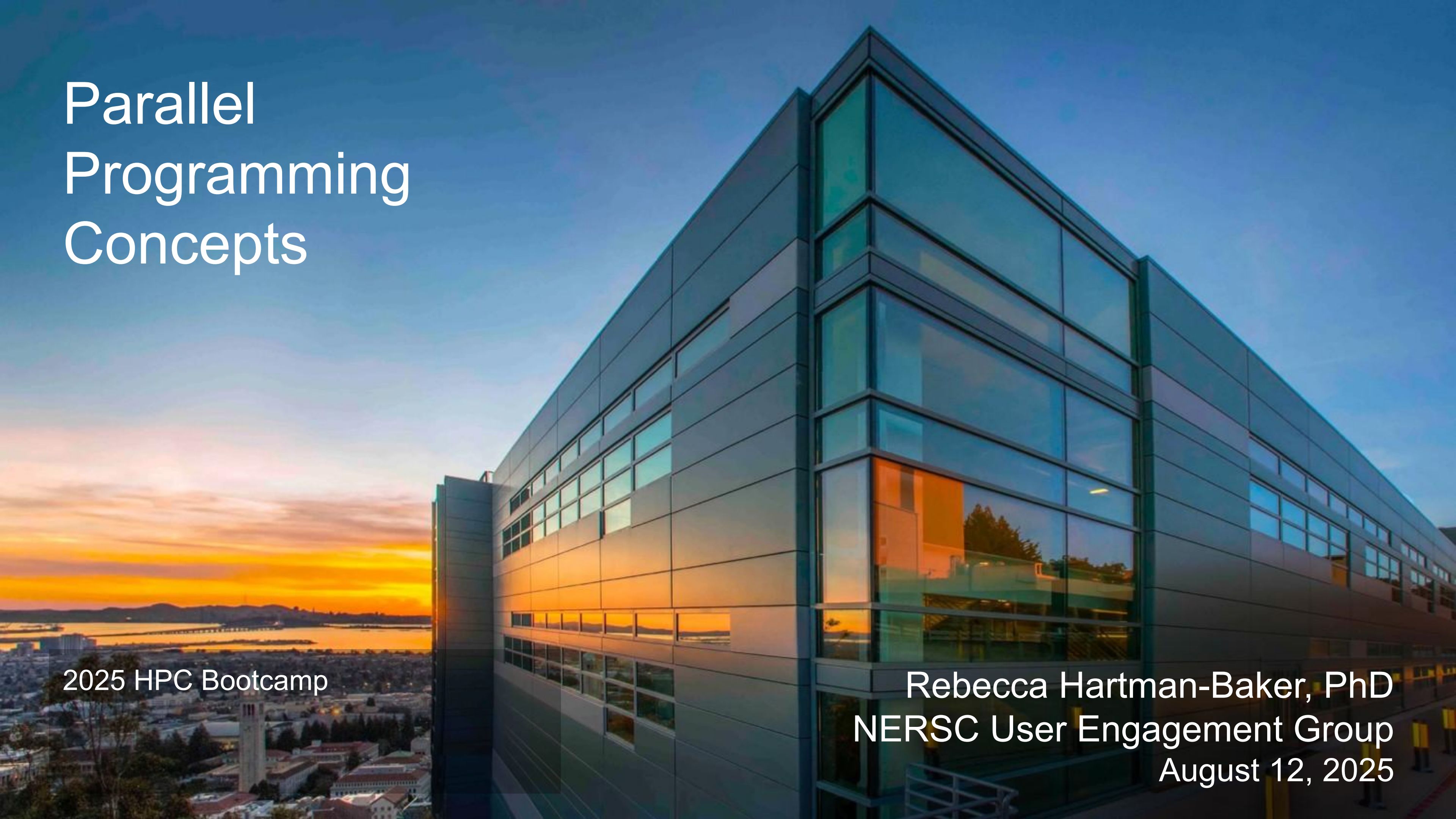


Parallel Programming Concepts

2025 HPC Bootcamp

Rebecca Hartman-Baker, PhD
NERSC User Engagement Group
August 12, 2025

All About Me

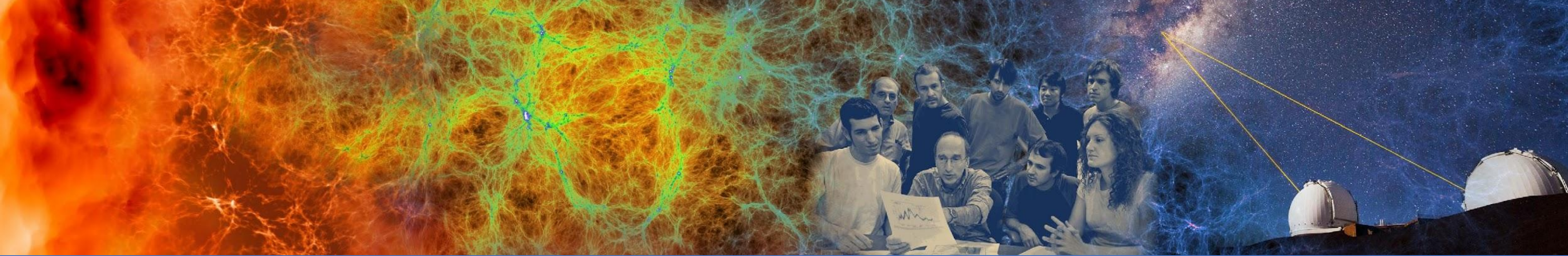
- User Engagement Group (UEG) Lead, NERSC
- World-famous violinist†
- Enthusiastic picker of fruits
- Mom to Vinny (18) & Elena (10)
- Kentucky native, former Illinoisan, honorary Aussie
- Algorithm enthusiast
- PhD, Computer Science, University of Illinois at Urbana-Champaign

Rebecca Hartman-Baker

† Slight exaggeration; I have played publicly in 3 countries on 2 continents

Outline

I. Let's talk parallel!
II. Supercomputer architecture
III. Designing parallel algorithms
IV. Calculating pi in parallel



I. Let’s talk parallel!

What Is Parallelism?

