Scientific Computing and Scientific Software Infrastructure - High Performance Computing (HPC)

3 April 2025

Oliver Gutsche - Fermilab
Zhiling Lan - University of Illinois Chicago & Argonne National Lab (joint)
Michael E. Papka - Argonne National Laboratory and University of Illinois Chicago

Supercomputing or High-Performance Computing

UIC UNIVERSITY OF ILLINOIS CHICAGO

> *No other single technology in the history of humanity has experienced a similar rate of growth, even in its relatively short existence. Within the span of a single human lifetime, supercomputers have* ***expanded their ability to perform calculations by a factor of 10 trillion*** *or 13 orders of magnitude, and this is a conservative estimate. From less than a 1000 basic operations per second in the late 1940s to today's performance in excess of a 100 quadrillion floating-point operations per second (over 100 petaflops), supercomputer speed has steadily* ***improved by about a factor of 200 times every decade*** *through a series of advances in technology, architecture, programming methods, algorithms, and system software.*
>
> — Thomas Sterling, Matthew Anderson, Maciej Brodowicz

- 1 trillion ≡ 1,000,000,000,000
- 200x means exascale in 2020 timeframe (that is a billion billion calculations per second)

High Performance Computing (HPC)

High Performance Computing: Nomenclature and Definitions

UNIVERSITY OF ILLINOIS CHICAGO

- High Performance Computing $\rightarrow$ HPC
- High Performance Computing $\equiv$ Supercomputing
- High-performance computing (HPC) uses supercomputers and computer clusters to solve advanced computation problems.[1]

- **Capacity Computing** also referred to as *throughput computing*
  - Job stream parallelism: a form of weak scaling
  - No interrelationship between independent tasks, that is no inter-synchronization, no data exchange
  - Performance measured by total floats across jobs normalized to time
- **Capability Computing**
  - Single job, fixed-size data set: strong scaling
  - Tightly coupled, strong interrelationship among parallel actions
  - Performance achieved through reduction of time to solution

[1] Wikipedia

High Performance Computing Disciplines

HPC is actually a collection of disciplines ...

- hardware
  - system level - how things integrate
  - component level (processor, I/O, network, ...)
- software
  - operating system
  - libraries
  - tools
  - applications (alternatively **workloads**)

High Performance Computing Systems

- In many ways HPC systems match the basic functionality of a desktop or laptop computer (both in hardware and software)
- What distinguishes an HPC system from a conventional computer
  - Interconnectivity
  - Scale of the component resources
  - Software to manage the operation of the system at that scale

  Here **scale** means physical (hardware) and logical (software) parallelism beyond what is typically seen in a desktop or laptop

Image: The IBM Blue Gene/P supercomputer *Intrepid* at Argonne National Laboratory runs **164,000 processor cores** using normal data center air conditioning, grouped in **40 racks/cabinets** connected by a **high-speed 3D torus network**

