Introduction to High Performance Computing

Costa Rica High Performance Computing School 2025

Esteban Meneses (emeneses@cenat.ac.cr)

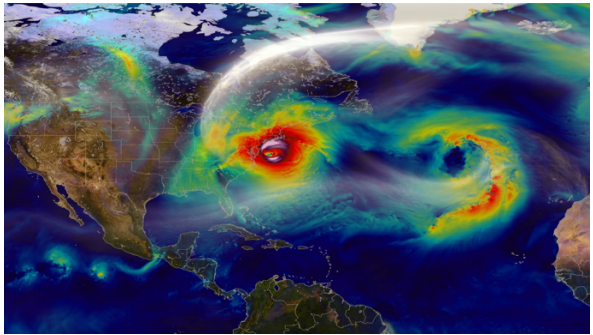
January 27, 2025

Agenda

1 High Performance Computing

High Performance Computing

Also called Supercomputing or Advanced Computing



Hurricane Sandy

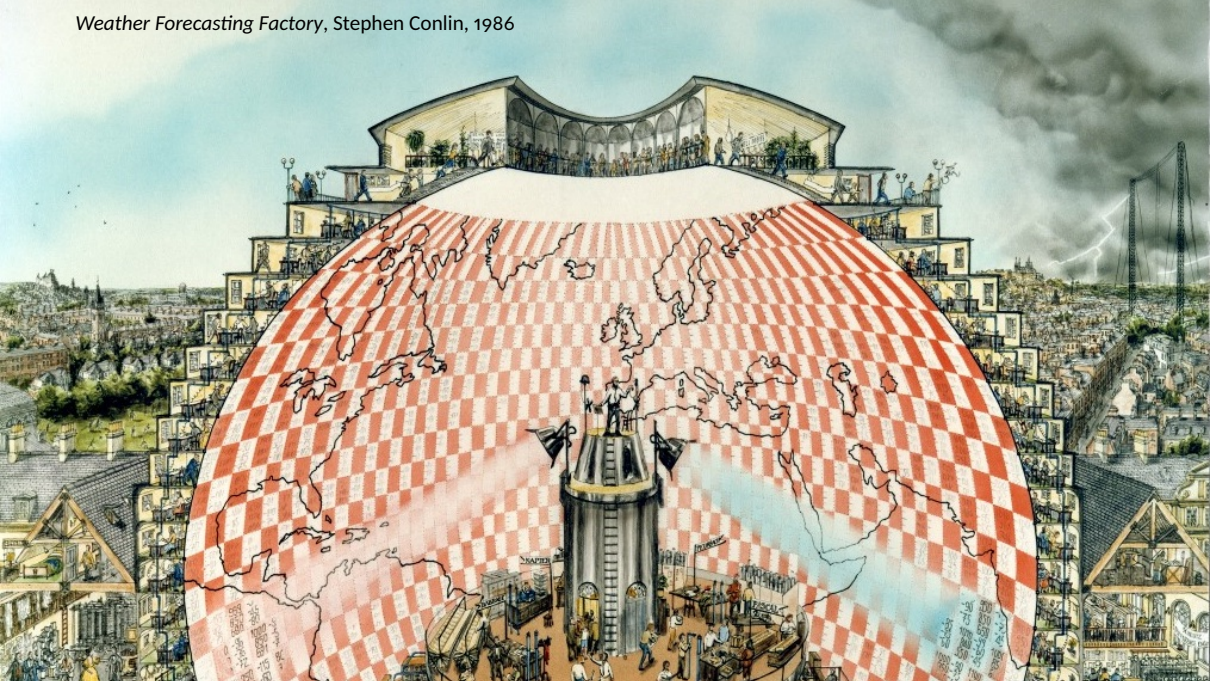
High-Performance Computing, or HPC, is the application of supercomputers to computational problems that are either too large for standard computers or would take too long

