Compute Big, Think Bigger: An Intro to High-Performance Computing (HPC)

Sameh Abdulah
Senior Research Scientist

Division of Computer, Electrical, and Mathematical Sciences and Engineering (CEMSE),
King Abdullah University of Science and Technology
Thuwal, Jeddah 23955, Saudi Arabia.

The Abdus Salam International Centre for Theoretical Physics

UM6P University Mohammed VI Polytechnic

Do it while you wait.

KAUST

Sameh's Background

- B.S. in Computer Science, Egypt, 2005
- M.S. in Computer Science, Egypt, 2009
- M.S. in Computer Science and Engineering, The Ohio State University (OSU), Columbus, Ohio, USA, 2014
- Ph.D. in Computer Science and Engineering, The Ohio State University (OSU), Columbus, Ohio, USA, 2016
- Postdoc at the Extreme Computing Research Center (ECRC), KAUST, Saudi Arabia, 2016-2019
- Research Scientist at the Extreme Computing Research Center (ECRC), KAUST, Saudi Arabia, 2019-2024
- Senior Research Scientist at the Extreme Computing Research Center (ECRC), KAUST, Saudi Arabia, 2025-now

Research Interests: HPC - Large-Scale Statistical Computing - Parallelization of Data-intensive and Compute-intensive Applications - Extreme Scale Machine Learning and Data Mining (MLDM) Algorithms

KAUST

Table of Contents

Table of Contents

KAUST

Parallel Computing

- Simultaneous execution of multiple calculations or processes
- Large problems are divided into smaller sub-problems, solved concurrently
- Types of Parallel Computing:
  - Data Parallelism: Distributes data across computing nodes, performing the same operation on each
  - Task Parallelism: Distributes different tasks across computing nodes
- Architectures of Parallel Computing:
  - Multicore Computing: Multiple processing units (cores) on a single chip, sharing memory and peripherals
  - Distributed Computing: Multiple autonomous computers connected through a network, each having its own memory and processors
  - Supercomputing: High-performance computing systems designed to perform complex and large-scale computations

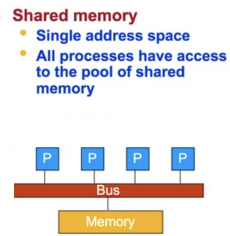
Achieving Parallelism

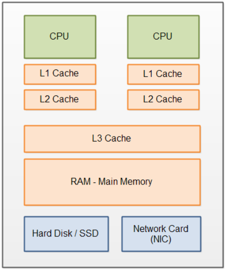
- Implicit Parallelism (Compiler/Runtime-managed)
  - Automatically detects opportunities for parallelism
  - Assigns tasks for parallel execution
  - Manages execution and synchronization

- Explicit Parallelism (Programmer-managed)
  - Annotates tasks for parallel execution
  - Assigns tasks to specific processors
  - Manually controls execution and synchronization

MultiCore Computing (Shared Memory)

- Multiple CPU cores on one chip share main memory
- Private caches (L1/L2) per core; shared L3 lowers latency for inter-core sharing
- Best for threads (OpenMP/TBB): parallel loops, reductions, task graphs
- Watch for:
  - Memory bandwidth effects: If memory bandwidth is saturated, adding cores will not help
  - NUMA effects: Accessing "local" memory is faster than "remote" memory on another socket





KAUST

Distributed Computing

- Multiple system processors can communicate with each other using messages sent over the network
  - Cluster: Interconnected computers acting as one system
  - Supercomputer: A single, extremely powerful machine designed for highly complex and intensive computational tasks
- With a sufficiently fast network, it is theoretically possible to scale to millions of CPU cores (and beyond)
- Benefits: Scalability, reliability, fault tolerance, and performance
- Challenges: Complex architecture, construction, and debugging processes

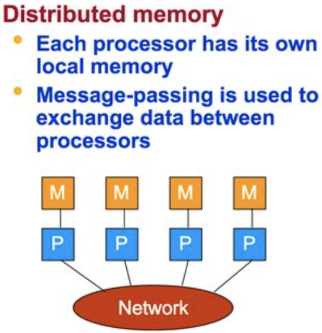
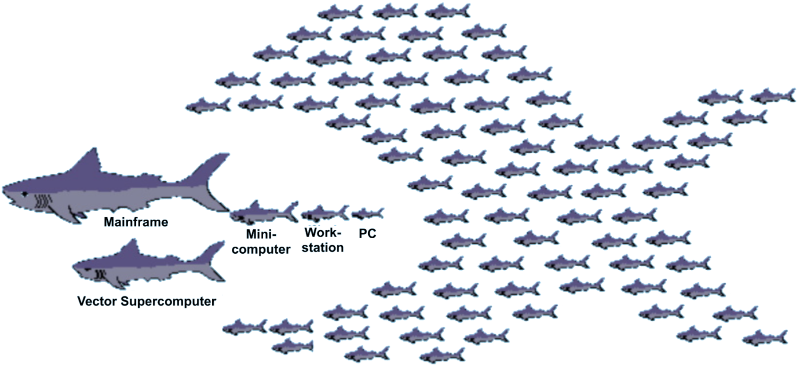


Table of Contents

Computer Food Chain



© late '90s Berkeley's Network of Workstations (NOW) project.