Summary of History

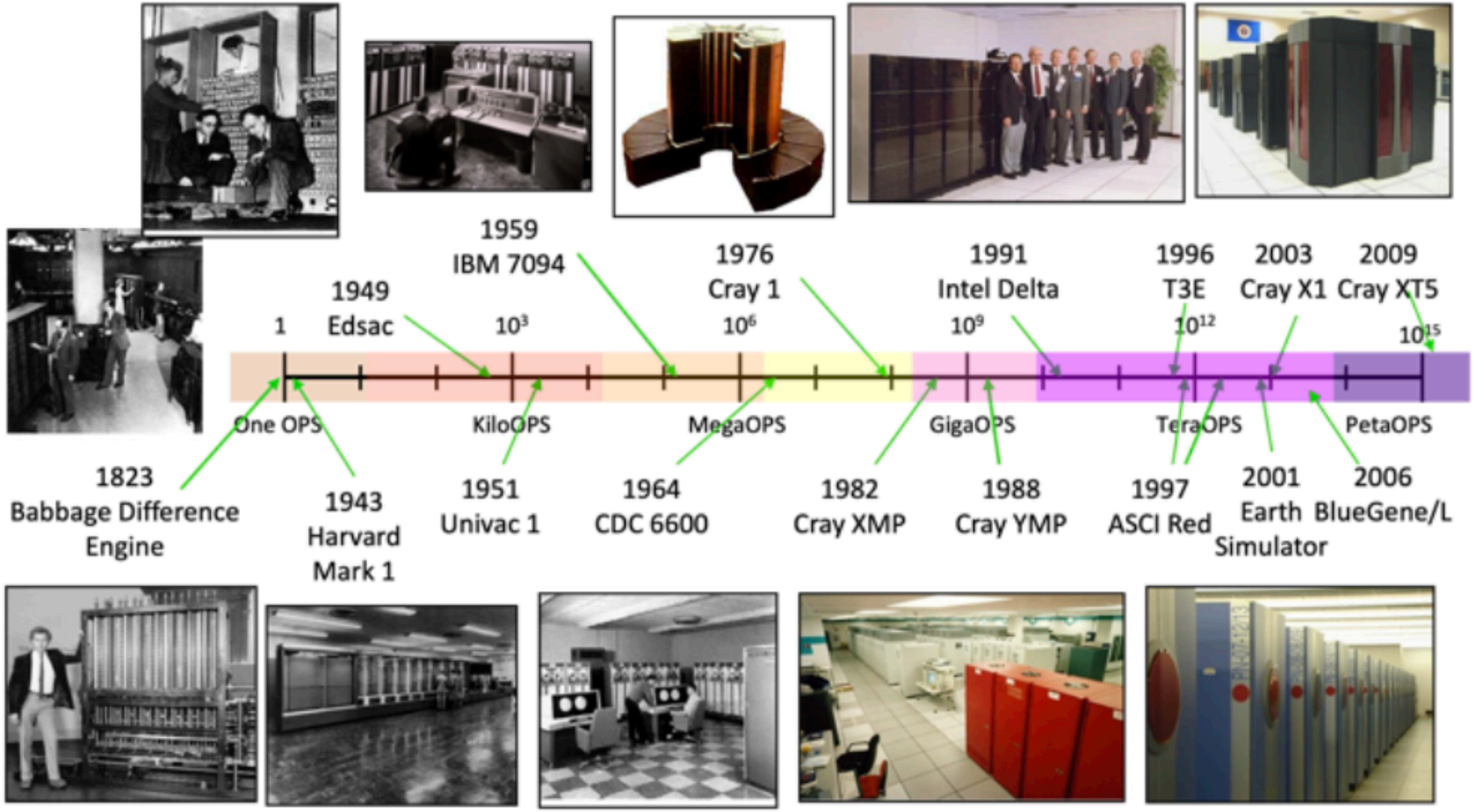


Image: Timeline created by Thomas Sterling

A Brief History of Supercomputing

- In a single lifetime, the capability of supercomputers has gained a growth factor of more than 10 trillion (10,000,000,000,000)
- HPC has seen performance gains at the rate of 200x each decade
- Changes in architecture enabled by new emergent technologies (vacuum tubes to transistors, spinning disk to solid state drives)
- Seven major periods mark HPC history (textbook)
  - Automated calculators through mechanical technologies
  - von Neumann architecture in vacuum tubes
  - Instruction-level parallelism
  - Vector processing and integration
  - Single-instruction multiple data array
  - Communicating sequential processors and very large-scale integration
  - Multicore petaflops

Von Neumann Architecture

von Neumann architecture describes a design architecture for an electronic digital computer with these components:

- A processing unit with both an arithmetic logic unit and processor registers
- A control unit that includes an instruction register and a program counter
- Memory that stores data and instructions
- External mass storage
- Input and output mechanisms

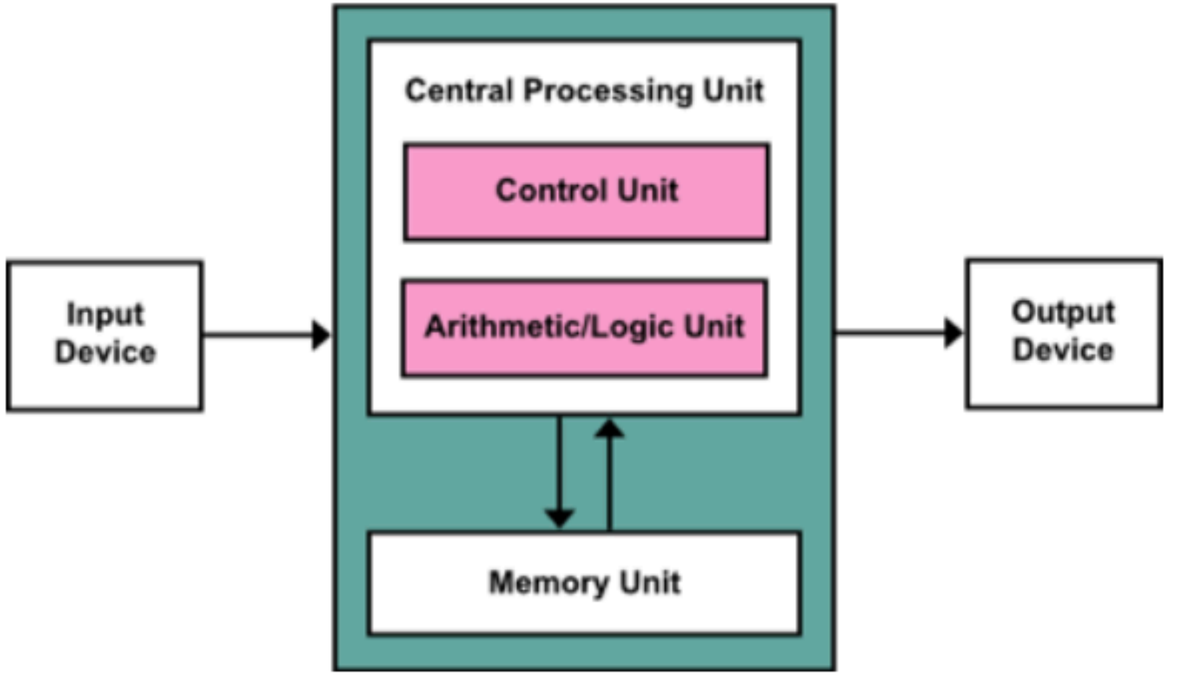


Image: von Neumann Architecture from wikipedia

Epoch VI: CSP (1990 - 2005)

- **Enabling Technology**
  - VLSI (Very Large Scale Integration)
  - 32-bit/64-bit Microprocessor (the Killer Micro)
  - System Area Network (SAN)
- **Fundamental Concepts**
  - Communicating Sequential Processing (CSP)
  - Distributed memory, message passing
  - Bulk Synchronous Parallel (BSP)
- **Accomplishments**
  - Commodity clusters and massively parallel processors (MPPs)
  - Dominates for two decades
  - From 10s of Gigaflops to Teraflops to Petaflops performance

Intel Delta

Beowulf cluster at NASA

Communicating Sequential Processors and VLSI

- Two strategies emerged for the exploitation of VLSI technology: the development of massively parallel processors (MPPs) and the creation of commodity clusters

Images: The Intel Paragon at Computer History Museum (left) and Thomas Sterling in front of NASA Beowulf system (left)

Epoch VII: Multicore/Heterogeneous (2005 - ?)