Parallelism is the ability to **perform more than one task at the same time**

- Why is this useful?
    - We can get more done in the same amount of time (or less time)
- For example, if I want to amuse everyone in this room, I can
    - Tell a joke to each person, one by one; or
    - Tell a joke while standing at the podium
- Which would achieve maximal laughs per effort?

Parallelism Is Applicable in Daily Life, Too!

- Applies to any processes that you want to be more efficient at doing
- E.g., doing 7 loads of laundry per week
  - Every day, you could run a load in the washer while you vacuum the house. Then you could shift it to the dryer while folding and putting away the previous load.
  - Weekly, you could take all your laundry to the laundromat, and use 7 washing machines, then 7 dryers, in parallel.

- Both the above examples employ parallelism to increase efficiency

Not Everything Is Parallelizable!

- Parallelizable tasks can be done in any order, because there are no dependencies between tasks
  - For example, washing towels or jeans first does not change the outcome
- On the other hand, some processes must proceed in a particular order due to dependencies
  - For example, you must wash your clothes, *then* dry them, *then* fold them; any other order makes no sense!
  - These are known as **sequential tasks**

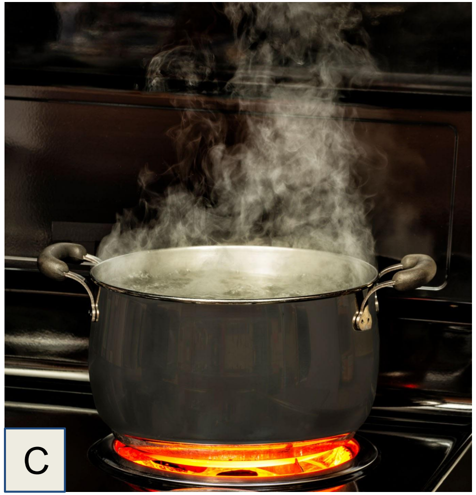
Thinking about Parallelization: Making Dinner

- We're making dinner tonight!
- Our menu: homemade lasagna, salad, and garlic bread
- Which tasks are parallel, and which tasks are sequential?

Parallelization of Lasagna

Tasks:

A. Make sauce
B. Grate cheese
C. Cook noodles
D. Assemble lasagna
E. Bake lasagna



Which tasks are sequential, and which are parallel?

Parallelization of Salad & Garlic Bread

Salad:

- Wash lettuce
- Cut up lettuce
- Wash vegetables
- Cut up vegetables
- Mix lettuce & vegetables
- Dress salad

Garlic bread:

- Cut loaf into slices
- Prepare garlic butter
- Spread garlic butter on slices
- Bake garlic bread

Which tasks are sequential, and which are parallel?

Sequential & Parallel Cooking Tasks

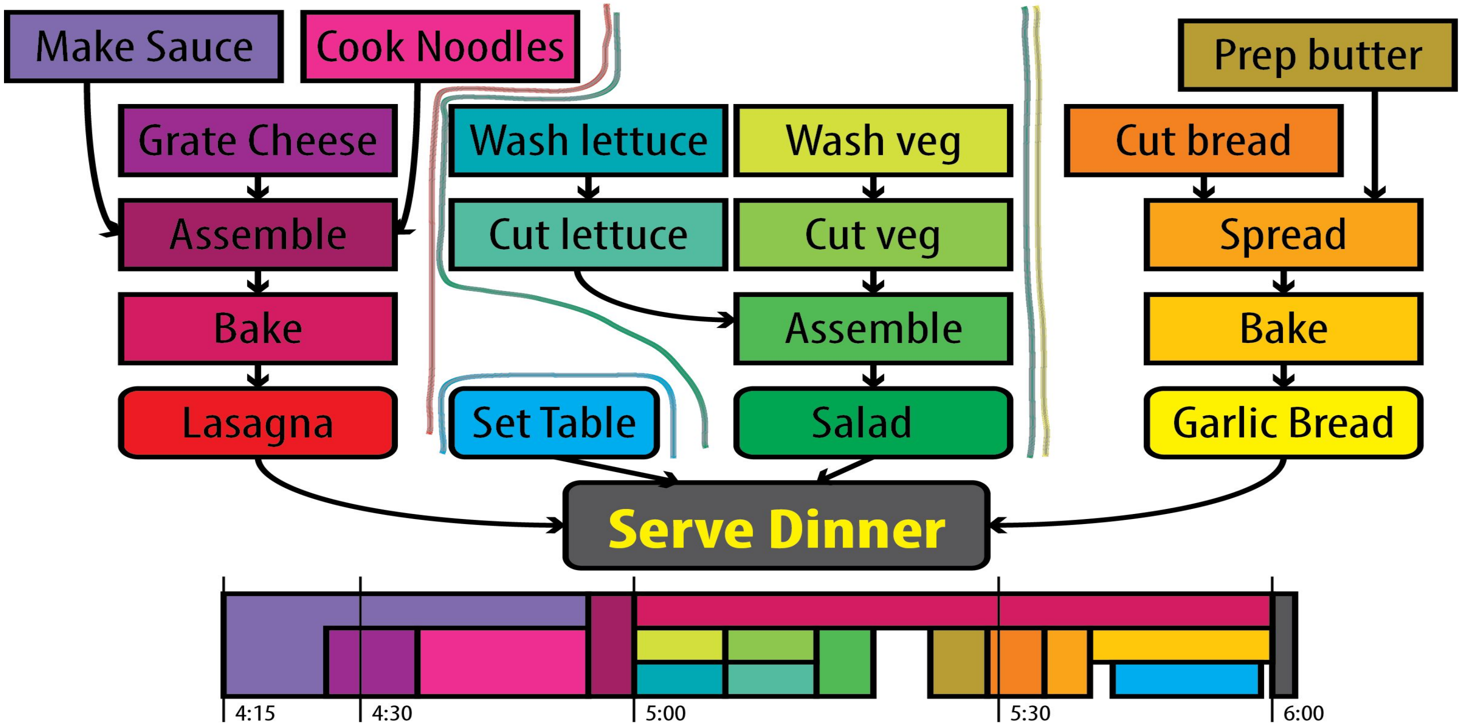
Sequential

- Lasagna: make sauce, then assemble lasagna, then bake lasagna
- Salad: wash veggies, then cut up veggies, then mix salad, then dress salad
- Garlic bread: cut bread, then spread garlic butter, then bake garlic bread
- Entire dinner: cook all foods, then eat them