National Institute for Computational Sciences

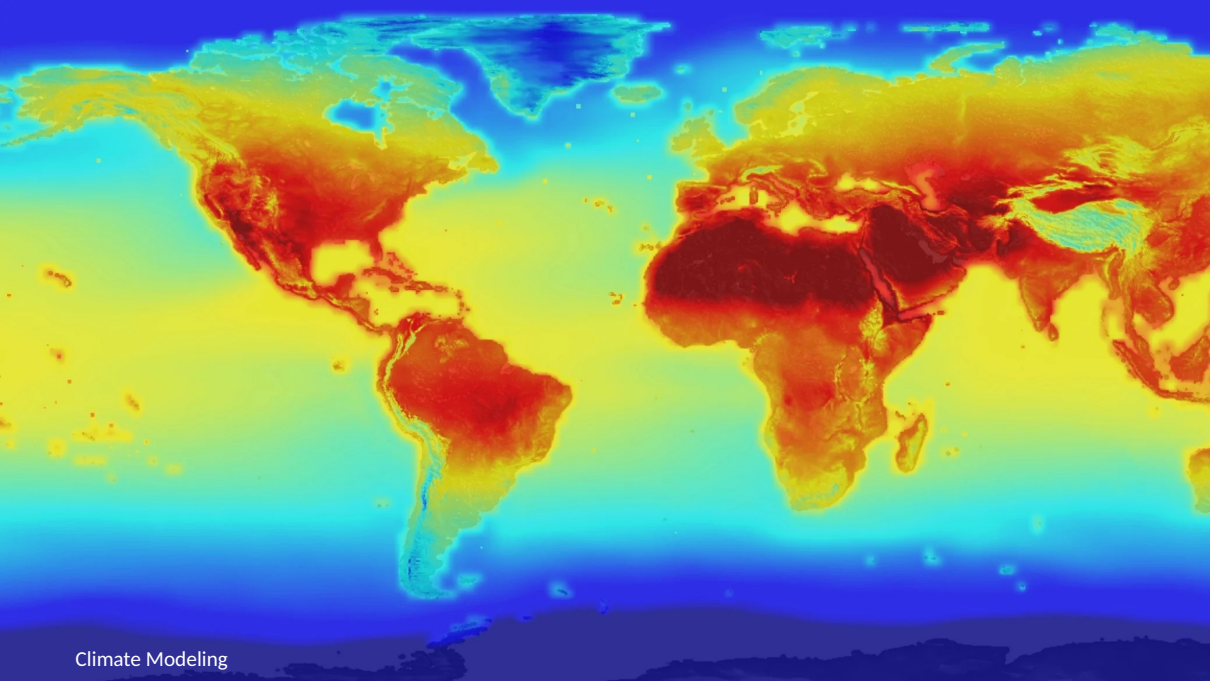
Lewis Fry Richardson

- English mathematician (1881-1953)
- Physicist, pacifist, meteorologist
- Pioneer in using a system of partial differential equations for predicting weather
- *Weather Prediction by Numerical Process* (1922)

