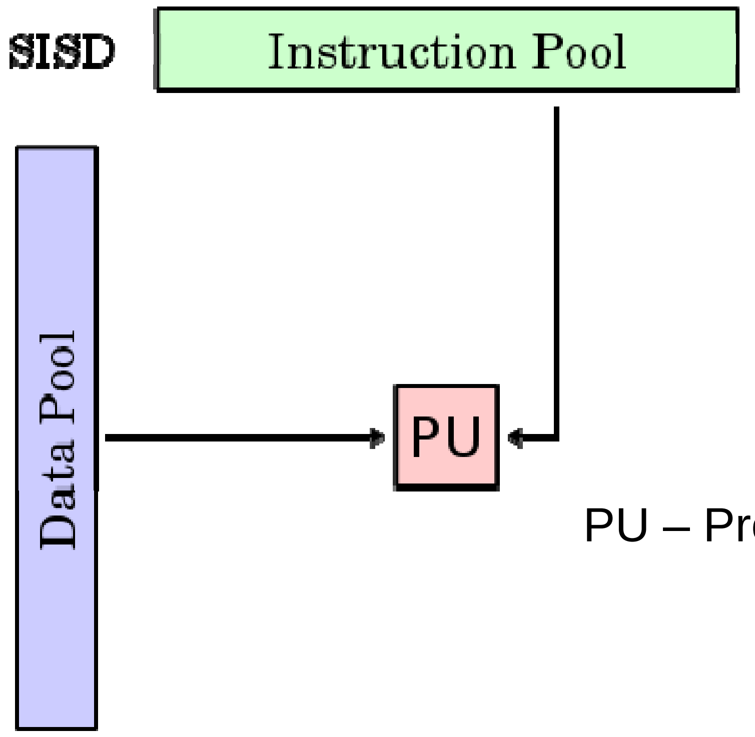
- Algorithm: recursive panel factorizations, multiple lookahead depths, bandwidth reducing swapping
- Easy to install, only needs MPI + BLAS or VSIPL
- Highly scalable and efficient from the smallest cluster to the largest supercomputers in the world

FIND OUT MORE AT **http://icl.eecs.utk.edu/hpl/**

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



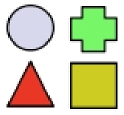
PU – Processing Unit

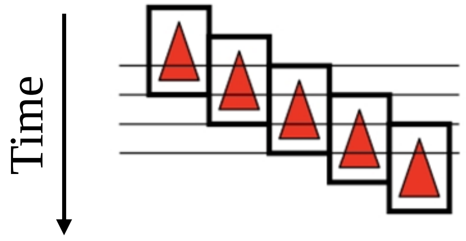
Your desktop, before the spread of dual core CPUs

Slide Source: Wikipedia, Flynn's Taxonomy

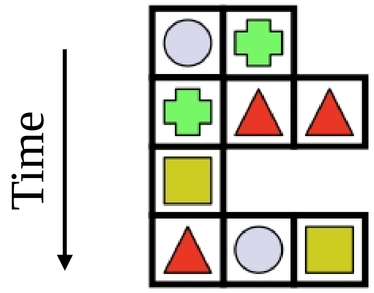
Increasing Throughput of SISD

Instructions:



