

# Introduction to Parallel Programming

Marty Kandes, Ph.D.

Computational & Data Science Research Specialist  
High-Performance Computing User Services Group  
San Diego Supercomputer Center  
University of California, San Diego

SDSC HPC/CI Training Series  
Friday, April 8th, 2022  
2:00PM - 4:00PM PST

# Today

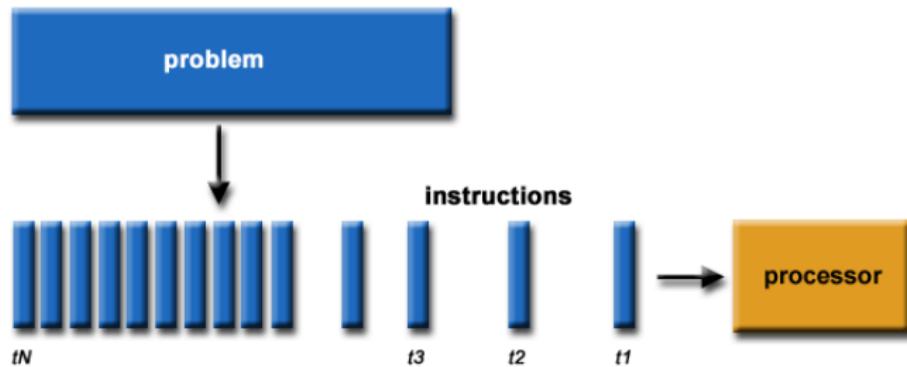
- ▶ Review basic parallel computing concepts
- ▶ Get you up and running on Expanse
- ▶ Embarrassingly parallel problems
- ▶ Thread-based parallel programming with OpenMP and Python

# Parallel Computing Concepts (for Non-Programmers)

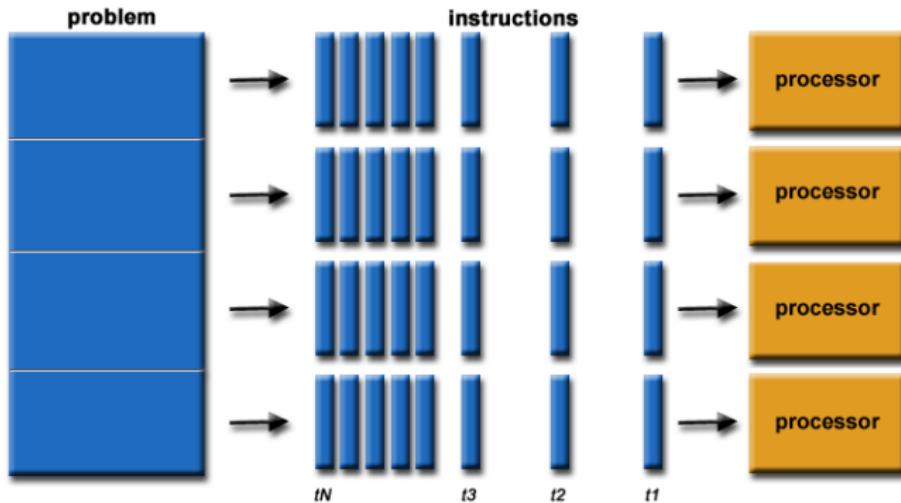
"The Parallel Computing Concepts talk presents a high-level overview of the technologies and software needed by HPC/CI users who need to access HPC systems, and understand the architecture and ideas behind HPC/Parallel computing. The session includes code and figures for presenting scaling data - the right way and the wrong way - and illustrating the limits on scalability imposed by Amdahl's Law."