High-Performance Computing (HPC)

- There is no clear definition!
- My preference: High-Performance Computing (HPC) refers to aggregating computing power to deliver much higher performance than one could get out of a typical desktop computer or workstation
- HPC is essential for several reasons, particularly in fields where complex and large-scale computing tasks are routine
  - Handling Large-scale Computations
  - Speeding Up Research and Development
  - Advanced Simulation Capabilities
  - Big Data Analytics
  - Artificial Intelligence and Machine Learning
  - Competitive Advantage in Industry
  - National Security and Defense

KAUST

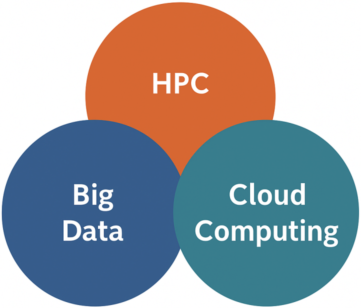
Cont.

- Parallel processing and distributed computing
  - Matured over the past decades
  - Emerged as a well-developed field in computer science
  - Still a lot of innovation, e.g., hardware/software
- Scientific computing with Matlab, R, etc.
  - Performed on small computing machines
  - Increasing number of cores enables better scientific computing today
  - Good for small/less complex applications, quick reach memory limits
- Advanced scientific computing
  - Used with computational simulations and large-scale machines
  - Performed on large parallel computers; often, scientific domain-specific approaches
  - Uses orders of magnitude multi-core chips, large memory, and many specific many-core chips
  - Enables simulations of reality, often based on known physical laws and numerical methods

KAUST

Why HPC?

- Massive growth in data across all fields and industries, for example, genomic data, electronic health records, and real-time patient monitoring in healthcare, creating unprecedented challenges and opportunities
- Urgent need for scalable computing solutions to handle large, complex datasets and computationally intensive tasks
- Cloud democratizes access to computational resources
- HPC accelerates scientific discovery through large-scale simulations and data analysis



When HPC?

- Complete a time-consuming operation in less time
- Perform a high number of operations per second
- Process datasets that exceed a single machine's memory
- Run large ensembles or many independent tasks
- Meet tight deadlines or real-time constraints (streaming/nowcasting)
- High-fidelity simulation and digital twins
- Train or serve large ML models efficiently (GPU/accelerators)
- Federated or cross-site analysis with privacy/security requirements

What does HPC include? 1/2

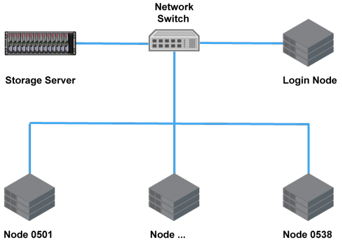
- Hardware stack
  - Parallel execution across many compute elements (CPUs, GPUs, and other accelerators)
  - High-speed interconnects between nodes (e.g., InfiniBand, HPE Slingshot)
  - Deep memory hierarchies (HBM + DDR; NUMA-aware node designs)
- Software stack
  - Programming models: MPI, OpenMP, CUDA/HIP/SYCL)
  - Math & domain libraries: BLAS/LAPACK/ScaLAPACK, FFTW, PETSc/Trilinos, MAGMA, oneMKL, cuBLAS/cuDNN
  - I/O & data formats: MPI-IO, HDF5, NetCDF
  - Compilers & build tooling: GCC/Clang/Intel/NVHPC/Cray; CMake, Spack, EasyBuild, environment modules

What does HPC include? 2/2

- Software stack
  - Schedulers & orchestration: Slurm, PBS Pro, LSF; workflows (Snakemake, Nextflow, Pegasus)
  - Profiling & debugging: perf/gprof, Valgrind, VTune, Arm MAP/Forge, Nsight/rocprof, TAU.
  - Containers & reproducibility: Apptainer/Singularity (runtime), Docker (build), CI/versioning
- Data & storage
  - Parallel filesystems (Lustre, GPFS/Spectrum Scale, BeeGFS), burst buffers, object stores
  - Checkpoint/restart and data management strategies
- Operations
  - Resource management, monitoring, and security; facility concerns (power, cooling, reliability)
  - User support, documentation, and training

How does HPC work?