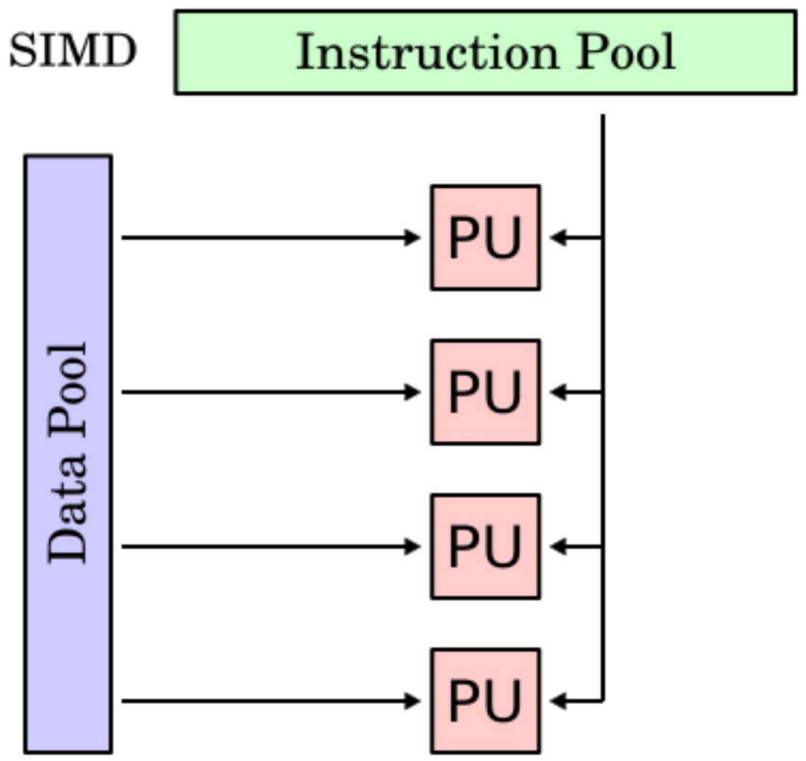


Pipelining



Multiple Issue

Single Instruction/Multiple Data Architectures

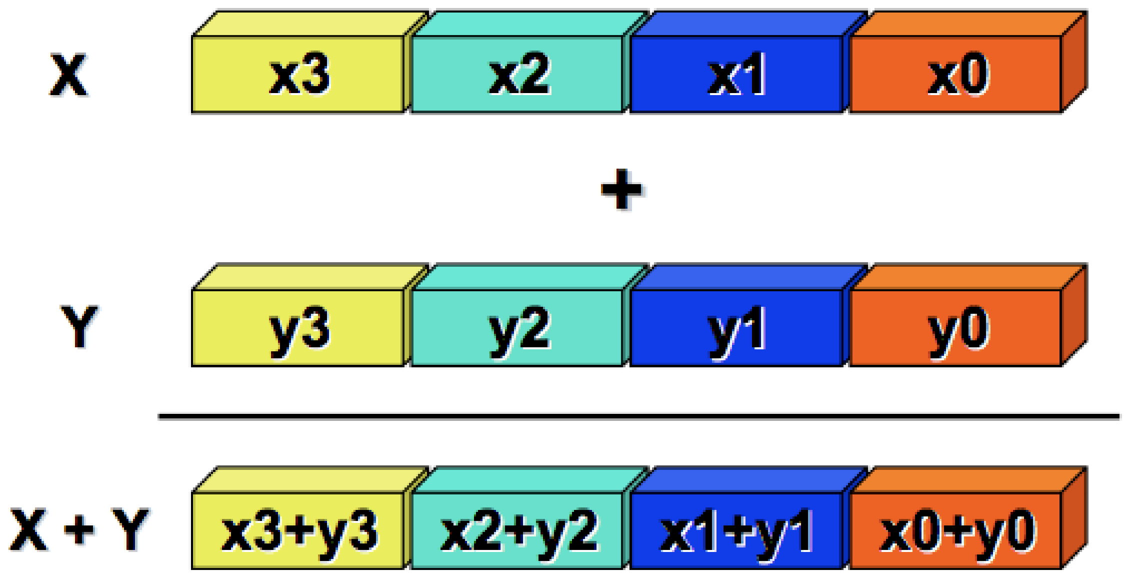


Processors that execute same instruction on multiple pieces of data: NVIDIA GPUs

Slide Source: Wikipedia, Flynn's Taxonomy

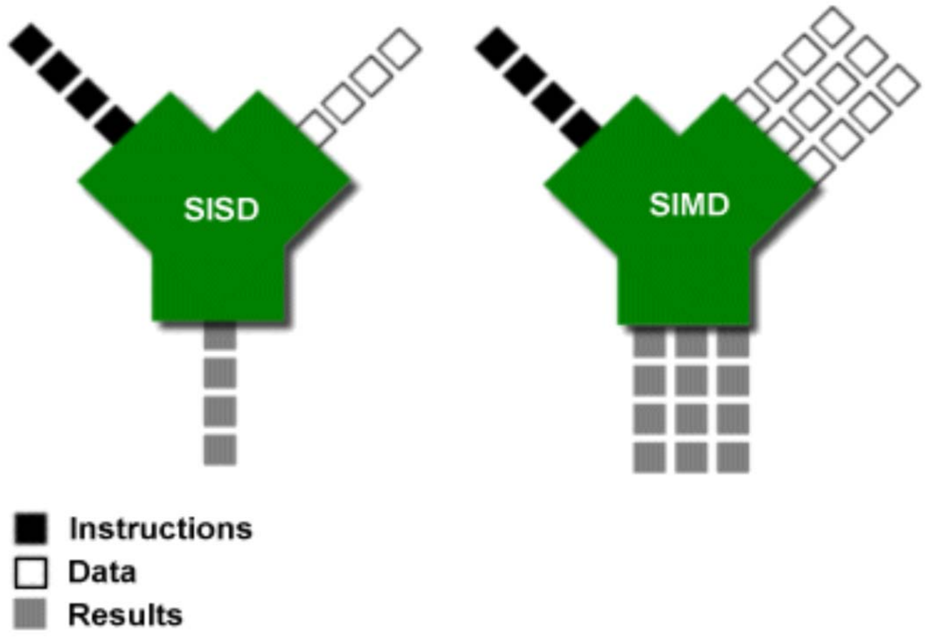
Single Instruction/Multiple Data [Cntd.]