<https://github.com/sdsc-hpc-training-org/hpc-training-2022>

# Serial Computation

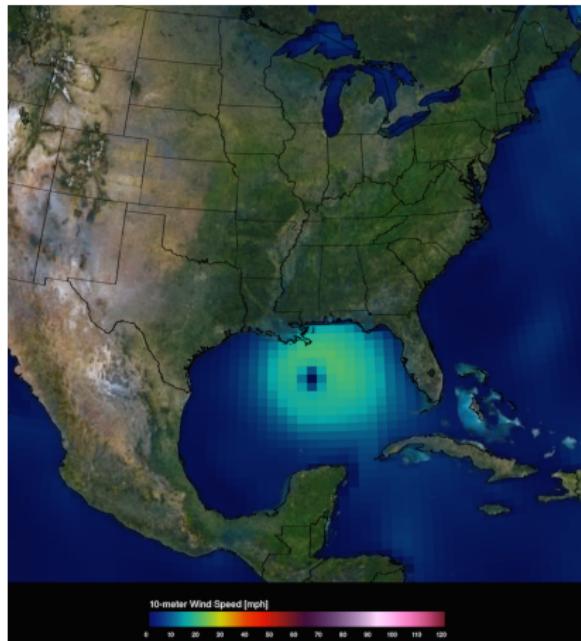


# Parallel Computation



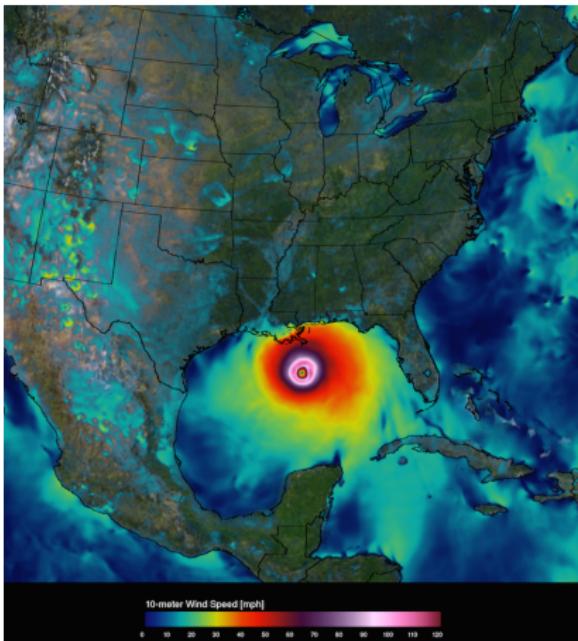
# Why go parallel?