- Three main components:
  - Compute (CPU/GPU nodes)
  - Network (high-speed interconnect)
  - Storage (parallel/distributed file systems)



- Programs and algorithms run simultaneously across servers (parallel jobs)
- Shared storage for reading inputs and capturing outputs
- A scheduler (e.g., Slurm/PBS) brokers user jobs, resources, and data flow so the system operates seamlessly to complete diverse tasks

HPC Myths (and Realities)

- A niche for researchers, geeks, and "eggheads."
  Reality: Used widely in oil & gas, automotive, aero, manufacturing, pharma, finance, and more
- It's not for the cloud / not needed in the cloud.
  Reality: Cloud offers HPC instances and high-speed interconnects
- HPC means one giant mainframe/supercomputer only.
  Reality: Modern HPC spans clusters, accelerators, and even edge
- HPC is only MPI/Fortran
  Reality: Ecosystem includes Python/R/C++, CUDA/HIP, OpenMP, SYCL, and task runtimes/workflows.
- HPC is only about FLOPS
  Reality: Memory BW, storage I/O, and latency often bottleneck
- HPC adoption is too costly
  Reality: Shared facilities and cloud cut costs; pay-as-you-go available
- HPC isn't reproducible
  Reality: Containers, modules, and workflow managers enable portability

KAUST

Table of Contents

KAUST

FLOPS: The Speed of HPC (kilo $\rightarrow$ yotta)

HPC term often applies to systems that function above a TFLOPS or $O(10^{12})$ floating-point operations per second (Flops/s)

Name	Unit	Value
kiloFLOPS	kFLOPS	10^3
megaFLOPS	MFLOPS	10^6
gigaFLOPS	GFLOPS	10^9
teraFLOPS	TFLOPS	10^{12}
petaFLOPS	PFLOPS	10^{15}
exaFLOPS	EFLOPS	10^{18}
zettaFLOPS	ZFLOPS	10^{21}
yottaFLOPS	YFLOPS	10^{24}

TOP500 at a Glance

- What it is? Biannual ranking of the world's fastest supercomputers (released at *ISC* in June and *SC* in November)
- Benchmark: HPL (LINPACK) in double precision; key numbers:
  - Rmax (measured),
  - Rpeak (theoretical),
  - system power (MW), and
  - efficiency (GF/W)
- Companion views: *Green500* (energy efficiency) and *HPCG* (memory/communication intensive performance)
- Trends: Heterogeneous (CPU+GPU) designs dominate; high-speed interconnects (e.g., InfiniBand/Slingshot); rising focus on perf/W
- Caveat: HPL is compute-bound; it may overestimate performance for memory/communication-bound workloads – use HPCG/application results for balance

LINPACK (HPL): TOP500 Performance Benchmark

- What it is? Solves a dense FP64 linear system via LU factorization; reports sustained PFLOP/s (Rmax). Basis of the TOP500
- How it runs? MPI + threads (often OpenMP), 2D block-cyclic data layout; tuned by problem size N, process grid $P \times Q$, block size, panel factorization/lookahead, GPU BLAS, pinned memory
- Strengths: Portable, comparable across systems; good proxy for peak floating-point throughput; exposes node/GPU capability and interconnect broadcast performance
- Caveats: Not representative of memory-bound or irregular apps; that's why other benchmarks (e.g., HPCG) exist
- Related: *HPL-AI / HPL-MxP* (mixed precision, tensor cores) and *HPCG* (memory/communication intensive) provide complementary views; energy tracked via GF/W (Green500)

Top 10 Supercomputers (HPL)

#	System	Site	Country	Rmax (PFLOP/s)	Rpeak (PFLOP/s)	Power (kW)
1	El Capitan	Lawrence Livermore National Lab. (LLNL)	United States	1,742.00	2,746.38	29,581
2	Frontier	Oak Ridge National Laboratory (ORNL)	United States	1,353.00	2,055.72	24,607
3	Aurora	Argonne National Laboratory (ANL)	United States	1,012.00	1,980.01	38,698
4	JUPITER	Jülich Supercomputer Center (JSC)	Germany	793.40	930.00	13,088
5	Eagle	Microsoft Azure	United States	561.20	846.84	N/A
6	HPC6	Eni S.p.A.	Italy	477.90	606.97	8,461
7	Fugaku	RIKEN R-CCS	Japan	442.01	537.21	29,899
8	Alps	Swiss National Supercomputing Centre (CSCS)	Switzerland	434.90	574.84	7,124
9	LUMI	EuroHPC/CSC	Finland	379.70	531.51	7,107
10	Leonardo	EuroHPC/CINECA	Italy	241.20	306.31	7,494