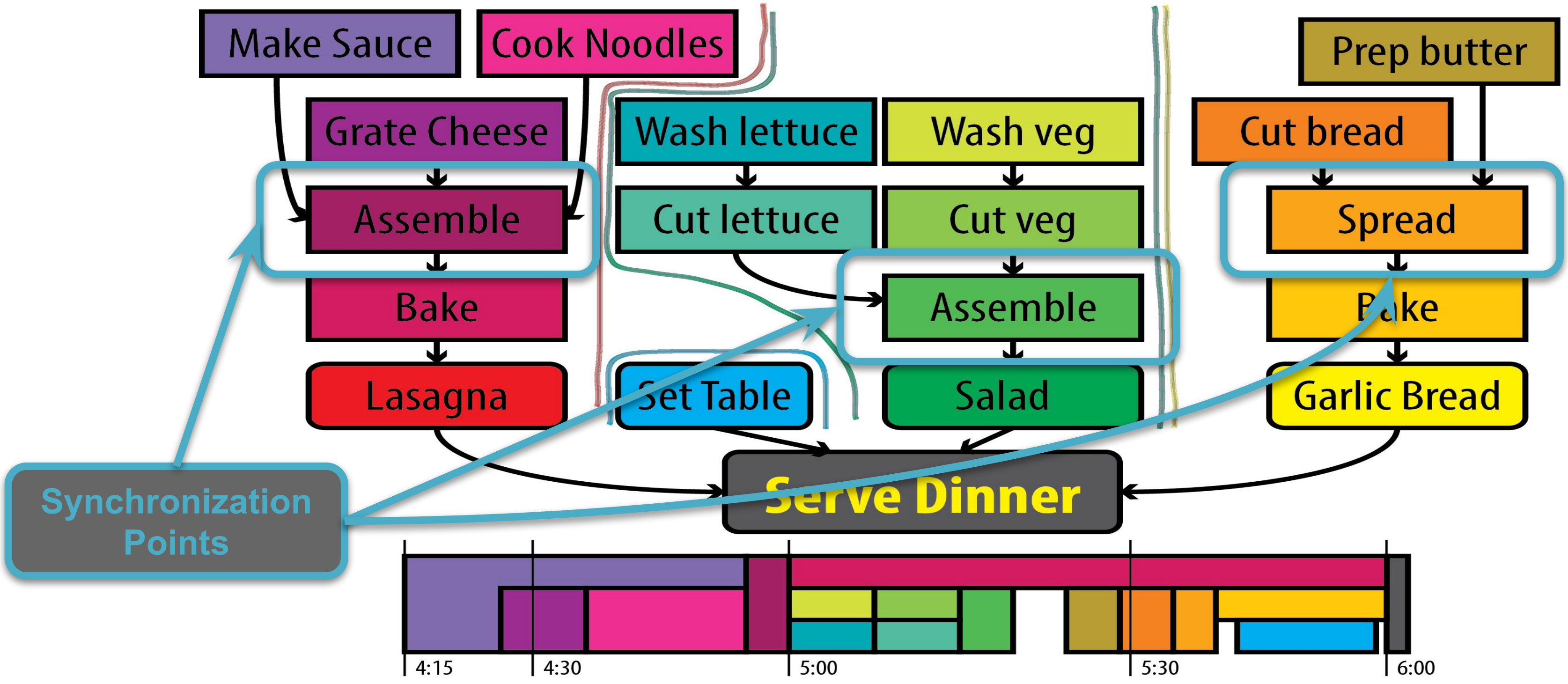
Parallel

- Lasagna: make sauce and grate cheese and cook noodles
- Salad: wash lettuce and wash veggies; cut up lettuce and cut up veggies
- Garlic bread: cut bread and prepare garlic butter
- Entire dinner: making lasagna and making salad and making garlic bread and setting the table

Serial vs Parallel: Graph of Making Dinner



Serial vs Parallel: Graph of Making Dinner

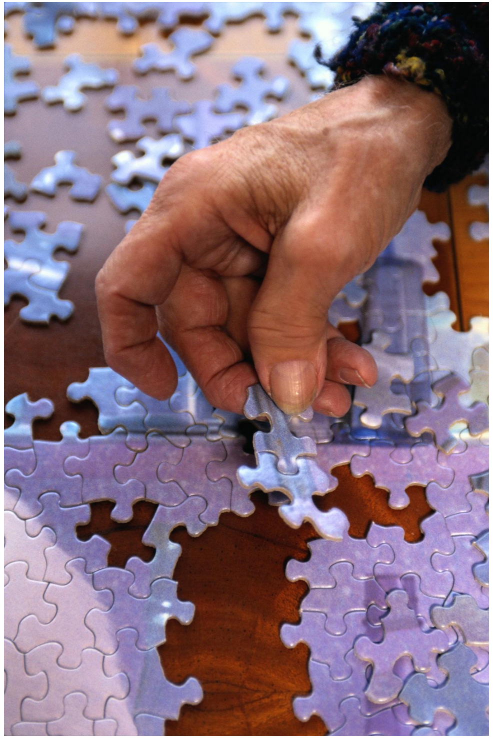


Parallelizing Dinner Preparation

- Add more chefs who can perform parallel tasks simultaneously
- Cook twice as much food, thereby feeding more people (or the same number of people for longer)
- Create a dinner factory, with specialist chefs cooking lasagna, salad, garlic bread