Weather Forecasting Factory, Stephen Conlin, 1986



Clementina XXI, Servicio Meteorológico Nacional, Argentina

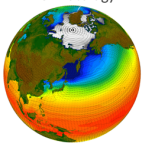


Climate Modeling

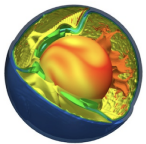
HPC is Ubiquitous

Applications of HPC span many fields

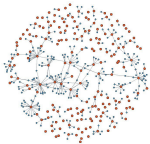
Climatology



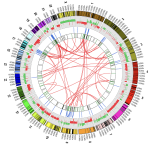
Geodynamics



Social Science

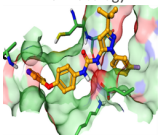


Genomics



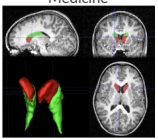
Economics

Pharmacology

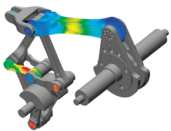


Cosmology

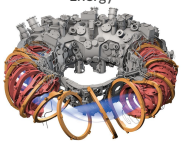
Medicine



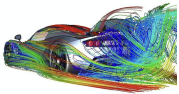
Material Science



Energy



Manufacturing



Entertainment

HPC Empowers Discovery

The third and fourth pillars of science

Telescope

Microscope

Computescope

Theory

Experimentation

Simulation

Data Analysis

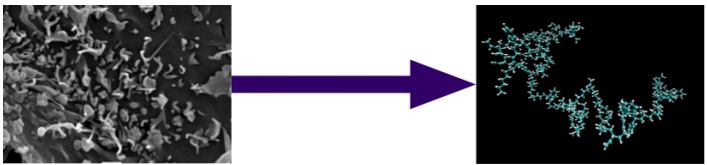
High Performance Computing

Components

Mathematical Models

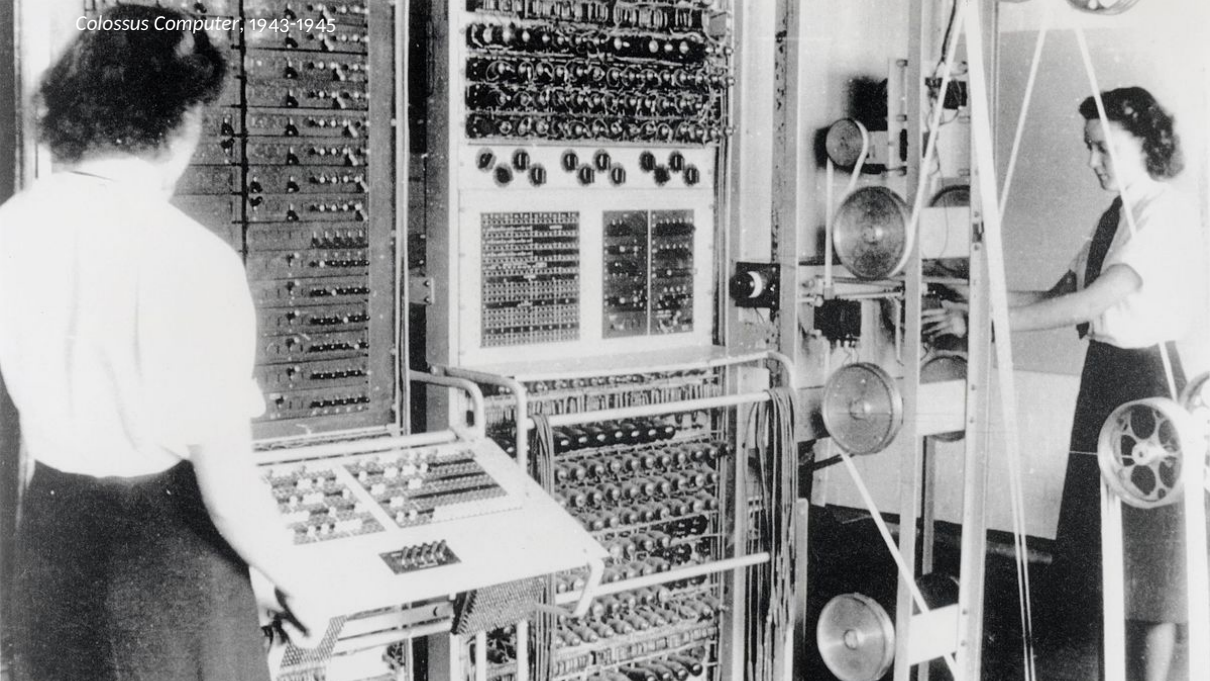
Software

Hardware



Thomas Harold Flowers

- English electrical engineer (1905-1998)
- Worked at the Research Post Office with telephone exchange
- Interacted with Alan Turing to break German codes during WWII
- Designed and built Colossus, the first programmable electronic computer



Colossus Computer, 1943-1945

Seymour Roger Cray

- American electrical engineer (1925-1996)
- Considered as the father of supercomputing
- Worked at Control Data Corporation, designed CDC-6600
- Created Cray Research and Cray Computer Corporation

CDC 6600 Supercomputer, the first supercomputer, 1964

El Capitan, current fastest supercomputer, 2025

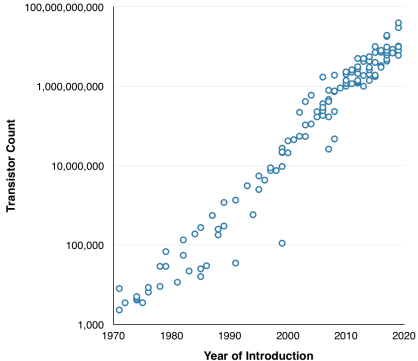
Ranking Supercomputers

Top 500

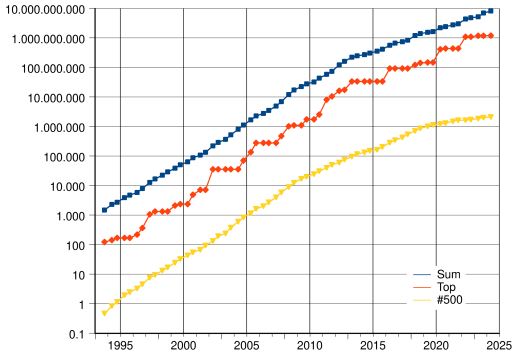
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	

Moore's Law

Exponential growth in transistor density



Fuente de datos: https://en.wikipedia.org/wiki/Transistor_count



Top500 (GigaFLOP)

Graphics Processing Units (GPU)