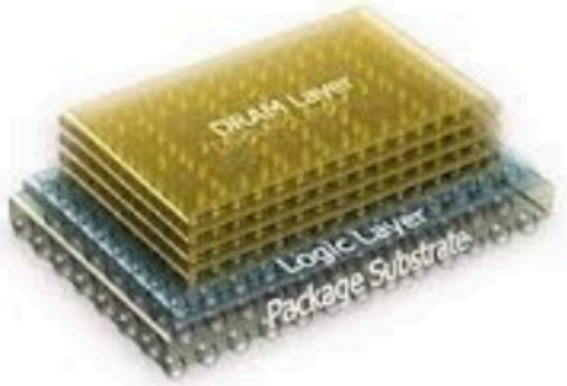
- **Enabling Technology**
  - Multi-core
- **Fundamental Concepts**
  - Hybrid execution
  - Heavy-weight nodes
- **Accomplishments**
  - Fastest computers in the world
  - Improved energy efficiency
  - Hybrid programming interfaces

Titan supercomputer at Oak Ridge National Laboratory

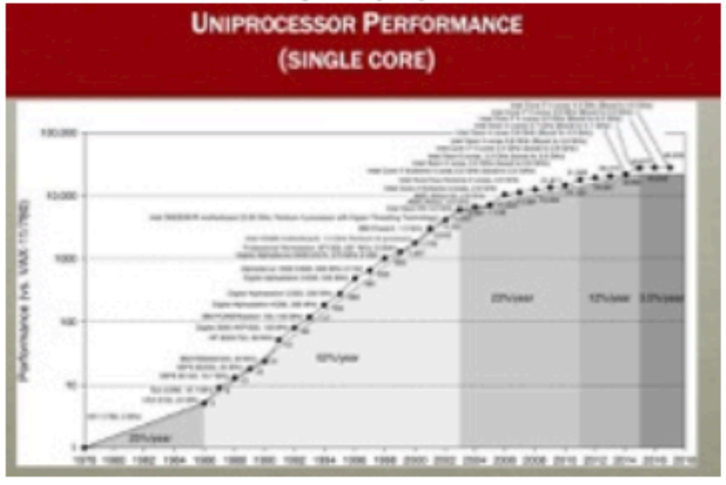
Tianhe-2 supercomputer in China

Epoch VIII: Exascale (2020 - ?)

- **Enabling Technology**
  - 3-D die stacking
  - Optical inter/intra socket networking
  - End of Moore's Law
  - Processor-in-Memory (PIM)
- **Fundamental Concepts**
  - Innovative execution models
  - Dynamic adaptive resource management
  - Message-driven computation
  - Multi-threading
- **Accomplishments**
  - Billion-way parallelism
  - Energy <20pJ/op
  - Runtime system software



3D die



End of Moore's Law

HPC: Key Properties

Key Properties of HPC Architecture

- Three key properties determine performance
  - Speed
  - Parallelism
  - Efficiency

$$P = e \times S \times a(R) \times \mu(E)$$

where P is the average performance, e is the efficiency, S is the scaling (the number of units that can operate at the same time), a function of reliability R (how often something fails) and μ is the clock rate which is a function of power E

- and they are influenced by
  - Power
  - Reliability
  - Programmability

Key Properties: Speed

- HPC system performance is directly related to the speed of its components, with a key parameter being the clock rate of its processor cores
- Different technologies have widely differing speeds or cycle times, and architecture must match these disparate speeds
- A modern architecture will include a memory hierarchy consisting of a mix of slower higher-density DRAMs for capacity with faster low-density SRAMs called *caches* to achieve speed
- The rate at which data can be transferred or communicated between any two points within the system is also important, with two measures applied: bandwidth and latency
- Architecture must balance these different time constants in structures and through methods to achieve the overall best-delivered performance for a user workload within normalizing cost factors like nonrecurring engineering (NRE), deployment, power, and user productivity

Key Properties: Parallelism

- No matter how fast the speed of the parts technology can be, it will never be fast enough to deliver the necessary performance required by major application problems
- HPC architecture is heavily dependent on structures that permit many actions to occur simultaneously; this is referred to as ***parallelism***
- The many different classes of parallel computer architecture are defined and distinguished by the diverse structures employed to achieve parallelism in different ways
- HPC architecture is also determined by how much parallelism is controlled; both the data path and the control path are factors in how an HPC architecture exploits parallelism

Key Properties: Efficiency

- Efficiency determines the delivered performance for a user workload
- A common measure of efficiency in HPC is the ratio of the sustained floating-point performance to the theoretical peak floating-point performance

$$e_{flops} = \frac{P_{sustained}}{P_{peak}}$$

where e_{flops} is the floating-point efficiency, such that $0 \leq e_{flops} \leq P_{peak}$, where P_{peak} is the theoretical peak performance of the HPC architecture in flops, and $P_{sustained}$ is what is achieved on average

- The efficiency measure reflects an earlier era when a floating-point operation was expensive; today, data movement and the costs of data access from memory are far more significant in both time and energy than a register-to-register floating-point operation
- Other factors are becoming increasingly important in determining the efficiency of HPC systems

Key Properties: Power

- Every computer uses electrical power for its operation, and the speed of processor cores is in part proportional to the power applied
- Electricity is required to drive the integrated circuits, interconnection channels, and input/output (I/O) devices as well as removing the resulting heat from the system
- Thermal control is essential to ensure the system is not overheating
- Active thermal control is becoming increasingly important; this leads to increased measurement of temperatures throughout the system[2] and monitoring of thermal gradients across the system
- Modern multicore processors have a variety of knobs to tune from variable clock rates, voltage adjustment, and numbers of active cores

[2] 16*K* per compute rack on Aurora

Key Properties: Reliability

- Reliability of HPC systems is compounded by their scale; errors can occur periodically due to hardware or software faults
- **Hard** faults occur when some part of the hardware breaks permanently, and **soft** faults occur when a part intermittently fails but otherwise operates correctly
- Software errors are due to flawed coding of either the user application program or the supporting system software (debugging is a part of the task of application development)
- One mitigation approach is **checkpoint/restart**, which periodically stores the program state at a point in time
- Root-cause analysis is required to establish the cause and possible source of the error before execution proceeds