# Speed



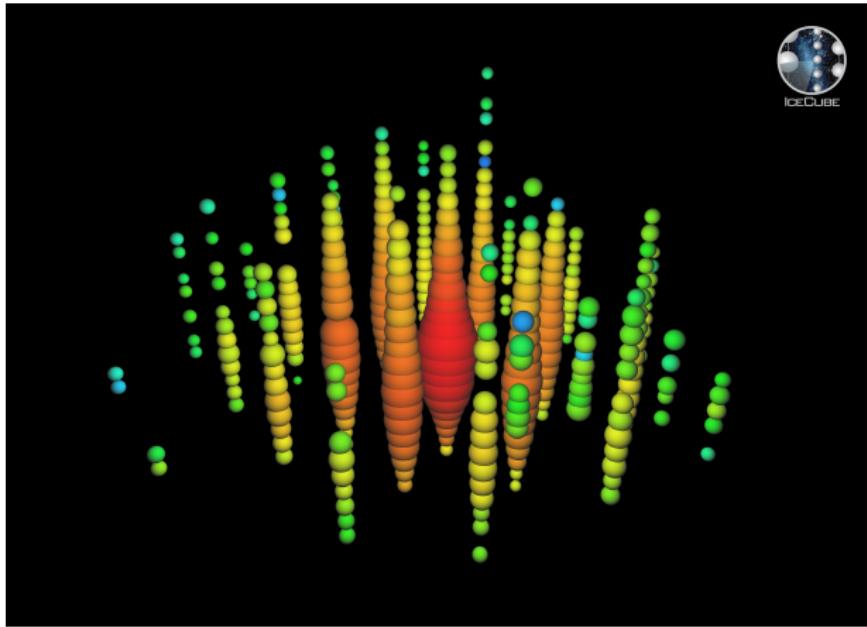
Solve a problem more quickly

# Scale



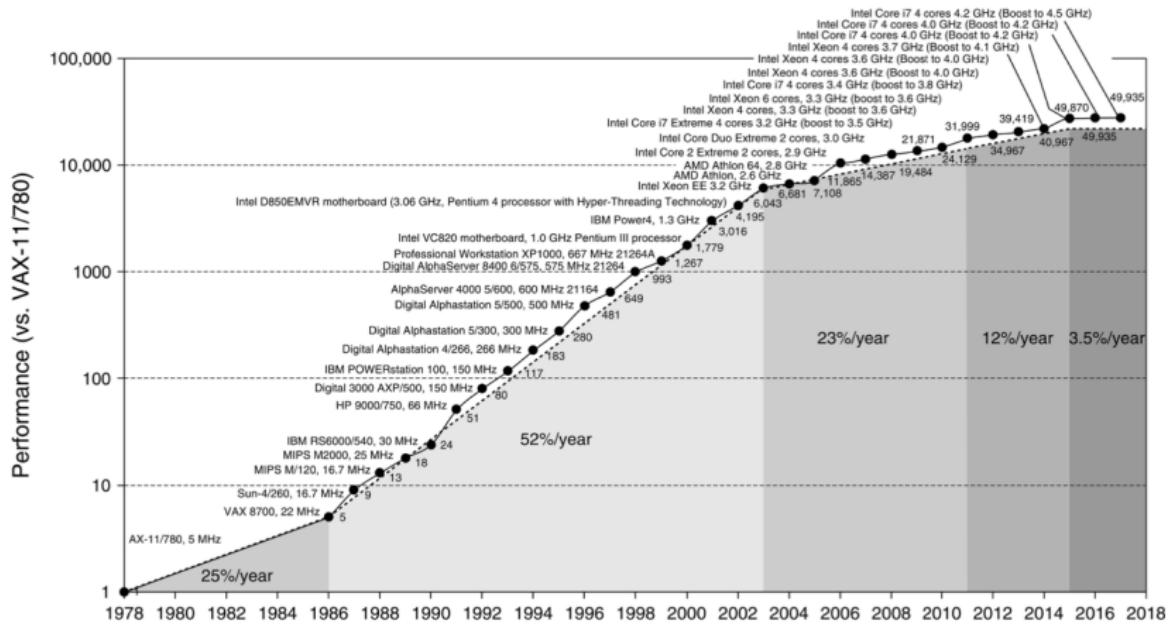
Solve a larger, more complex problem with higher fidelity

# Throughput



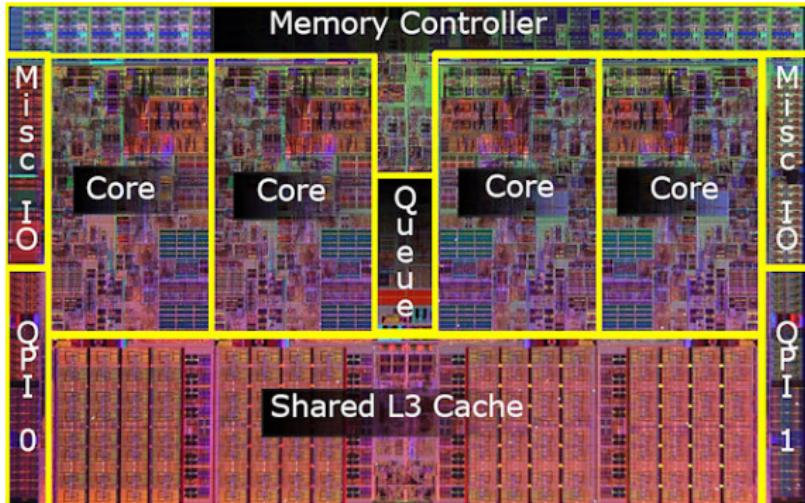
Solve many (simple) problems more quickly