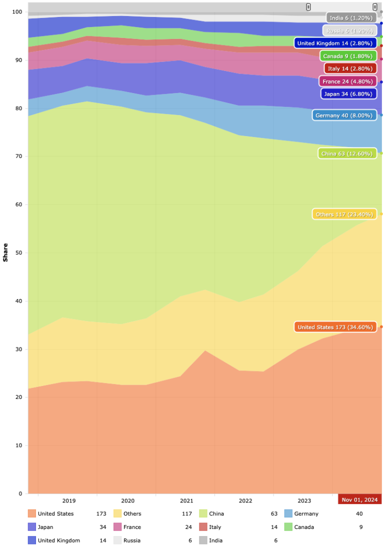
Top Systems: HPL (R_{max}) and HPCG

Rank	System	Country	Cores	Rmax [PF/s]	Rpeak [PF/s]	Power [kW]
1	El Capitan	United States	11,039,616	1,742.00	2,746.38	29,581
2	Frontier	United States	9,066,176	1,353.00	2,055.72	24,607
3	Aurora	United States	9,264,128	1,012.00	1,980.01	38,698
4	JUPITER Booster	Germany	4,801,344	793.40	930.00	13,088
5	Eagle	United States	2,073,600	561.20	846.84	—
6	HPC6	Italy	3,143,520	477.90	606.97	8,461
7	Supercomputer Fugaku	Japan	7,630,848	442.01	537.21	29,899
8	Alps	Switzerland	2,121,600	434.90	574.84	7,124
9	LUMI	Finland	2,752,704	379.70	531.51	7,107
10	Leonardo	Italy	1,824,768	241.20	306.31	7,494

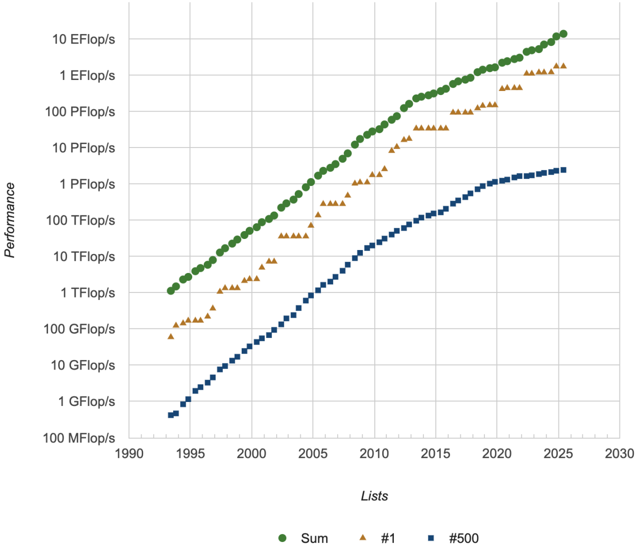
Top 10 Supercomputers (Green500)

#	TOP500 Rank	System	Country	Cores	Rmax (PFLOP/s)	Power (kW)	Efficiency (GFLOPS/W)
1	259	JEDI	Germany	19,584	4.50	67	72.733
2	148	ROMEO-2025	France	47,328	9.86	160	70.912
3	484	Adastra 2	France	16,128	2.53	37	69.098
4	183	Isambard-AI phase 1	United Kingdom	34,272	7.42	117	68.835
5	255	Otus (GPU only)	Germany	19,440	4.66	N/A	68.177
6	66	Capella	Germany	85,248	24.06	445	68.053
7	304	SSC-24 Energy Module	South Korea	11,200	3.82	69	67.251
8	85	Helios GPU	Poland	89,760	19.14	317	66.948
9	399	AMD Duranos	France	16,632	2.99	48	66.464
10	412	Henri	United States	8,288	2.88	44	65.396