Discussion: Jigsaw Puzzle

- Suppose we want to do a large, N-piece jigsaw puzzle (e.g., $N = 10{,}000$ pieces)
- Assume that the time for one person to complete a puzzle is T hours
  - (Also assume that all people can do puzzles at the same rate)
- How can we decrease walltime to completion?
  - (Literally, the amount of time elapsing on the clock on the wall)



The Challenge

Parallelizing a Jigsaw Puzzle

Suppose we recruited a friend to help, and now we have 2 people sitting at the table

- How long would it take 2 people to complete the puzzle?
  - Is there anything that would make it take longer than expected?
  - Are there any conditions that are different with 2 people vs only one person?
  - Is there anything that we have to do when there are 2 people at the table that we don’t have to do with only one person?
- How long would it take p people at the table to complete the puzzle, where p = 4, 8, …5,000 ?

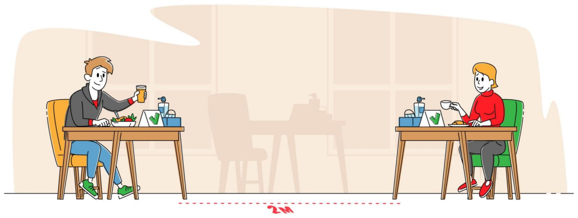
Parallel, but Not Always Pleasingly Parallel…

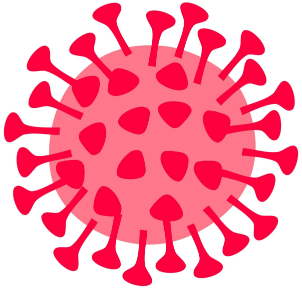
- In the majority of cases, additional overhead is introduced by parallelizing an algorithm
  - Extra setup steps
  - Resource contention
  - Communication
- The overhead introduced limits the efficiency of the algorithm
  - In the limit, our computation takes no time, but we still have this overhead
- Algorithms that can be parallelized with very little (or no) overhead are called “embarrassingly parallel” or “pleasingly parallel”

Parallelizing a Jigsaw Puzzle, COVID Edition

Suppose I set up p tables for p people and put N/p pieces on each table? (Assume that the pieces are pre-sorted so each table has only adjacent pieces.)

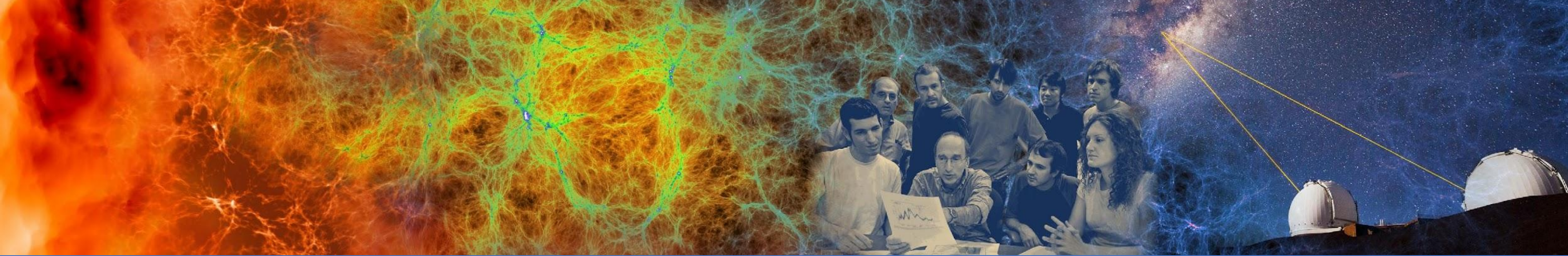
- How long would it take p people to complete the puzzle?
- What overhead would make it take longer than T/p hours?





Distributed Parallelism, Hands-On!

- My advent calendar puzzle is designed for doing 1/24th of the puzzle each day
- Each group gets one day’s puzzle, parallelizing at their table to complete their section
- The complete puzzle is distributed across teams
- Does this simulation live up to our predictions?
- What is the most difficult part of completing this puzzle?



II. Supercomputer Architecture

II. Supercomputer Architecture

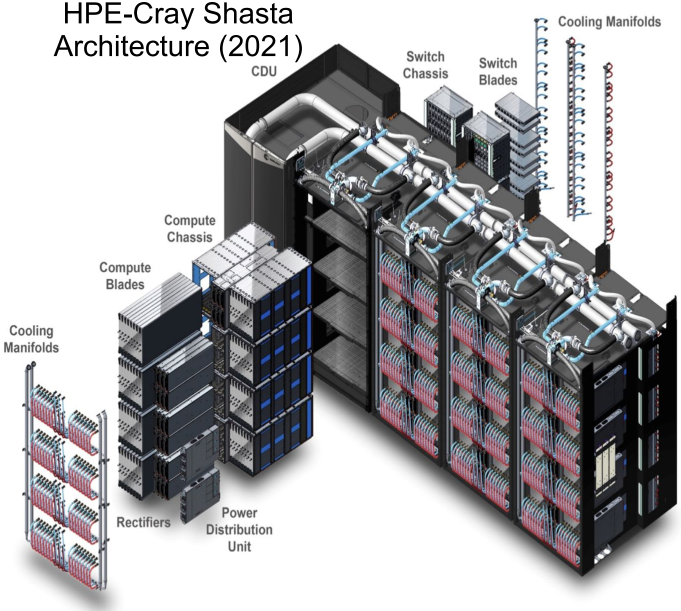
- What is a supercomputer?
- Conceptual overview of architecture

Cray 1 (1976)

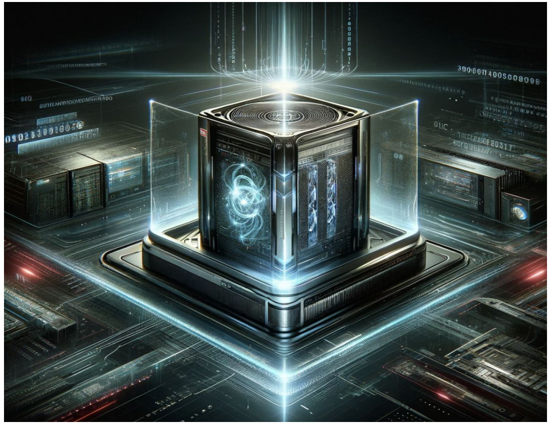
IBM Blue Gene (2005)

Cray XT5 (2009)

HPE-Cray Shasta Architecture (2021)



Future HPC Architecture (2029-???)