Key Properties: Programmability

- How difficult it is to write or develop application code reflects the programmability of the system
- Programmability is affected by factors such as processor core and system architectures, programming models and facility of the language, effectiveness of the system software, and the skills of the programmers themselves
- The level of effort required to write the application is strongly related to the performance ultimately achieved
- Improving the ease of use of HPC systems for domain applications benefits from techniques making up a discipline of code development, such as code reuse and libraries of common codes developed by experts
- *Software engineering* provides principles and practices that guide overall control of workflow management, including testing, and contribute to programmability

HPC: Parallelism

Parallel Architecture Families - Flynn's Taxonomy

A classification system that categorizes computer systems based on their ability to process multiple instructions or data streams simultaneously. It has four categories:

- Single instruction, single data (SISD) - the system can only process one instruction at a time, using one data stream; an example of this would be a traditional single-core computer
- Single instruction, multiple data (SIMD) - the system can process one instruction at a time but can work on multiple data streams simultaneously; an example of this would be a computer using vector processing
- Multiple instruction, single data (MISD) - the system can process multiple instructions at the same time but only works on one data stream
- Multiple instruction, multiple data (MIMD) - the system can process multiple instructions simultaneously and work on multiple data streams simultaneously; an example would be a multi-core computer.

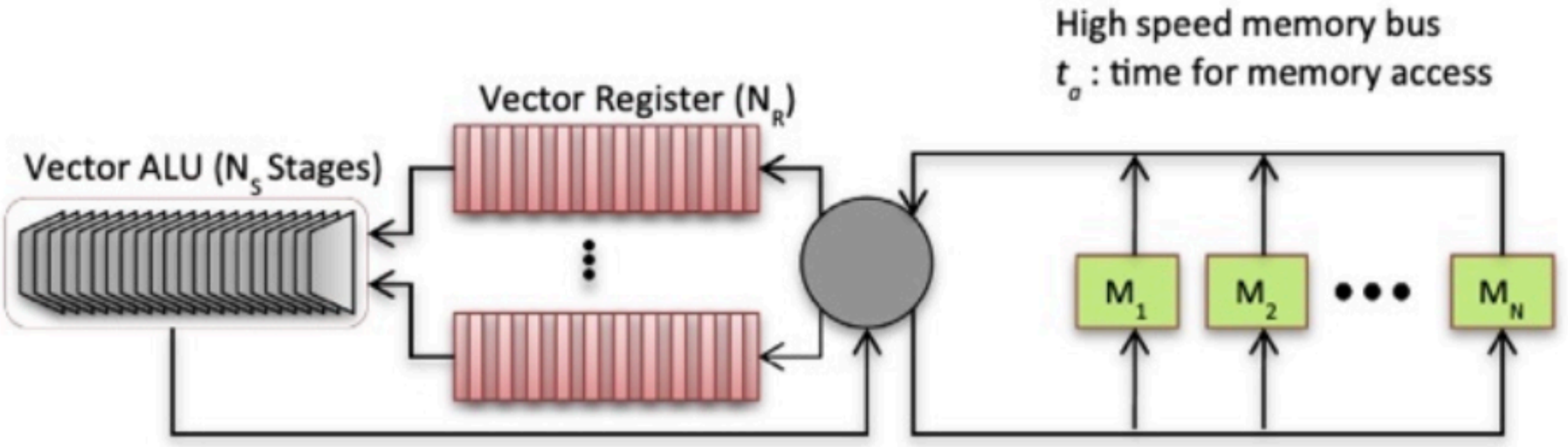
Additional Reading: Flynn's taxonomy

Pipeline Parallelism

- Pipeline parallelism is derived by dividing a complicated action into a sequence of simpler actions that can be performed independently
- Pipelining allows for multiple instances of an operation to be performed simultaneously by dividing it into simpler steps and performing different parts of the operation at the same time
- The parallelism achieved by pipelining depends on the number of instances of the operation that need to be performed and the number of pipeline stages in which the operation can be divided
- Both hardware and software need to work together to exploit pipelining parallelism
- The speedup of a pipeline structure is calculated by comparing the total throughput rate with the execution time of a monolithic version of the same function

Vector Processing

- Vector-processing architecture exploits pipelining to achieve the advantages of fine-grain parallelism, latency hiding, and amortized control overheads
- Pipelining also permits a high clock rate for vector-based computer architecture by keeping the pipeline stages small in terms of the logic depth



Key Differences: SIMD versus Vector Processing

UIC UNIVERSITY OF ILLINOIS CHICAGO

Feature	SIMD	Vector Processing
Scope	Broad category of parallelism	Specialized form of SIMD
Implementation	CPUs (e.g., AVX, SSE), GPUs, DSPs	Supercomputers, modern vector extensions (ARM SVE, RISC-V V)
Data Storage	Typically uses wide registers	Uses dedicated vector registers
Flexibility	Fixed-size data chunks (e.g., 128-bit, 256-bit)	Supports variable-length vectors

Amdahl's Law

Amdahl's Law created by Gene Amdahl in 1967 states that the maximum speed of a task is related to the sum of parallel and serial components

$$speedup = \frac{1}{(1-p)+\frac{p}{n}}$$

- *speedup* is the theoretical speedup of the execution of the whole task
- *n* is the number of processors used;
- *p* is the fraction of the task that can be parallelized

$$20.0095 = \frac{1}{(1-0.95)+\frac{.95}{100}} \qquad 20.0000095 = \frac{1}{(1-0.95)+\frac{.95}{100{,}000}}$$

Source: Amdahl's Law in Wikipedia

Amdahl's Law

- Amdahl's law is a way to calculate the maximum theoretical speedup of a system when only a part of it is improved
- It states that the overall speedup of a system is limited by the proportion of the system that can't be improved, called the ***bottleneck***
- For example, if a system is 90% parallelizable, meaning 90% of it can be improved, the maximum theoretical speedup when using a faster processor is only 10 times, no matter how much faster the new processor is.