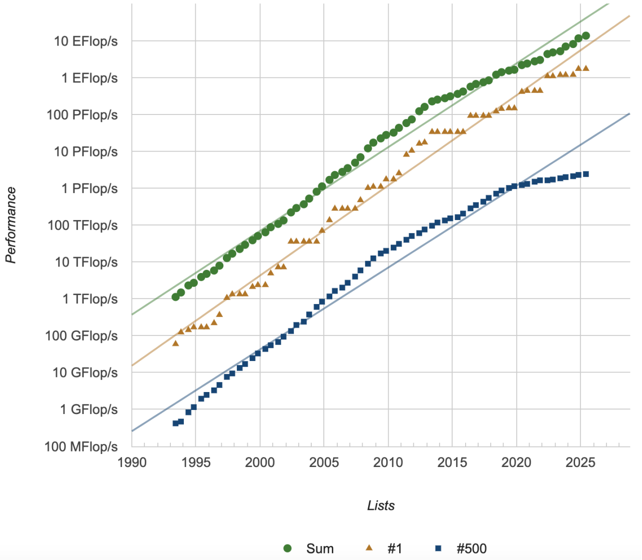
TOP500 List - Countries Share Over Time



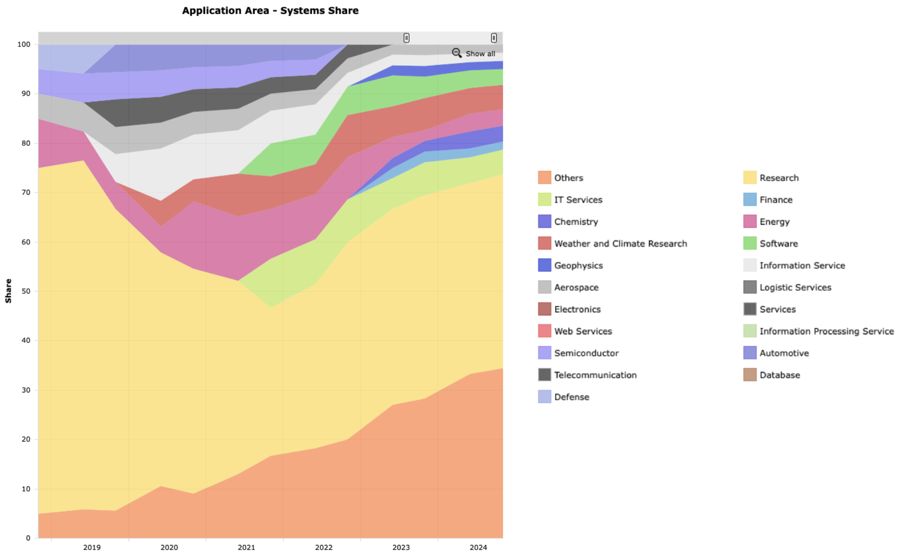
TOP500 List - Performance Over Time



TOP500 List - Projected Performance Over Time

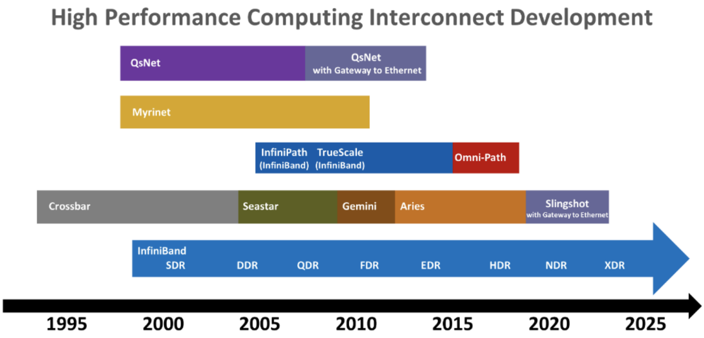


TOP500 List - Applications Share Over Time



Interconnects in Supercomputers

- The network fabric that links CPUs, GPUs, and nodes so they can exchange data fast and in parallel
- Latency (μs), per-link bandwidth (GB/s), message rate (Mmsg/s)
- Common fabrics: InfiniBand (HDR/NDR), HPE Slingshot, high-speed Ethernet/RoCE; in-node NVLink/NVSwitch



Top-performing Interconnects — Top 5

Rank	Interconnect	Link rate	Why it leads
1	InfiniBand NDR400	400 Gb/s	Very low latency, high msg rate; SHARP offload; GPUDirect RDMA — common in exascale/AI systems.
2	InfiniBand NDR200	200 Gb/s	NDR features with lower per-port rate; mature ecosystem and toolchain.
3	Slingshot-11	200 Gb/s	Adaptive routing on Dragonfly+, congestion control; backbone of HPE Cray exascale systems.
4	InfiniBand HDR	200 Gb/s	Proven across many TOP500 systems; strong collectives and RDMA offloads.
5	InfiniBand EDR	100 Gb/s	Lower latency and jitter than 100G Ethernet/RoCE for HPC collectives.

Note: "Best" depends on workload and network design (topology/rails), but these five typically deliver the highest sustained HPC performance.

TOP500 Interconnect System Share

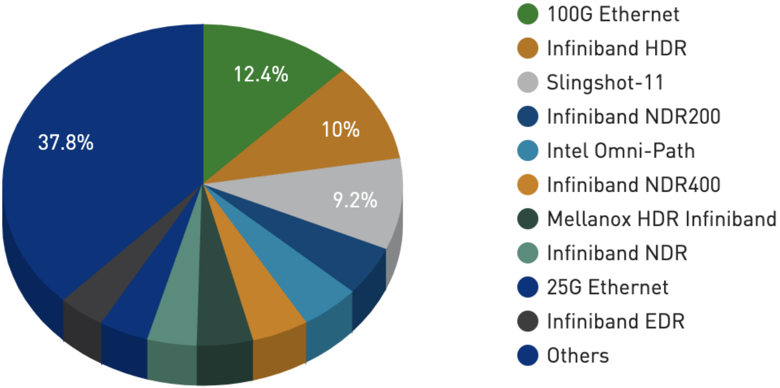


Table of Contents

Energy Efficiency in HPC (1/3): Why It Matters

- Scale = Power Hungry: Top supercomputers consume tens of MW (e.g., Frontier $\sim$25 MW)
- A typical household in Morocco uses 3,500 kWh/year (3.5 MWh).
- Frontier in one hour (25 MWh) uses as much electricity as about 7 average Moroccan households consume in an entire year.
- In one year, Frontier consumes about as much as 62,000 Moroccan households combined.
- Environmental impact: Carbon footprint of HPC centers pushes sustainability frontiers
- Metrics:
  - GFLOPS/W (Green500 benchmark) – efficiency measured per watt
- Future exascale $\rightarrow$ zettascale computing will be constrained by watts, not FLOPS