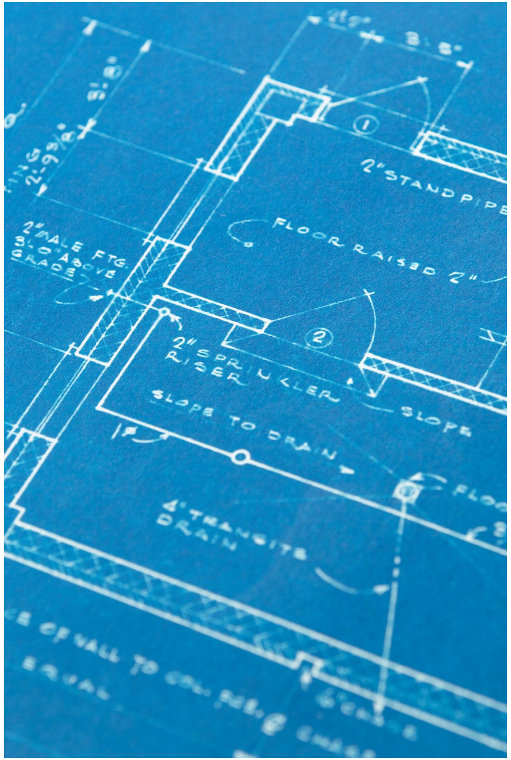
What Is a Supercomputer?

"The biggest, fastest computer right this minute." – Henry Neeman

- Identifying a supercomputer
  - Generally, at least 100 times more powerful than current PC
- This field of study known as supercomputing, high-performance computing (HPC), or scientific computing
- Scientists utilize supercomputers to solve complex problems that often can't be solved in other ways
  - Really hard problems need really LARGE (super)computers

Supercomputing Architectures

- **Symmetric Multiprocessing (SMP) Architecture**
  - Multiple processors or compute cores share a single memory space in common
  - Ideal for parallelizing loops and array operations
  - Use threading (OpenMP, Pthreads) for compute processes
- **Massively Parallel Processing (MPP) Architecture**
  - Many processors, each with their own memory space, perform computations; communications (when needed) across network
  - Ideal for parallelizing independent tasks with little or no overlap
  - Use MPI (Message Passing Interface) for compute processes
- **Cluster Architecture**
  - Connecting multiple standalone compute systems together to work together
  - Standard strategy for building supercomputers today



Puzzling through Supercomputing Architectures

- Jigsaw puzzle = “computations”
- People = processors
- Table = memory

- Everyone at the same table = SMP architecture
- Everyone distributed across tables = MPP architecture
- Tables of groups of people = Cluster architecture

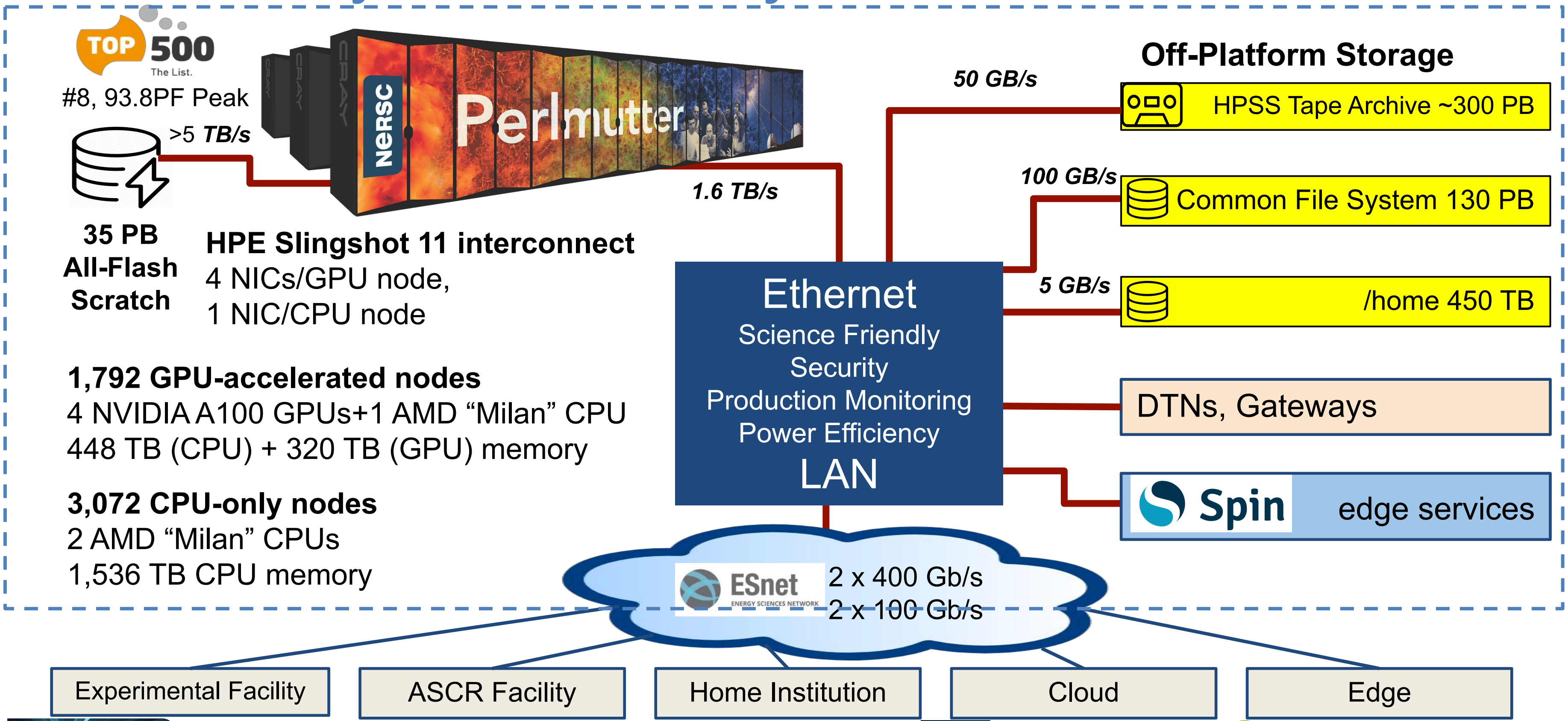
State-of-the-Art Architectures: CPU Clusters