So, with respect to SIMD, you can calculate speed up as:

$$speedup = \frac{1}{(1-p)+\frac{p}{n}}$$

$$speedup = \frac{1}{(1-0.9)+\frac{0.9}{10}} = \frac{1}{0.1+0.09} \approx 5.26$$

$$speedup = \frac{1}{(1-0.9)+\frac{0.9}{100}} = \frac{1}{0.1+0.009} \approx 9.17$$

Amdahl's Law and 500x Speedup I

- **Algorithmic Improvements:** Changes to the algorithm, making it more efficient, can lead to significant speedup. These improvements are independent of parallelization and can lead to dramatic increases in performance.
- **Hardware Enhancements:** Utilizing more advanced or specialized hardware (such as GPUs, TPUs, or FPGAs) can provide substantial speedups, especially for tasks well-suited to such hardware.
- **Better Utilization of Resources:** Optimizations that allow for better use of the available computational resources, such as cache optimizations, reduced memory access times, and elimination of bottlenecks, can also lead to significant speed gains.

Amdahl's Law and 500x Speedup II

- **Combining Parallelization with Other Techniques:** Besides parallelizing the parallelizable portion of the code, other techniques like vectorization, pipelining, or asynchronous execution can be applied to enhance performance further.
- **Improved Compiler Optimizations:** Modern compilers can perform sophisticated optimizations that significantly increase the efficiency of the generated machine code, leading to better performance.
- **Reduction of Overheads and Latency:** Minimizing communication overhead, synchronization latency, and other non-computational delays in a parallel computing environment can lead to higher-than-expected speedups.

Sources of Parallel Overheads

- Interprocessor communication
  - Data movement costs
- Load imbalance
- Synchronization
- Extra computation
  - Computation not performed by serial version, e.g., partially-replicated computation to reduce communication
- Contention
  - Memory
  - Network

HPC: Hardware Infrastructure

Mutliprocessors

- Shared-Memory Multiprocessors
- Massively Parallel Processors
- Commodity Clusters

Shared-Memory Multiprocessors

- A shared-memory multiprocessor is an architecture in which a modest number of processors all have direct access to all main memory in the system
- The key component is the interconnection network that directly connects all processors shared memories
- Extra care is needed to retain cache coherence across all caches of all processors in the system
- There are two main configurations of shared-memory multiprocessors:
  - SMP - where all processors can access each memory block in the same amount of time,
  - NUMA - where access times vary, but all processors still have equal access to all memory blocks

Massively Parallel Processors

UIC UNIVERSITY OF ILLINOIS CHICAGO

- MPP architecture is the structure that most easily scales to the extremes of computing system size and performance
- MPP are not shared-memory architectures but are distributed memory, where each group of processor cores is directly connected to their own local memory
- MPPs use a logical capability for message passing enabled by the physical system area network (SAN) that integrates all the nodes to form a single system

Commodity Clusters

UIC UNIVERSITY OF ILLINOIS CHICAGO

- Commodity clusters are a type of supercomputer that consists of commodity subsystems or COTS (commodity off-the-shelf) components
- They emerged in the mid-1990s and leveraged economy of scale to achieve dramatic improvements in performance to cost
- They typically have lower efficiencies compared to MPP (massively parallel processing) systems but are good for throughput computing, such as parameter sweeps
- They are currently the dominant class of deployed supercomputers, with 80% of the systems on the Top 500 list being clusters
- Commodity clusters have historically used a variety of COTS networks such as ATM, Myrinet, and Ethernet 100BaseT, but currently primarily use Gigabit Ethernet or Infiniband Networks

Heterogeneous Computer Structures

- Heterogeneous computing systems use multiple types of processing components, such as multicore processors and accelerators, to perform computation
- Accelerators, such as GPUs, are designed to perform certain classes of computation, such as linear algebra and signal processing, extremely well and are attached to a system node via the I/O bus
- Modern computer architecture often incorporates aspects of different types of architecture, such as sequential, pipelining, SIMD, and multiprocessor organizations, into a single microprocessor socket

What is a Commodity Cluster?

- It is a distributed/parallel computing system
- It is constructed entirely from commodity subsystems
  - All subcomponents can be acquired commercially and separately
  - Computing elements (nodes) are employed as fully operational standalone mainstream systems
- Two major subsystems:
  - Compute nodes
  - **System area network** (SAN)
- Employs industry standard interfaces for integration
- Uses industry standard software for majority of services
- Incorporates additional middleware for interoperability among elements
- Uses software for coordinated programming of elements in parallel

Why are Clusters so Prevalent

- Excellent performance to cost for many workloads
  - Exploits economy of scale
    - Mass produced device types
    - Mainstream standalone subsystems
  - Many competing vendors for similar products
- Just in place configuration
  - Scalable up and down
  - Flexible in configuration
- Rapid tracking of technology advance
  - First to exploit newest component types
- Programmable
  - Uses industry standard programming languages and tools
- User empowerment
  - Low cost, ubiquitous systems
  - Programming systems make it relatively easy to program for expert users

HPC: resource management

Need for Resource Management

UIC UNIVERSITY OF ILLINOIS CHICAGO

- Protecting significant infrastructure investment
  - Supercomputing hardware
  - Hosting data center
  - Power delivery (cabling, UPSs, generators, flywheels)
  - Cooling infrastructure (HVAC units, heat exchangers, pipes for water cooling, etc.)
- Cost of ownership
  - Electricity ($0.5M/1MW per year)
  - Support contracts (both hardware and software)
  - Maintenance crew (system administrators and technicians)
- Getting the most for your investment
  - Efficient workload scheduling (maximum utilization)
  - Maximizing hardware resource utilization (maximum computation)
  - Compute job prioritization

Types of Managed Resources

- Compute nodes
- Processing cores
- Interconnect
- Permanent storage and I/O
- Accelerators

Compute Job

- A self-contained work unit that may need specific input data and produces a result in the course of its execution (output)
  - Input may be typed by the user or a file
  - Output could be a file, a printout on the console or graphics displayed on screen
- Primary modes of job processing
  - Interactive: requires interaction with user
  - Batch: resource requirements, job description, and inputs are fully specified at job submission time
- Batch processing accounts for most of machine's computational throughput
- May be subdivided into smaller steps or tasks
  - Data staging
  - Data pre- and post-processing
  - Visualization
  - Processing application pipelines