Energy Efficiency in HPC (2/3): Techniques

- Hardware-level:
    - GPUs/accelerators with higher performance-per-watt than CPUs
    - Specialized chips (TPUs, IPUs, DPUs) designed for efficiency
    - High Bandwidth Memory (HBM) reduces energy per byte
- Algorithmic:
    - Communication-avoiding and energy-aware algorithms
    - Mixed-precision and low-rank approximations to reduce compute load
    - Load balancing to avoid idle power consumption
- System/Software:
    - Dynamic voltage/frequency scaling (DVFS)
    - Energy-aware scheduling in Slurm/PBS.
    - Containers + lightweight OS for reducing overhead

Energy Efficiency in HPC (3/3): Future & Challenges

- Exascale & beyond: Must sustain ∼20–30 MW budgets for exascale, ∼100 MW infeasible
- Design frontier: Co-design of algorithms, software, and hardware for efficiency
- Emerging trends:
  - Near-memory and in-network computing
  - Liquid cooling and advanced facility design for thermal efficiency
- Trade-offs: Performance vs. accuracy vs. watts
- Take-home: The real "race" is not only FLOPS, but *FLOPS per Watt*

Table of Contents

KAUST

What is Distributed Computing?

- Computation spread across multiple machines: Tasks are divided and executed simultaneously on a network of interconnected computers (nodes) to solve large or complex problems more efficiently
- Coordination and communication required: Nodes must work together, exchanging data and synchronizing operations to ensure consistency and correctness.
- Enables scalability, fault tolerance, and resource sharing across distributed systems
- Used in scientific simulations, data processing pipelines, distributed databases, and real-time analytics applications

KAUST

Distributed System Architectures (Software View)

- Client-Server: A centralized server provides resources or services to multiple client machines that request them
  - Example: A bank's centralized servers handle customer authentication, account queries, and transactions, while clients (apps, ATMs, or browsers) send requests.
- Peer-to-Peer (P2P): All nodes have equal roles as clients and servers, sharing resources directly without centralized coordination
- Master-Slave: A master node controls and delegates tasks to multiple slave nodes, which perform the actual computations or operations
  - Example: Hadoop MapReduce framework, database replication
- Each architecture has trade-offs in scalability, fault tolerance, performance, and complexity

Cluster Management

- Resource managers: Software systems that allocate and monitor resources (CPU, memory, storage) across a distributed cluster
  - Examples: YARN (Yet Another Resource Negotiator) for Hadoop, Mesos for multi-framework support, Kubernetes for managing containerized applications, and SLURM for resource management and job scheduling on high-performance computing (HPC) clusters
- Scheduling jobs across nodes: Assigning tasks to available nodes based on resource availability, workload balancing, and priority policies to optimize performance and throughput
- Fault tolerance: Detecting and recovering from node failures by automatically rescheduling tasks or redistributing workloads
- Monitoring and scaling: Tracking system health, usage metrics, and scaling clusters up or down dynamically to match demand

Communication in Distributed Systems

- Message Passing Interface (MPI): A standardized and portable communication protocol used for parallel programming on distributed memory systems.
- Point-to-Point Communication: Enables direct message exchange between pairs of processes
  - Example: `MPI_Send` and `MPI_Recv`
- Collective Communication: Involves groups of processes to perform operations like broadcast, scatter, gather, and reduce
  - Example: `MPI_Bcast`, `MPI_Reduce`
- Synchronization and Coordination: MPI provides mechanisms like barriers and communicators to synchronize tasks and manage groups of processes
  - Example: `MPI_Barrier`, `MPI_Comm_split`
- Challenges: Scalability, deadlock avoidance, efficient communication patterns, and portability across HPC architectures

MPI Execution Model

- Programs follow the SPMD (Single Program, Multiple Data) model.
  - All processes execute the same program.
  - Each process can follow different branches depending on its rank.
- Each process is assigned a unique rank in a communicator.
  - `MPI_Comm_size`: total number of processes.
  - `MPI_Comm_rank`: rank ID of the current process (0, 1, 2, ...).
- Execution is started by the runtime environment:
  - `mpirun -np N ./program` (launches N processes).
- Typical workflow:
  1. Initialize MPI: `MPI_Init`.
  2. Query communicator info: `MPI_Comm_rank`, `MPI_Comm_size`.
  3. Perform communication (send/receive, collective).
  4. Finalize MPI: `MPI_Finalize`.