# The End is Near: Single-Core (Uniprocessor) Performance



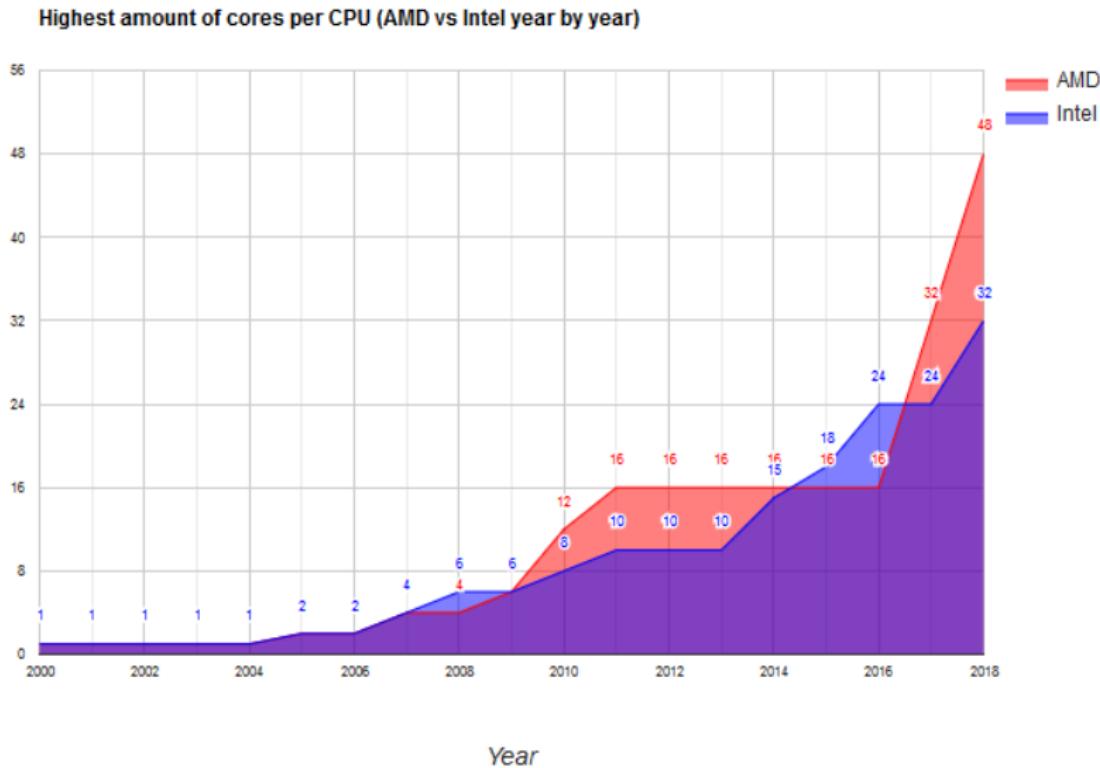
What's the solution?