Energy-efficient floating-point operations

CeNAT

HPC Empowers Innovation

Large impact on industry

Industry	Average Revenue*	Average Cost Saving*
Academic	$9.2	$44.3
Defense	$75.0	$5.3
Government	$1205.7	$112.0
Life Sciences	$160.0	$40.9
Manufacturing	$83.0	$20.2
Oil and Gas	$416.0	$53.7
Telecommunications	$210.7	$30.4
Transportation	$1804.3	$15.6

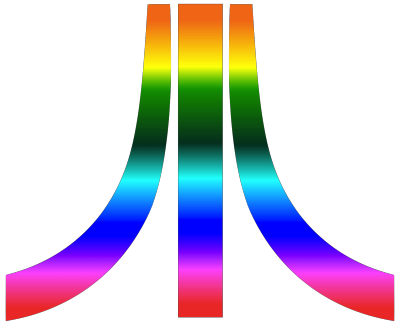
*Per each $ spent on HPC

Economic Models For Financial ROI And Innovation From HPC Investments,

Joseph, Conway, and Norton, 2018.

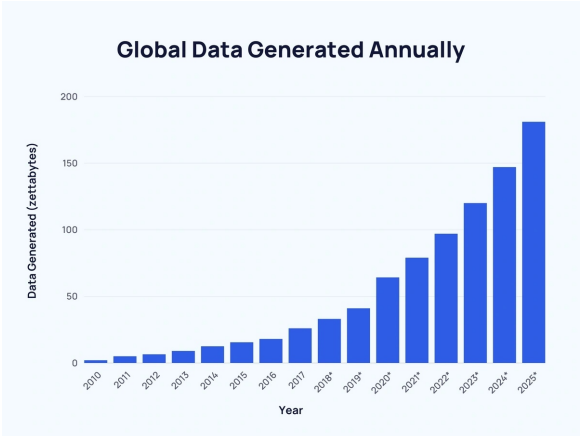
Convergence of Fields

High Performance Computing, Data Science and Artificial Intelligence



Big Data

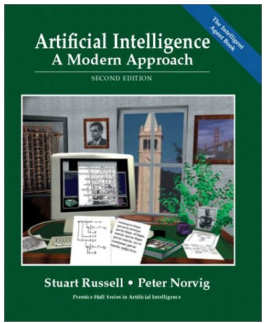
Exponential growth

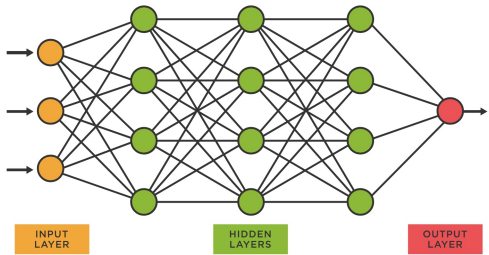


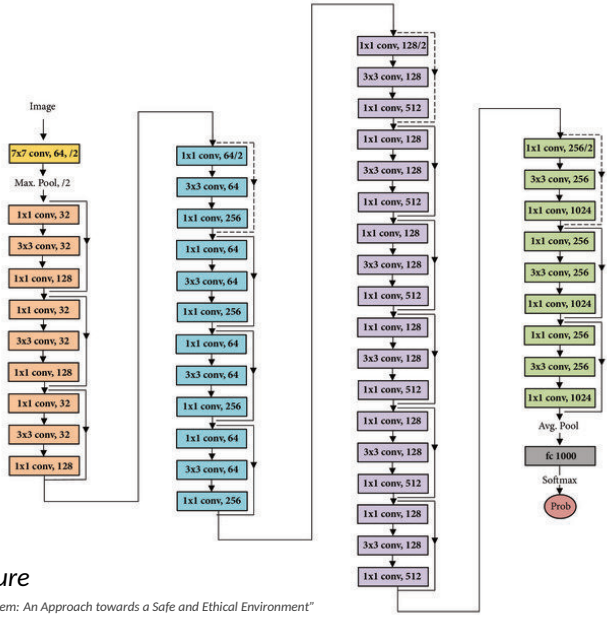
https://www.statista.com/statistics/871513/worldwide-data-created/