- 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

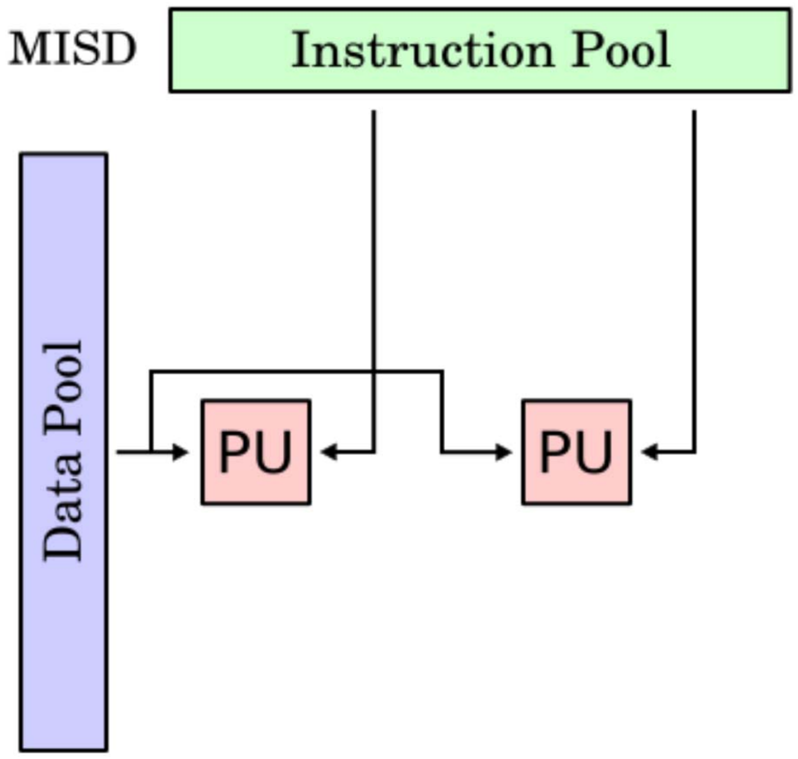
SISD versus SIMD



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

Slide Source: ars technica, Peakstream article

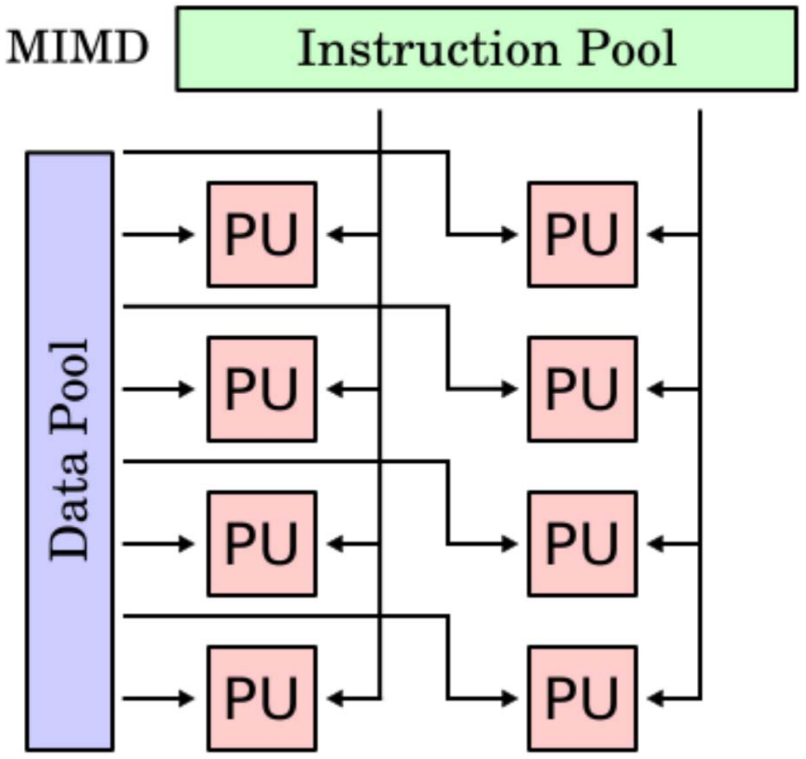
Multiple Instruction/Single Data



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

Slide Source: Wikipedia, Flynn's Taxonomy

Multiple Instruction/Multiple Data

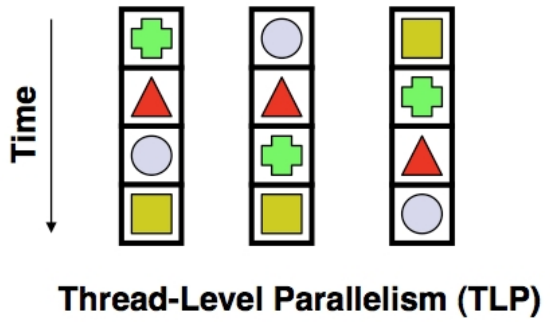


Almost all our desktop/laptop chips are MIMD systems

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



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 (mathematical) Scaling** of an algorithm
  - Refers to how the number of calculations required by the algorithm scales with 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 $O(N \log N)$
    - 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
    - You have strong scaling if you keep doubling the amount of processors only to see a halving of the run time
  - **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 ration of problem size/processor constant
    - You have weak scaling if when you keep doubling the size of the problem, doubling the amount of processors and the run time stays the same

  - NOTE: Strong scaling is harder to get than weak scaling

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 algorithm used to solve the problem; i.e., the amount of parallelism it exposes
   - The attributes (organization) of the underlying hardware: you’re not using the right HW