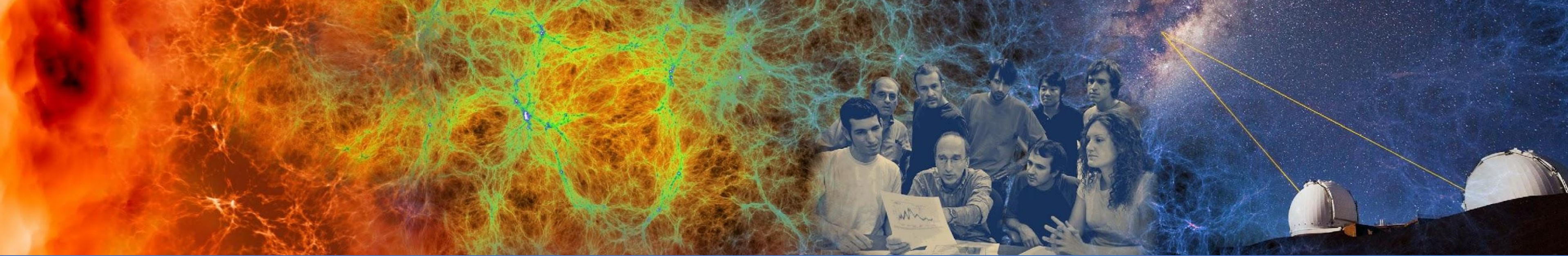
- Typical **CPU-based supercomputers** are built as a cluster of O(100-10,000) nodes (each node comparable to a state-of-the-art workstation)
  - Within a node: SMP architecture
    - Many cores within the CPU(s), sharing a common bank of memory (typically 100s of GB)
  - System scale: MPP architecture
    - Nodes each have their own memory, inaccessible to other nodes
    - Nodes connected to each other with a fast network (but network communication is slow compared to computation speed)

State-of-the-Art Architectures: GPU Clusters

- Typical **GPU-based supercomputers** are also built as a cluster of O(100-10,000) nodes (each node comparable to a state-of-the-art workstation)
  - Within a node: typically one CPU and one or more GPUs
    - The CPU is similar to the CPU cluster – multiple cores, sharing memory
    - The GPU has its own separate memory and an interconnect connecting it to the node's CPU
  - System scale: MPP architecture
    - Each node has its own memory footprint and nodes are connected to each other with a fast network

NERSC Systems Ecosystem





III. Designing Parallel Algorithms

What Is an Algorithm?

- “[A] process or set of rules to be followed in calculations or other problem-solving operations, esp. by a computer: *a basic* ***algorithm for*** *division*.”
- A finite set of rules that precisely defines a sequence of operations
- A set of operations that can be simulated by a Turing-complete system
  - (Most programming languages, including C/C++/Fortran/Python, are Turing complete)
- Algorithms are a deep and rich topic that you could spend multiple lifetimes learning about!

Considerations for Parallelizing an Algorithm

- **Complexity:** how does the time to solution grow as a function of problem size?
  - For a problem of size n, does it grow linearly, quadratically ($O(n^2)$), exponentially ($O(2^n)$)...?
  - Some algorithms with higher complexity may be more parallelizable (and therefore more feasible at large problem sizes)
- **Dependencies:** are there components in the algorithm that depend on other components?
  - Data dependencies, e.g., need value of $f(x \pm \Delta x)$ to compute $f(x)$
  - Sequential dependencies, e.g., need data from step j before starting step $j+1$
  - Algorithmic dependencies, e.g., need to complete subroutine before proceeding to next instruction
- **Performance:** how does the algorithm perform on idealized problems (lower bound) or on adversary-selected problems (upper bound)? What can parallelism do for you?

Algorithm for Parallelizing Algorithms: PCAM

- **Partition:** Decompose the problem into fine-grained tasks to maximize the potential for parallelism
- **Communication:** Determine the communication pattern among tasks
- **Agglomeration:** Combine into coarser-grained tasks, if necessary, to reduce communication requirements or other costs
- **Mapping:** Assign tasks to compute processes (e.g., MPI processes or threads), subject to the tradeoff between communication cost and concurrency

Step 1: Partition

- Find the finest-grained tasks in the algorithm
  - Don't worry about practicality at this point
- Common strategies:
  - **Domain decomposition:** subdivide geometric domain
  - **Functional decomposition:** subdivide system into components
  - **Independent tasks:** divide into embarrassingly parallel tasks
  - **Array parallelism:** simultaneous operations on array entries
  - **Divide-and-conquer:** recursively divide into tree-like hierarchy of subproblems
  - **Pipelining:** break problem into sequence of stages for each object in sequence