# Multi-Core Processor

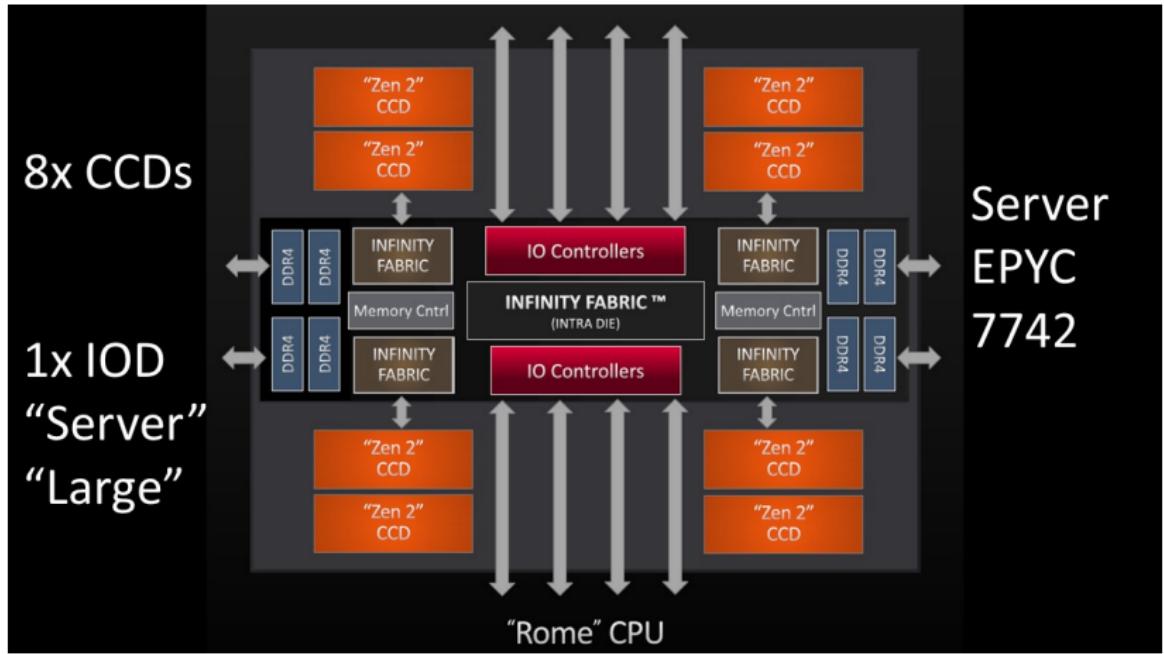


A **multi-core processor** is a computer processor with two or more separate central processing units or cores, each of which reads and executes program instructions, as if the computer had several single-core uniprocessors.

# How many cores do you want with that?



# EXPANSION - AMD EPYC (ROME) 7742

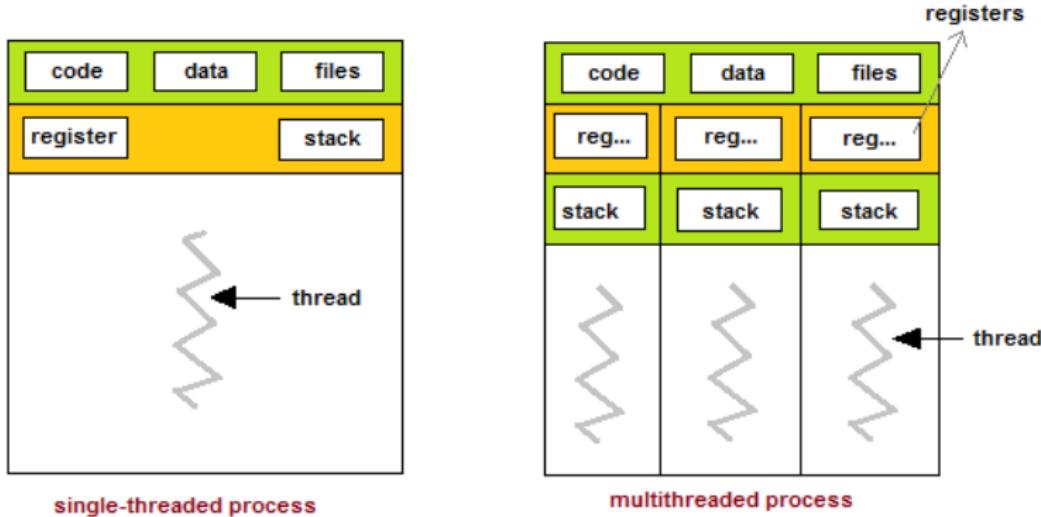


# What is a Process?

CPU[ <span style="background-color: #ffcc00;">11</span> ]	2.0%	Tasks: 16 total, 1 running															
Mem[ <span style="background-color: #ffcc00;">     </span> ]	13/123MB]	Load average: 0.37 0.12 0.04															
Swp[	0/109MB]	Uptime: 00:00:50															
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command						
3692	per	15	0	2424	1204	980	R	2.0	1.0	0:00.24	htop						
1	root	16	0	2952	1852	532	S	0.0	1.5	0:00.77	/sbin/init						
2236	root	20	-4	2316	728	472	S	0.0	0.6	0:01.06	/sbin/udevd --daemon						
3224	dhcp	18	-2	2412	552	244	S	0.0	0.4	0:00.00	dhclient3 -e IF_ME						
3488	root	18	0	1692	516	448	S	0.0	0.4	0:00.00	/sbin/getty 38400						
3491	root	18	0	1696	520	448	S	0.0	0.4	0:00.01	/sbin/getty 38400						
3497	root	18	0	1696	516	448	S	0.0	0.4	0:00.00	/sbin/getty 38400						
3500	root	18	0	1692	516	448	S	0.0	0.4	0:00.00	/sbin/getty 38400						
3501	root	16	0	2772	1196	936	S	0.0	0.9	0:00.04	/bin/login --						
3504	root	18	0	1696	516	448	S	0.0	0.4	0:00.00	/sbin/getty 38400						
3539	syslog	15	0	1916	704	564	S	0.0	0.6	0:00.12	/sbin/syslogd -u s						
3561	root	18	0	1840	536	444	S	0.0	0.4	0:00.79	/bin/dd bs 1 if /p						
3563	klog	18	0	2472	1376	408	S	0.0	1.1	0:00.37	/sbin/klogd -P /var						
3590	daemon	25	0	1960	428	308	S	0.0	0.3	0:00.00	/usr/sbin/atd						
3604	root	18	0	2336	792	632	S	0.0	0.6	0:00.00	/usr/sbin/cron						
3645	per	15	0	5524	2924	1428	S	0.0	2.3	0:00.45	-bash						
F1	Help	F2	Setup	F3	Search	F4	Invert	F5	Tree	F6	SortBy	F7	Nice -F8	Nice +F9	Kill	F10	Quit