Job Queue

- Defines order in which jobs are executed on the machine
  - Typically FIFO (“first-in, first-out”)
  - Job schedulers may deviate from the prearranged order to improve utilization/throughput or due to operator’s override
- Groups jobs targeting the same set of resources
  - Most systems have multiple job queues, for example
  - Production
    - Job size
    - Job duration
  - Debug
  - Interactive
  - Managing specific resources, such as large memory nodes or accelerators

Job Scheduling

- Critical to achieving good utilization of the machine
- Must accommodate potentially hundreds or thousands of waiting jobs and a significant number of concurrently executing jobs
- Parameters affecting job scheduling
  - Resource availability
  - Job priority
  - Resources allocated to the user
  - Maximum number of jobs
  - Requested execution time
  - Elapsed execution time
  - Job dependencies and prerequisites
  - Occurrence of specified events
  - Operator availability
  - Software license availability

Common Resource Managers

- Simple Linux Utility for Resource Management (Slurm)
- Portable Batch System (PBS)
- OpenLava based on the Load Sharing Facility (LFS)
- Moab Cluster Suite
- Tivoli Workload Scheduler LoadLeveler
- Univa Grid Engine
- HTCondor
- OAR
- Hadoop Yet Another Resource Negotiator (YARN)

HPC: software you need to know

What is OpenMP?

OpenMP is ...

- an application programming interface (API) to support shared-memory multiple-thread parallel application development
- stands for ***open multiprocessing***
- simplifies the development of multiple-thread parallel programming
- has bindings for Fortran, C, and C++
- initially released v1.0 1998 (C/C++), v2.0 2002, v2.5 2005 (C/C++/Fortran), v3.0 2008, v4.0 2013, v5.0 2018, **current version v5.2**, version v6.0 due for release in 2024

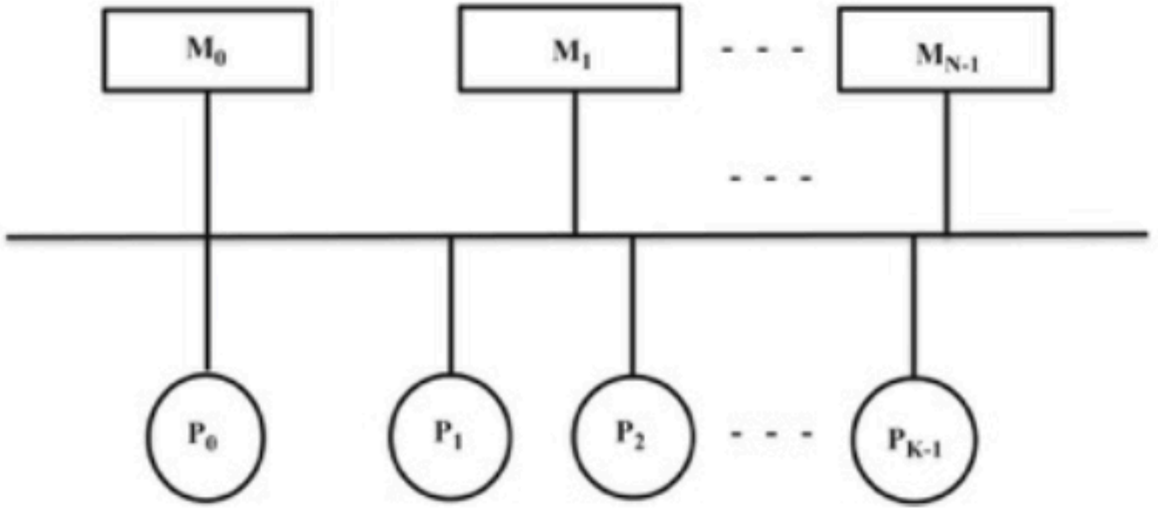
What is OpenMP?

OpenMP is ...

- **NOT** a programming language
- A set of compiler **directives** that help the application developer to parallelize their workload
- These directives, environment variables, and the library routines form OpenMP

OpenMP Programming Model

- OpenMP provides a shared-memory multiple-threads programming model with the assumption that there is underlying hardware support, including
  - virtual addresses
  - cache coherency (both among processors cores and across sockets)
- Introduction to:
  - thread parallelism
  - thread variables
  - runtime library and environment variables



Message Passing Interface Standard

- Developed from 1992 - 1994
- By a community of both vendors and users
- Goal to create a standard interface to message passing calls
- Targeted at distributed memory parallel computers - initially, this was for massively parallel supercomputers as clusters weren't widely deployed at the start
- MPI-1 intended just as an API
- FORTRAN77 and C bindings
- Reference implementation (MPICH) also developed
- Many vendors had their **own** implementations
- MPI 4.1 is the latest version of the standard[1]

Note: Textbook uses the MPI-1 standard

[1] MPI Standard Documents (11/02/2023), MPI 5.0 future version

Evolution of the Standard

- **MPI-1:**
  - Released in 1994
  - Standardized point-to-point communication and collective operations
  - Included support for synchronous and asynchronous communication
  - Provided basic support for process topologies and process groups
- **MPI-2:**
  - Released in 1997
  - Added support for one-sided communication and remote memory access
  - Introduced dynamic process management and process topologies
  - Included support for non-blocking collective operations
  - Provided additional support for input/output and parallel I/O

Evolution of the Standard

- **MPI-3:**
  - Released in 2012
  - Added support for more advanced one-sided communication features, such as remote atomics and shared memory windows
  - Improved support for collective operations, including neighborhood collectives and non-blocking collectives
  - Added support for multithreading and concurrent communication
  - Provided additional support for input/output and parallel I/O
- **MPI-4:**
  - Current version 4.1 (development started in 2015)
  - Improves support for non-blocking operations and one-sided communication
  - Adding support for fault tolerance and resiliency
  - Improving support for parallel I/O and performance optimization

HPC: Benchmarks

Computer Performance

Performance in general is how well something performs; in the context of computing, common parameters are **time** and **work**

- **Peak** performance
  - Maximum rate at which operations can be accomplished theoretically by the hardware resources
  - Determined by:
    - Clock rate provided by the device technology
    - Hardware parallelism determined by the computer architecture

$$PeakFLOP/s = numprocessors \times numcores\ per\ processor \times clockspeed \times num\ instructions$$

- **Sustained** performance is the actual or real performance achieved by a supercomputer system in running an application program

Computer Performance: Degradation

What makes a computer **SLOW**?