Basic MPI Operations

- Point-to-point communication:
  - MPI_Send, MPI_Recv
  - Used for explicit message passing between two processes
  - Can be blocking (waits until completion) or non-blocking (MPI_Isend, MPI_Irecv)
- Collective communication:
  - MPI_Bcast: send data from one process to all others
  - MPI_Gather, MPI_Scatter: collect data from all processes into one or distribute data from one to all
  - MPI_Reduce, MPI_Allreduce: combine values across processes (sum, max, min, etc.) and return the result
  - MPI_Alltoall: every process sends data to every other process (useful in matrix operations)
- Synchronization:
  - MPI_Barrier: all processes wait until everyone reaches this point before continuing
- Communicators and groups:
  - Default communicator is MPI_COMM_WORLD
  - Allows defining subgroups of processes for targeted communication

MPI in Practice: Applications and Optimization

- Applications:
  - Climate modeling, computational fluid dynamics, molecular dynamics simulations.
  - Widely used in scientific codes like WRF, LAMMPS, GROMACS, and PETSc
- Performance Optimization:
  - Use non-blocking communication (`MPI_Isend`, `MPI_Irecv`) to overlap computation and communication
  - Minimize communication volume with domain decomposition
  - Exploit process affinity and topology-aware communication
- Scalability Considerations:
  - Efficient load balancing and minimizing communication bottlenecks are key to scaling MPI applications to thousands of cores
  - Hybrid models (e.g., MPI+OpenMP) are often used on modern HPC architectures

Distributed File Systems

- Hadoop Distributed File System (HDFS): Designed to store very large files across multiple machines; provides fault tolerance through data replication
  - Example: Used by Hadoop for storing input and output data for MapReduce jobs
- Google File System (GFS): Proprietary file system developed by Google to support large-scale data-intensive applications with scalability, fault tolerance, and high throughput
  - Example: Basis for Google's search indexing and data processing pipelines.
- Lustre: Open-source distributed file systems designed for high-performance computing and enterprise storage
  - Lustre: Focused on high-performance parallel file access; widely used in supercomputing environments
- Key features: Data replication, fault tolerance, parallel access, scalability, and distributed metadata management

Fault Tolerance and Recovery

- Checkpointing: Periodically saving the state of an application or system so it can be restarted from the last saved state after a failure, rather than from the beginning
  - Example: Spark writes lineage and intermediate results to storage to recover failed jobs
- Replication: Storing multiple copies of data across different nodes to ensure data availability even if some nodes fail
  - Example: HDFS stores each data block in three separate nodes for redundancy
- Leader election: Mechanism to dynamically choose a leader node among distributed nodes to coordinate tasks or manage system state; critical when leaders fail or leave the system
  - Example: ZooKeeper uses leader election to maintain consistency across distributed services
- Goal: Ensure system reliability, minimize downtime, and recover quickly from failures without data loss

Scalability Considerations

- Horizontal vs Vertical Scaling:
  - Horizontal scaling: Adding more machines or nodes to distribute the workload
  - Vertical scaling: Increasing the capacity (CPU, memory) of an existing machine
  - Example: Adding more servers to a cluster (horizontal) vs upgrading a server's RAM (vertical)
- Load Balancing: Distributing incoming tasks and requests evenly across multiple nodes to prevent overloading any single node and ensure efficient resource utilization
  - Example: Using a load balancer in front of a web server cluster
- Bottlenecks in Distributed Environments: Identifying and mitigating performance bottlenecks caused by factors like network latency, disk I/O limits, uneven data distribution, or coordination overhead
  - Example: A single slow node delaying the completion of a distributed job (straggler problem in MapReduce)
- Goal: Design systems that can scale efficiently with increased data volume and user demand without compromising performance or reliability

KAUST

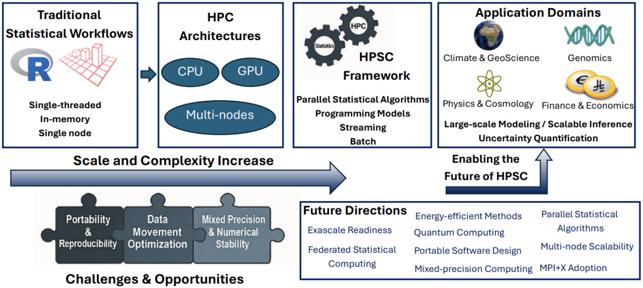
Strong vs Weak Scaling

- Strong Scaling:
  - Measures how the solution time decreases with more processors for a fixed total problem size
  - Ideal strong scaling: doubling processors halves the runtime
  - Challenge: Communication overhead may dominate as processor count increases
  - Example: Solving a 1 million-point matrix with 4, 8, and 16 processors
- Weak Scaling:
  - Measures how the solution time remains constant when the problem size increases proportionally with the number of processors
  - Ideal weak scaling: adding more processors with proportional data keeps runtime constant
  - Challenge: Maintaining efficiency with increased data and processor count
  - Example: Each processor solves a 100k-point subproblem; as processors increase, total size increases too
- Use Case in HPC:
  - Strong scaling is useful for reducing time-to-solution
  - Weak scaling is crucial for solving increasingly large problems efficiently