Artificial Intelligence in Retrospective

A *modern* approach





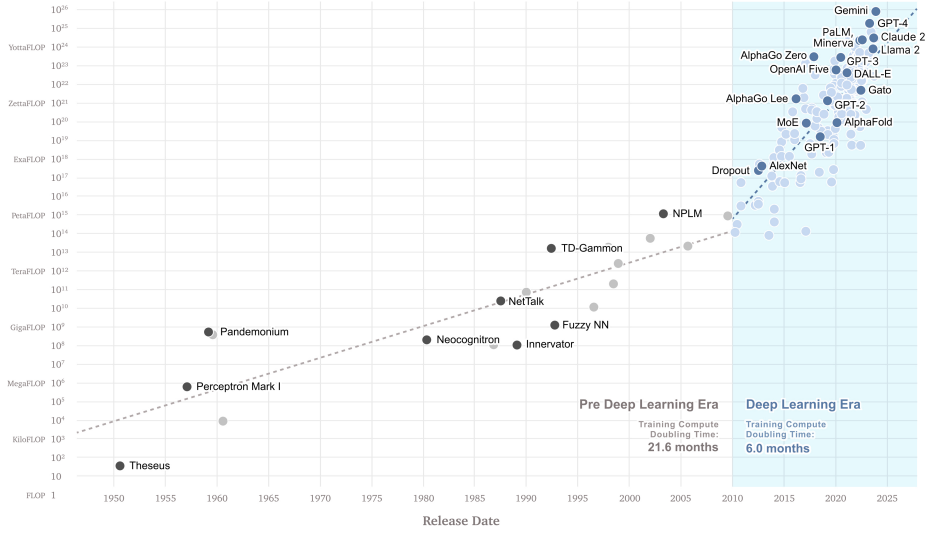


ResNet50 Architecture

Source: "Explicit Content Detection System: An Approach towards a Safe and Ethical Environment"

Compute Used for AI Training Runs

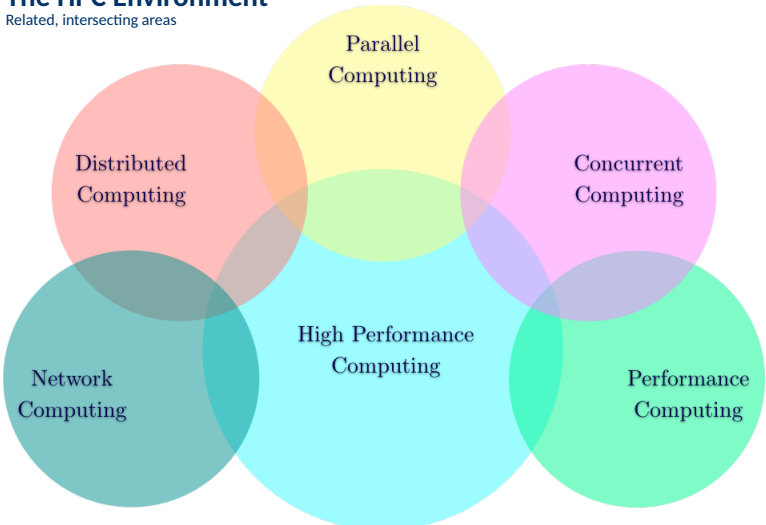
Total compute used to train notable AI models, measured in total FLOP (floating-point operations) | Logarithmic



Source: "Computing Power and the Governance of Artificial Intelligence"

The HPC Environment

Related, intersecting areas



Agenda

2 Parallel Computing

- ▶ High Performance Computing
  - Definition
  - History
  - Applications
- ▶ Parallel Computing
  - Definition
  - Programming

CeNAT

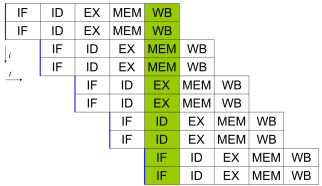
Parallel Computing

Using a collection of computers

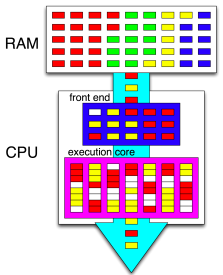
- Parallel computing is the use of multiple processing entities in combination to solve a single problem
- Parallel Computing (PC) versus Distributed Computing (DC)
- The goal of PC is usually to improve performance; DC aims at improve convenience (availability, reliability, security)
- The processing entities in PC are tightly coupled; interaction in DC is loosely coupled

Levels of Parallelism

Two levels are well-known



- Instruction-level parallelism (ILP), simultaneous execution of multiple instructions:
  - Instruction pipelining
  - Out-of-order execution
  - Vector operations
  - Speculation, branch prediction
- Thread-level parallelism (TLP), concurrent execution of multiple threads