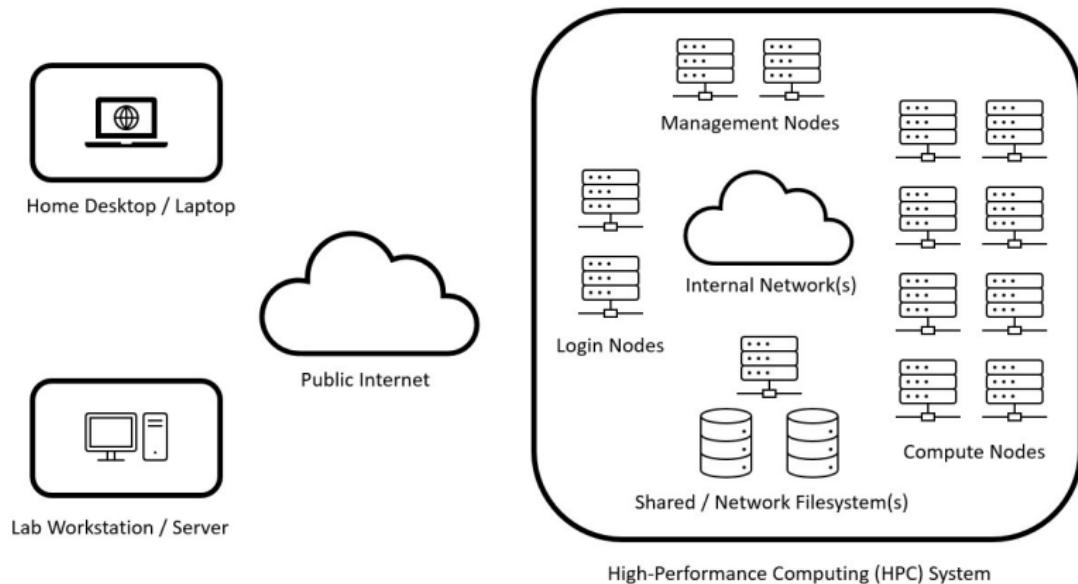
A **process** is the instance of a computer program that is being executed by one or more *threads*.

# What is a Thread?



A **thread** is a subset of program instructions within a process that can be managed and run independently by an operating system's scheduler.

# Your Standard HPC Ecosystem



## Hilbert Matrix

A Hilbert matrix is a square matrix with elements that are unit fractions given by

$$H_{ij} = \frac{1}{i+j-1}$$

For example, the Hilbert matrix of dimension 4 is

$$H = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix}$$

## Exercise: Matrix-Vector Multiplication

If  $H$  is a Hilbert matrix of dimension  $n = 122880$  and  $x = [1 \cdots 1]^T$  is an all-ones vector of the same dimension, compute the resultant vector  $y = Hx$ .

Check your result by computing the sum of the elements of  $y$ . For  $H$  and  $x$  of dimension  $n$ , the sum of the elements of  $y$  is given analytically by

$$\sum_{i=1}^n y_i = n + \sum_{j=1}^{n-1} \frac{n-j}{n+j}.$$

Extra credit: Optimize your code and recompute for both  $n = 122880$  and  $n = 1048576$ . Plot the parallel speedup and efficiency of your code.

# References

- ▶ Introduction to Parallel Computing  
<https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial>
- ▶ OpenMP Tutorial  
<https://hpc-tutorials.llnl.gov/openmp>
- ▶ An Intro to Threading in Python  
<https://realpython.com/intro-to-python-threading>

# Questions?