Table of Contents

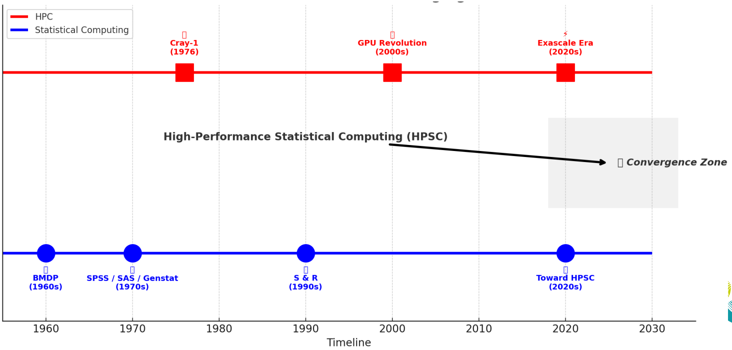
What is High-Performance Statistical Computing (HPSC)?

- High-Performance Statistical Computing (HPSC) is the integration of Statistical Computing (SC) with High-Performance Computing (HPC)
- Enables scalable and fast statistical analysis on modern supercomputing systems
- Formalizes the intersection of statistical methods with computational infrastructure



KAUST

- Statistical Computing (SC) Evolution: BMDP (1960s), SPSS, SAS, R, Julia
- HPC Evolution: Cray-1 to Frontier and El Capitan
- Shift from single-core to multicore, GPUs, and hybrid architectures
- Dataflow vs MPI+X paradigms



Why HPSC?

- Modern data scales overwhelm traditional statistical methods
- Demand for real-time inference, uncertainty quantification, and large-scale simulations
- Need for collaboration between statisticians and HPC engineers
- Our Efforts: Building the HPSC community:
  - `HPSC4Science.org` – hub for resources, events, and publications
  - LinkedIn HPSC Group – networking and engagement

HPSC4Science.org

LinkedIn Group

Challenges in HPSC and Applications

- Adapting sequential statistical algorithms to parallel systems
- Efficient memory and data management at scale
- Ensuring numerical stability in approximation environments (e.g., low-rank mixed-precision)
- Portability and reproducibility across diverse HPC platforms
- Applications Across Domains:
  - Climate Science: Scalable spatial modeling
  - Genomics: Phylogenetic inference, GWAS with GPU acceleration
  - Finance/Economics: Real-time pricing, Bayesian asset modeling
  - Physics: Simulations in cosmology and quantum mechanics.

KAUST

Opportunities and Future Directions

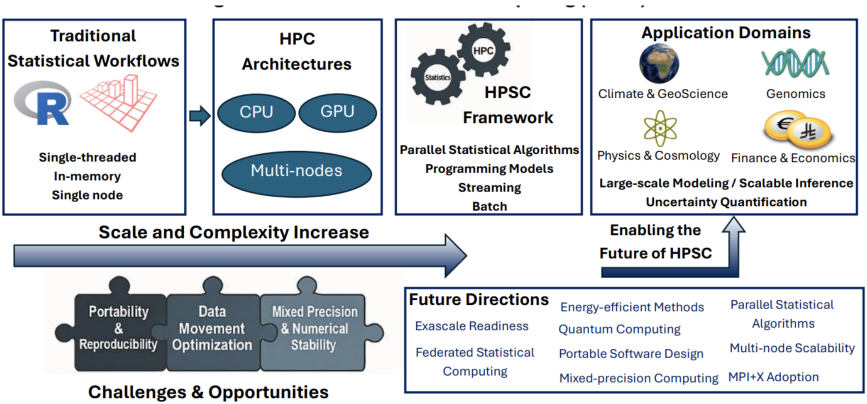


Table of Contents

Take-Home Messages: HPC (1/2)

- HPC = scale: aggregate many CPUs, GPUs, and accelerators to solve problems beyond a single workstation
- Performance is multidimensional: FLOPS, memory bandwidth, latency, interconnect, I/O
- Parallelism is essential: data parallelism + task parallelism across nodes and cores
- MPI+X programming model: MPI for distributed memory, OpenMP/threads/GPU kernels for shared memory
- Data movement dominates cost: optimize for locality, tiling, batching, and communication-avoiding algorithms

Take-Home Messages: HPC (2/2)

- Roofline thinking: balance compute vs. memory to reach peak performance
- Scaling strategies: strong scaling to reduce time-to-solution; weak scaling to handle larger problems
- Reproducibility matters: containers, workflow engines, and environment modules
- Future ready: heterogeneous architectures, exascale systems, energy efficiency
- Key message: HPC is no longer niche — it underpins modern science, engineering, AI, and industry

Q & A

Do it again....