Mainstream Parallel Computing

Parallel architecture is commonplace

=

Multicore Processor

+

Graphics Processing Unit

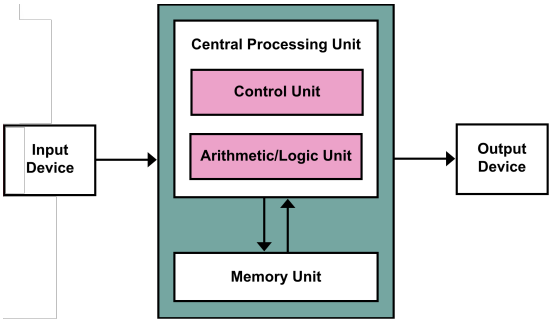
- Two major trends:
  - Processor architects focus on throughput, not clock speed, to improve performance
  - Access to widely available graphic processing units for general processing.
- The reason is heat dissipation and power consumption

John von Neumann

- Hungarian-American mathematician (1903-1957)
- Contributions to mathematics, economics, computer science, and statistics
- Member of Manhattan Project and Institute for Advanced Study
- Proposed a design for a digital computer (EDVAC) in 1945 that later became the von Neumann model
- Introduced cellular automata
- Designed merge sort algorithm

Von Neumann Architecture

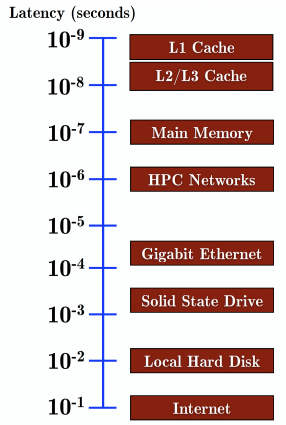
Base of computer structure



- Control unit: decides which instruction gets executed next
- Arithmetic/Logic unit (ALU): executes the instruction
- Memory unit: stores data and instructions

Memory Hierarchy

Levels of storage

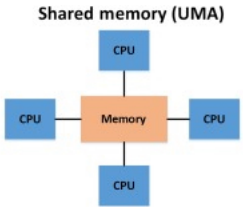


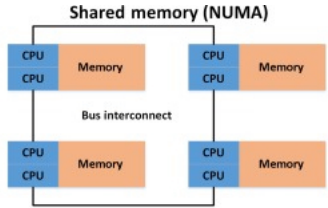
Memory wall:

- Access to registers is orders of magnitude faster than access to memory
- Solutions: caching, prefetching, multithreading

Shared-memory Systems

- Uniform Memory Access (UMA): memory is equally accessible to all processors with the same performance (bandwidth and latency).
- Cache-coherent Non Uniform Memory Access (ccNUMA): memory is physically distributed but appears as a single address space. Performance (bandwidth and latency) is different for local and remote memory access.





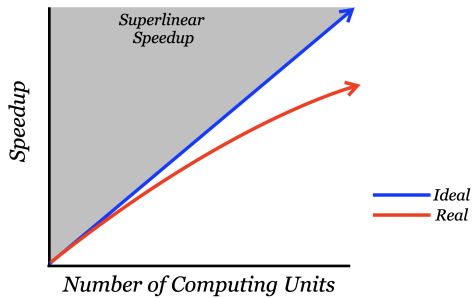
Scalability

Performance of a computer program as more computing units are used

The usual performance measure is execution time (T).
Imagine we are using p computing units:

- Speedup
  $S_p = \frac{T_1}{T_p}$
- Efficiency
  $E_p = \frac{S_p}{p}$
- Parallel Speedup
  $R_p = \frac{T_k}{T_p}$

T_1 is also known as the sequential time.



Strong and Weak Scaling

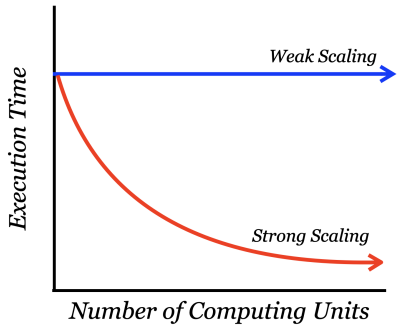
How problem size grows as more computing units are used

Why scaling?

- solve given problem in less time
- solve larger problem in same time
- obtain sufficient memory to solve given problem

Types of scaling:

- *Strong scaling*: problem size remains invariant
- *Weak scaling*: problem size is proportional to number of computing units



Amdahl's Law

The curse of strong scaling

Assume a fraction s of work for given problem is serial, with $0 \leq s \leq 1$, while the remaining portion, $1 - s$, is p-fold parallel:

$$T_p = sT_1 + (1 - s)\frac{T_1}{p}$$

$$S_p = \frac{T_1}{T_p} = \frac{1}{s + \frac{(1-s)}{p}}$$

and hence $S_p \to \frac{1}{s}$ as $p \to \infty$

For example, if serial fraction exceeds 1 percent, then speedup can never exceed 100 for any p.