- **S**tarvation - there is not enough work to keep all the functional units busy, or the problem is not balanced (some functional units have too much work and others not enough)
- **L**atency - the time it takes for information to travel from one part of the system to another
- **O**verhead - the amount of additional work to manage the system
- **W**aiting - time waiting for access to a resource

Performance and Metrics

- No single measure of **performance** fully reflects all aspects of the quality of computer operations
- **Metric** is a quantifiable, observable operational parameter
- Two basic measurements are used with respect to HPC
  - Time
  - Number of operations
- Historically in HPC *floating-point operations per second* or *flops* was the most widely used metric, but today you will hear *green flops* (flops per watt) or *ops* (operations per second) for a given number of bits

Introduction

- **Benchmarking** is a way to measure the performance of a supercomputer empirically
- Provides some standardized type of workload that may vary in size or input dataset
- Outputs a performance metric that can be used for comparing the performance between different supercomputers based on a specific workload
- Example performance metrics:
  - floating point operations per second (FLOPS)
  - traversed edges per second (TEPS)
  - giga updates per second (GUPS)
  - degrees of freedom per second (DOFS)
- Computational benchmark workloads come in two types:
  - synthetic, where workloads are designed and created to impose a load on a specific component in the system
  - application, where the workload is derived from a real-world application

Key Properties of an HPC Benchmark

UIC UNIVERSITY OF ILLINOIS CHICAGO

- Good benchmarks are:
  - Relevant and meaningful to the target application domain
  - Applicable to a broad spectrum of hardware architectures
  - Respected and used by users and vendors to enable comparative evaluation
- Tend to be relatively short and use standard parallel programming API's

Benchmark	Approximate Line Count	Parallelism		Language	
		MPI	OpenMP	C	C++
HPL	26700	X		X	
STREAM	1500			X	
RandomAccess	5800	X	X	X	
HPCG	5700	X	X		X
IS	1150	X	X	X	
Graph500	1900	X	X	X	
HPGMG	5000	X	X	X	

Key Properties of an HPC Benchmark

Different benchmarks test/characterize different things

Benchmark	Supercomputer	Location	Results	Cores
HPL	El Capitan	Oak Ridge National Laboratory	1.742 EFlop/s	11,039,616
HPGC	Fugaku	RIKEN Center for Computational Science	16 PFlop/s	7,630,848
Graph500	Fugaku	RIKEN Center for Computational Science	204.068 TTEPS[1]	7,299,072
Green500	Jedi	EuroHPC/FZJ	4.50 PFlop/s	19,584

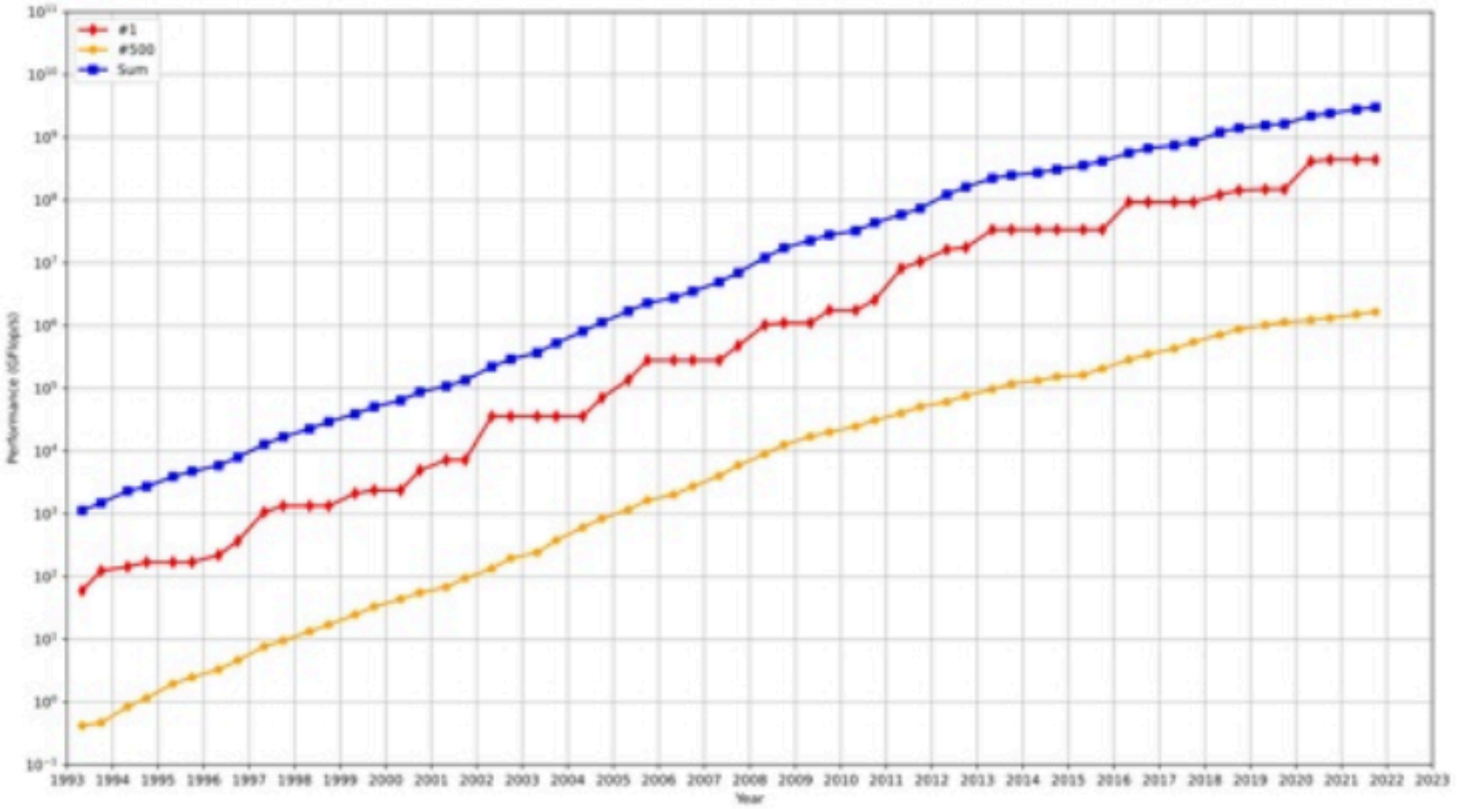
#1 on one benchmark does not imply you will be #1 on another