Step 2: Communication

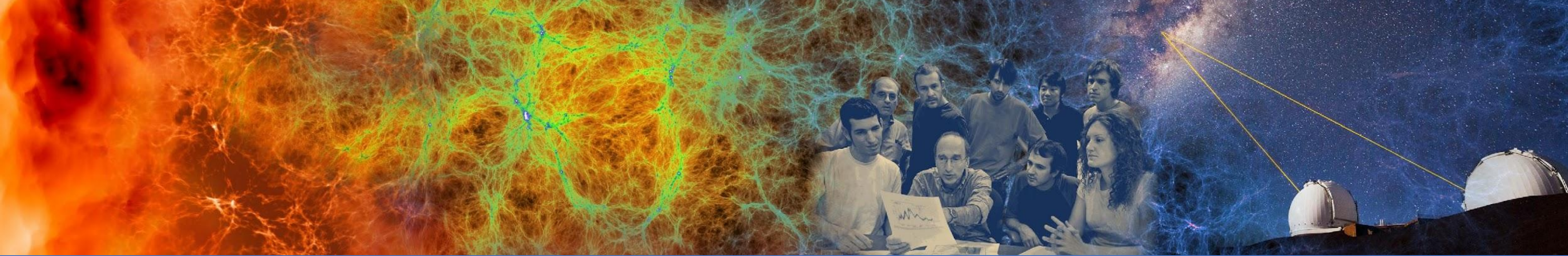
- Determine the communication pattern among tasks
- Sometimes helpful to draw a graph with tasks for nodes and communications for edges
- Dependencies and synchronization points mean communication
- Watch out for obvious manager-worker patterns and other bottlenecks, and figure out whether they can be eliminated

Step 3: Agglomeration

- Combine into coarser-grained tasks, if necessary, to reduce communication requirements or other costs
- Agglomerate dependencies to improve parallelism, while keeping an eye on load balancing
- Larger tasks can reduce communication but may also reduce potential concurrency

Step 4: Mapping

- Assign tasks to compute processes, subject to the tradeoff between communication cost and concurrency
    - How will mapping impact the algorithm's performance at various process and problem sizes?
    - Can you exploit the structure of the problem for mapping?
    - Sometimes random mapping performs better than structured mapping, by avoiding communication hotspots



IV. Computing π

Computing π

- We want to compute π
- One method: Method of Darts†
- Based on the principle that the ratio of the area of a square to the area of an inscribed circle is proportional to π

†This is a TERRIBLE way to compute pi! Don't do this in real life!!! (See Appendix for better ways)

"Picycle" by Tang Yau Hoong, from http://www.flickr.com/photos/tangyauhoong/5609933651/sizes/o/in/photostream/

Method of Darts

- Imagine a dartboard of radius R inscribed within a square
- Area of circle = πR^2
- Area of square = $(2R)^2 = 4 R^2$
- $\frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi R^2}{4R^2} = \frac{\pi}{4}$

“Dartboard” by AndyRobertsPhotos, from http://www.flickr.com/photos/aroberts/2907670014/sizes/o/in/photostream/

Method of Darts: Conceptual Algorithm

- Calculate ratio of areas to determine π
- How do we find the areas?
  - Suppose we threw darts (completely randomly) at a square with an inscribed dartboard
  - Count # darts landing within the circle and total # darts landing within the square
  - The ratio of these numbers gives approximation to π
  - The quality of the approximation increases with the # of darts thrown
    - This algorithm is exponential in complexity: for one additional digit of precision, we need to throw 100x more darts

Parallelizing the Method of Darts

- What tasks must be performed sequentially?
- What tasks are independent of each other?
- Applying PCAM

Method of Darts cake in celebration of Pi Day 2009, Rebecca Hartman-Baker

Step 1: Partition

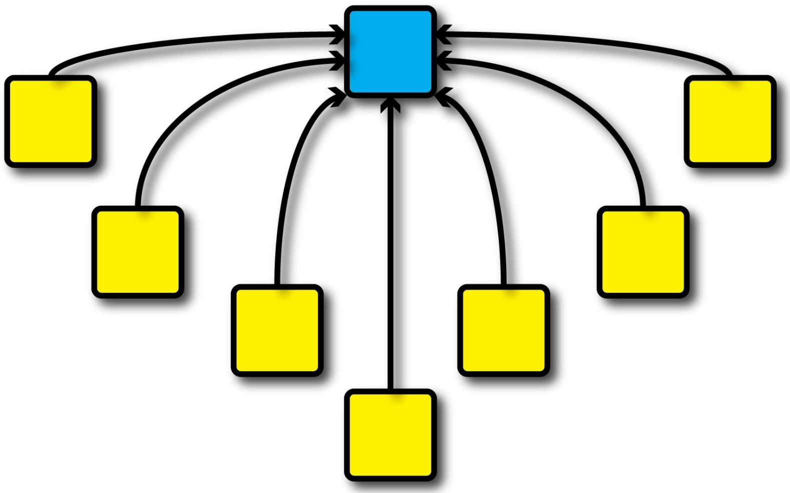
“Decompose the problem into fine-grained tasks to maximize the potential for parallelism”

- Finest grained task: throw of one dart
- Each throw independent of all others
- If we had a huge computer, we could assign one dart throw to each process

Step 2: Communication

“Determine the communication pattern among tasks”

- To calculate π, we need to tally up the results in a centralized location
- Each process throws dart(s) then sends the results to a manager process
- This type of algorithm is known as a “manager/worker algorithm”



Step 3: Agglomeration

“Combine into coarser-grained tasks, if necessary, to reduce communication requirements or other costs”

- To get a good value of π, we will need millions of dart throws
- We don’t have millions of compute processes available
- Furthermore, even if we did, cost of communication would outweigh the cost of throwing the dart
- Solution: divide up the number of dart throws evenly amongst compute processes, so each one does a share of the work