Gustafson's Law

The bliss of weak scaling

Assume the total serial work of an algorithm is Q, and R is the work performed by each processor in parallel. The total parallel work should grow with p:

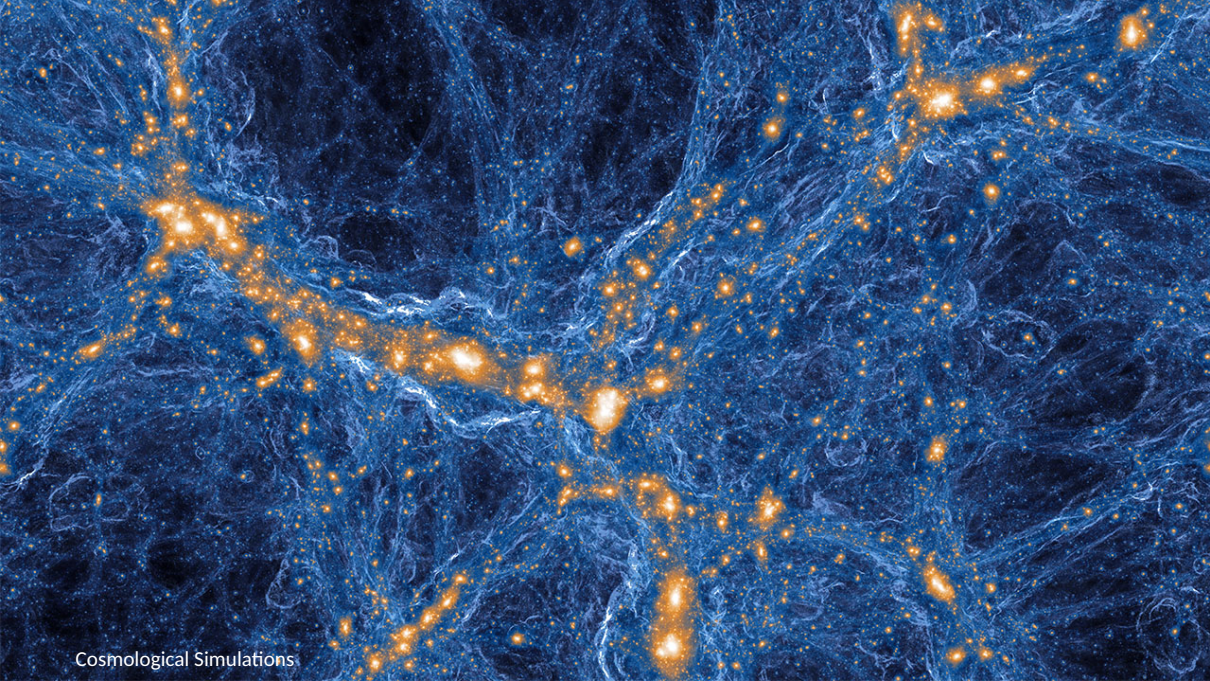
$$T_1 = Q + pR$$

$$S_p = \frac{T_1}{T_p} = \frac{Q + pR}{Q + R}$$

$$\alpha = \frac{Q}{Q + R}$$

$$S_p = \alpha + p(1 - \alpha) = p - \alpha(p - 1)$$

For a small value of α, S_p approaches p. In fact, α can be controlled by increasing problem size.


Cosmological Simulations

CeNAT

Parallelizing Compilers

The holy grail of parallel computing

- Parallelization as an optimization
- A compiler may automatically parallelize a sequential code
- Big quest during the 90s:
  - General parallelization is too hard for a compiler
  - Figuring out all data dependences is impossible at compile time
  - Relative success on more restrictive languages (CoArray Fortran, High Performance Fortran)
  - Vectorization of loops has become mainstream
- The programmer has to be involved in the parallelization of the code.