Supercomputer	Top500	HPGC	Green500	Graph500
El Capitan	1	-	-	-
Frontier	2	2	-	4
Aurora	3	3	-	5
Eagle	4	-	-	-

[1] Tera Traversed Edges per Second

Performance and Metrics: Benchmarks

- Standard application used to measure performance is a **benchmark**
- Most common benchmark is *Linpack* or *HPL* (highly parallel Linpack)
  - Used to determine the Top500 fastest computers since 1993
  - Solves a set of linear equations in dense matrix form



Additional reading: The LINPACK Benchmark: past, present and future by Jack J. Dongarra, Piotr Luszczek, Antoine Petitet

HPL vs. HPCG: What's the Difference?

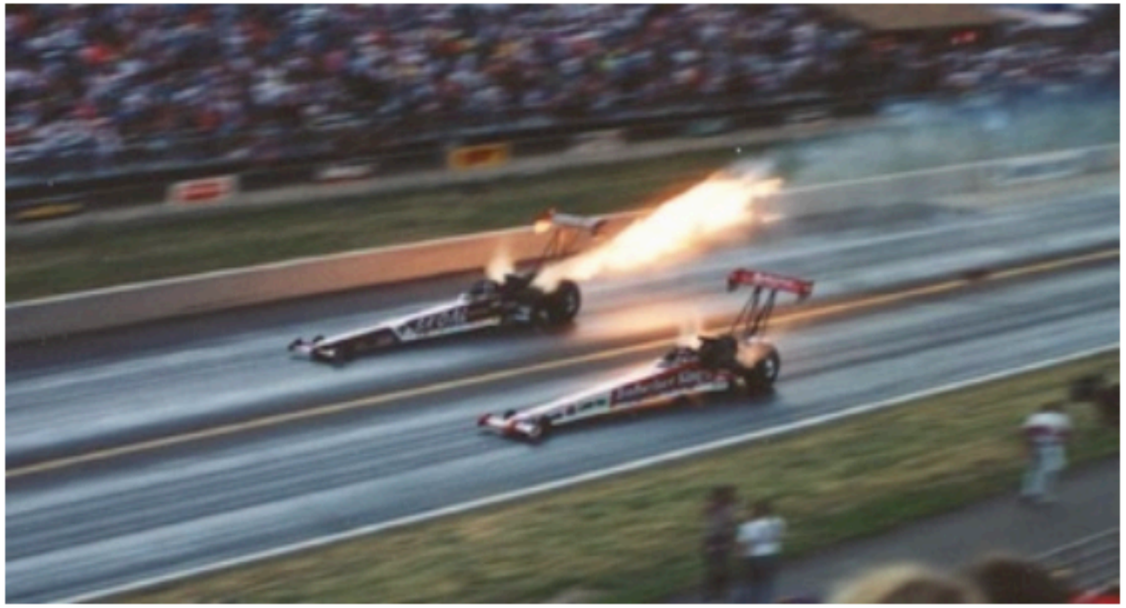
UIC UNIVERSITY OF ILLINOIS CHICAGO

HPL (High-Performance Linpack)	HPCG (High-Performance Conjugate Gradient)
Measures the **best-case scenario** for supercomputers, solving large dense linear systems	Measures a **realistic scenario**, solving sparse linear systems, which is more common in real-world applications
Uses **LU factorization**, which is highly optimized and fully uses floating-point performance	Uses **iterative methods**, which are harder to optimize and rely on memory bandwidth and system balance
Runs well on systems with high **floating-point performance** (FLOPs) and optimized matrix multiplication	Runs well on systems with **high memory bandwidth** and good performance on irregular computations
Often results in very **high performance numbers**, making supercomputers look more powerful than they are for general workloads	Produces **more realistic performance**, reflecting the challenges of real-world scientific computing

HPL vs. HPCG: What's the Difference?

UIC UNIVERSITY OF ILLINOIS CHICAGO

- Think of **HPL** like testing a car's speed on a straight highway—measuring raw horsepower.
- **HPCG** is like testing how well the car handles traffic, turns, and rough roads—closer to real-world driving.



HPC: Summary

Summary

- High Performance Computing (HPC) is able to solve the largest and most complicated computational problems
  - Improved steadily by factor 200 every decade
  - Exploits performance in both hardware and software simultaneously
- It achieves performance through a combination of speed, parallelism, efficiency
  - Success is influenced by power, reliability and programmability
- Famous benchmark indicate the performance of HPC machines
  - Different benchmarks test different characteristics of a machine
- Programming for an HPC machine is very sophisticated and has its own optimized toolkits like OpenMP and MPI

Slides are based on a current university course of Mike Papka (ANL and UIC) and work of Zhiling Lan (UIC and ANL). Many thanks!