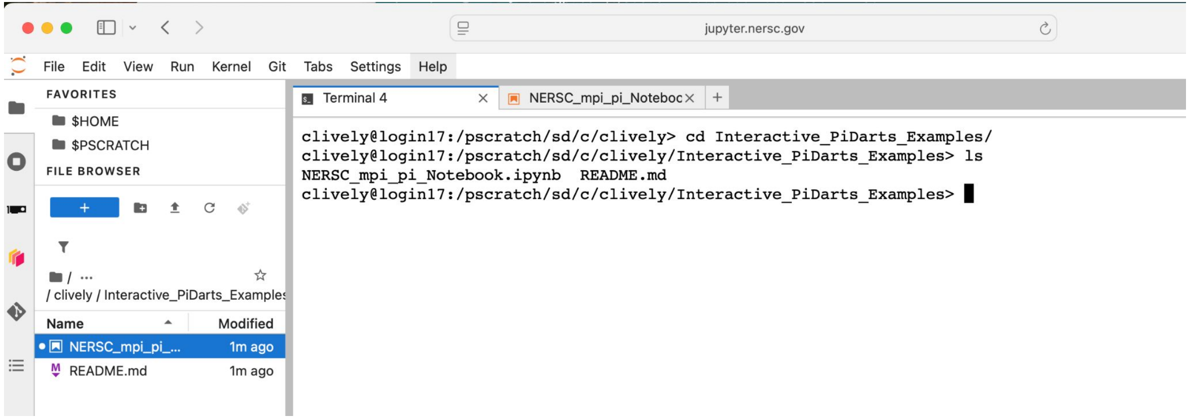
Step 4: Mapping

“Assign tasks to compute processes, subject to the tradeoff between communication cost and concurrency”

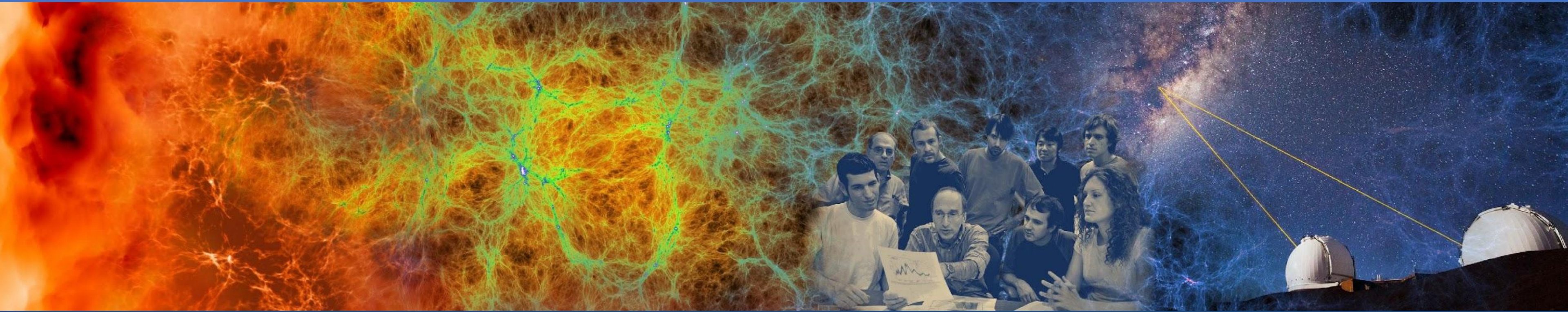
- This is a manager-worker algorithm as we saw in step 2
- Assign the role of manager to compute process number 0
- Number 0 will receive tallies from all the other processes, and compute the final value of π
- Every process, including the manager, will perform an equal share of dart throws

Hands-on: Method of Darts in a Jupyter Notebook

- cd $PSCRATCH
- git clone https://github.com/NERSC/Interactive_PiDarts_Examples.git
- cd Interactive_PiDarts_Examples/



Appendix: Better Ways of Computing π



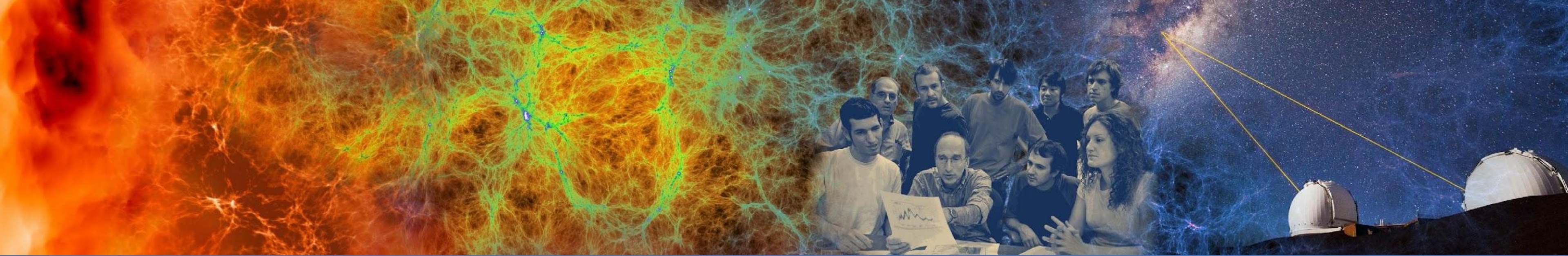
NERSC BERKELEY LAB U.S. DEPARTMENT of ENERGY Office of Science

Better Ways of Computing π

- The Method of Darts is a TERRIBLE way to compute π
  - Accuracy proportional to the square root of the number of dart throws
- Many better alternatives:
  - Look it up on the internet, e.g, 100,000 Digits of Pi
  - Compute with the BBP (Bailey-Borwein-Plouffe) formula

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

- For less accurate computations, try your programming language's constant, or quadrature or a power series expansion



Resources

PCAM Parallel Algorithm Design

- Ian Foster, Designing and Building Parallel Programs
- Michael Heath, Parallel Algorithm Design from CS554, Parallel Numerical Algorithms, University of Illinois at Urbana-Champaign