Parallel Programming Languages

Writing code with concurrent execution

Parallel Languages in the mid 90s

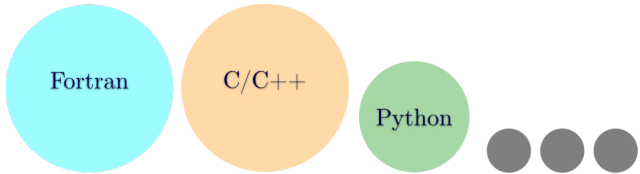
"C* in C	CUMULVS	Java RMI	P-RIO	Quake
ABCPL	DAGGER	javaPG	PAL	Quark
ACE	DAPPLE	JAVAR	Pá-Linda	Quick Threads
ACT++	Data Parallel C	JavaSpaces	Pablo	Sage++
ADDAP	DC++	JIDL	PADE	SAM
Adl	DCE++	Joyce	PADRE	SCANDAL
Admith	DDD	Karma	Panda	SCHEDULE
AFAPI	DICE	Khoros	Papers	SciTL
ALWAN	DIPC	KOAN/Fortran-S	Para++	SDDA
AM	Distributed Smalltalk	LAM	Paradigm	SHMEM
AMDC	DOLIB	Legion	Parafrase2	SIMPLE
Amoeba	DOME	Lilac	Paralation	Sina
AppLeS	DOSMOS	Linda	Parallaxis	SISAL
ARTS	DRL	LiPS	Parallel Haskell	SMI
Athapascan-0b	DSM-Threads	Locust	Parallel-C++	SONiC
Aurora	Ease	Lparx	ParC	Split-C
Automap	ECO	Lucid	ParLab++	SR
bb_threads	Eilean	Maisie	ParLin	Strands
Blaze	Emerald	Manifold	Parlog	Strand
BlockComm	EPL	Mentat	Parmacs	SUIF
BSP	Excalibur	Meta Chaos	Parti	SuperPascal
C*	Express	Midway	pC	Synergy
C**	Falcon	Millipede	pC++	TCGMSG
C4	Filaments	Mirage	PCN	Telegrphos
CarlOS	FLASH	Modula-2*	PCP:	The FORCE
Cashmere	FM	Modula-P	PCU	Threads.h++
CC++	Fork	MOSIX	PEACE	TRAPPER
Charlotte	Fortran-M	MpC	PENNY	TreadMarks
Charm	FX	MPC++	PET	UC
Charm++	GA	MPI	PETSc	uC++
Chu	GAMMA	Multipol	PH	UNITY
Cid	Glenda	Munin	Phosphorus	V
Cilk	GLU	Nano-Threads	POET	ViC*
CM-Fortran	GUARD	NESL	Polaris	Visifold V-NUS
Code	HAsL	NetChaos++	POOL-T	VPE
Concurrent ML	HORUS	Nexus	POOMA	Win32 threads
Converse	HPC	Nimrod	POSYBL	WinPar
COOL	HPC++	NOW	PRESTO	WWWinds
CORBELATE	HPF	Objective Linda	Prospero	XENOOPS
CparPar	IMPACT	Occam	Proteus	XPC
CPS	ISETL-Linda	Omega	PSDM	Zounds
CRL	ISIS	OOF90	PSI	ZPL
CSP	JADA	Orca	PVM	
Cthreads	JADE	P4++	qPC++	

Patterns for Parallel Programming (Tim Matson et al, 2004)

- Based on C/C++, Fortran, Python
- A few dominant libraries:
  - OpenMP: shared-memory
  - MPI: distributed memory
  - Kokkos: performance portability
- Other languages:
  - Unified Parallel C (UPC): partitioned global address space
  - Charm++: parallel objects
  - Fortress, Chapel, X10: DARPA's High Productivity Computer Systems program

HPC Programming Languages

A braid of traditional languages



I don't know what the programming language of the year 2000 will look like, but I know it will be called FORTRAN

Charles Anthony Richard Hoare

CeNAT

Parallel Programming

Considered harder than sequential programming

- Training in computer science is based on von Neumann architecture (sequential)
- Considerations in parallel programming:
  - Synchronization, coordinating access to shared resources
  - Load balancing, distributing computation evenly across processor set
  - Communication, sending data between processors
- The best sequential algorithm to solve a problem may not lead to the best parallel algorithm
- The best parallel algorithm to solve a problem heavily depends on the underlying architecture

Concluding Remarks

3 Conclusion

- HPC is used to solve bigger problems, or to solve known problems faster
- Processing entities are tightly coupled in parallel computing
- The current trend in processor architecture is to move from ILP to TLP
- Parallelism is not always an optimization to sequential code

Acknowledgements

- Lecture notes and discussions with Prof. Laxmikant V. Kalé, Prof. Michael T. Heath, and Prof. David Padua from University of Illinois at Urbana-Champaign

CeNAT

Thank you!
Questions?