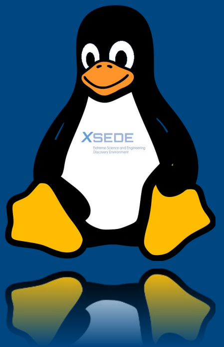
XSEDE

Extreme Science and Engineering
Discovery Environment

Estimating Pi in Parallel with MPI

Presented by:

- Kate Cahill, Ohio Supercomputer Center (kcahill@osc.edu)
- Je’aime Powell, Texas Advanced Computing Center (jpowell@tacc.utexas.edu)



XSEDE Code of Conduct

- XSEDE has an external code of conduct for XSEDE sponsored events which represents XSEDE's commitment to providing an inclusive and harassment-free environment in all interactions regardless of gender, sexual orientation, disability, physical appearance, race, or religion. The code of conduct extends to all XSEDE-sponsored events, services, and interactions.
- **Code of Conduct:** https://www.xsede.org/codeofconduct
- Contact:
  - Event organizer: Rosalia Gomez (rosie@tacc.utexas.edu)
  - XSEDE ombudspersons:
    - Linda Akli, Southeastern Universities Research Association, (akli@sura.org)
    - Lizanne Destefano, Georgia Tech (lizanne.destefano@ceismc.gatech.edu)
    - Ken Hackworth, Pittsburgh Supercomputing Center, (hackworth@psc.edu)

Goals for this session

Work with Monte Carlo calculations using loops and parallel processing

Use the batch system on Stampede2 to run jobs

Manage job scripts

Review job output for errors

Code Review

```
def throw_darts(n):
    darts = 2*np.random.random((n,2)) - 1
    return darts

def in_unit_circle(p):
    return np.linalg.norm(p,axis=1)<1

def estimate_pi(n):
    d_arr = throw_darts(n)
    h_arr = in_unit_circle(d_arr)
    return 4 * np.sum(h_arr) / n
```

Loops

```
if __name__ == '__main__':
    n_ests = 5
    n = 10000
    N = n_ests*n

    start = time.time()

    for i in range(n_ests):
        pi_ests = []
        pi_ests.append(estimate_pi(n))



    pi_est_mean = np.mean(pi_ests)
    pi_est_std  = np.std(pi_ests)
```

Loops - Runs vs Darts

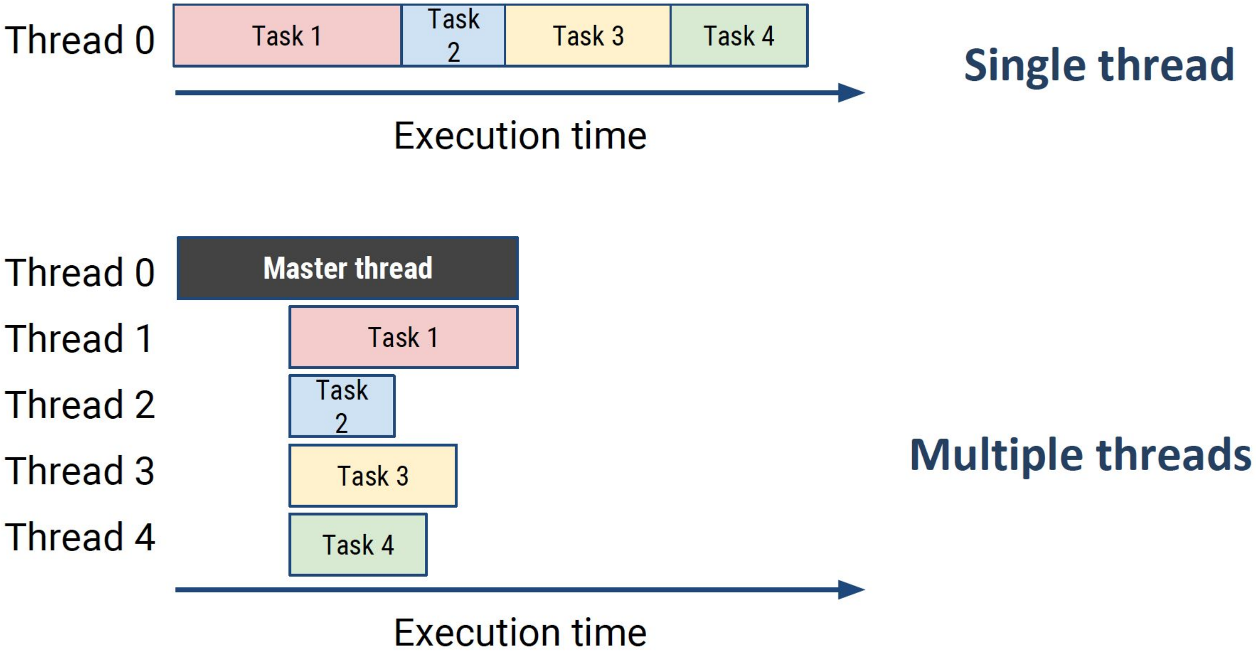
```
while darts < 10000000:
    pi_ests = []
    start = time.time()
    for i in range(n_ests):


        pi_ests.append(estimate_pi(darts))
    t = time.time() - start
    print("Number of runs = {:d}, t = {:2.4f}".format(n_ests, t))
    plt.figure()
    plt.hist(pi_ests)

    pi_est_mean = np.mean(pi_ests)
    pi_est_std  = np.std(pi_ests)
    formstr = "pi_est_mean = {:2.10f}, pi_est_std = {:
    print(formstr.format(pi_est_mean, pi_est_std, dart

    darts = darts*10
```

```
Number of runs = 50, t = 0.0019
pi_est_mean = 3.1536000000, pi_est_std = 0.1640214620, darts = 100
Number of runs = 50, t = 0.0055
pi_est_mean = 3.1442400000, pi_est_std = 0.0506622384, darts = 1000
Number of runs = 50, t = 0.0231
pi_est_mean = 3.1444720000, pi_est_std = 0.0173927231, darts = 10000
Number of runs = 50, t = 0.2262
pi_est_mean = 3.1417144000, pi_est_std = 0.0042169563, darts = 100000
Number of runs = 50, t = 2.8439
pi_est_mean = 3.1414833600, pi_est_std = 0.0016668142, darts = 1000000
```

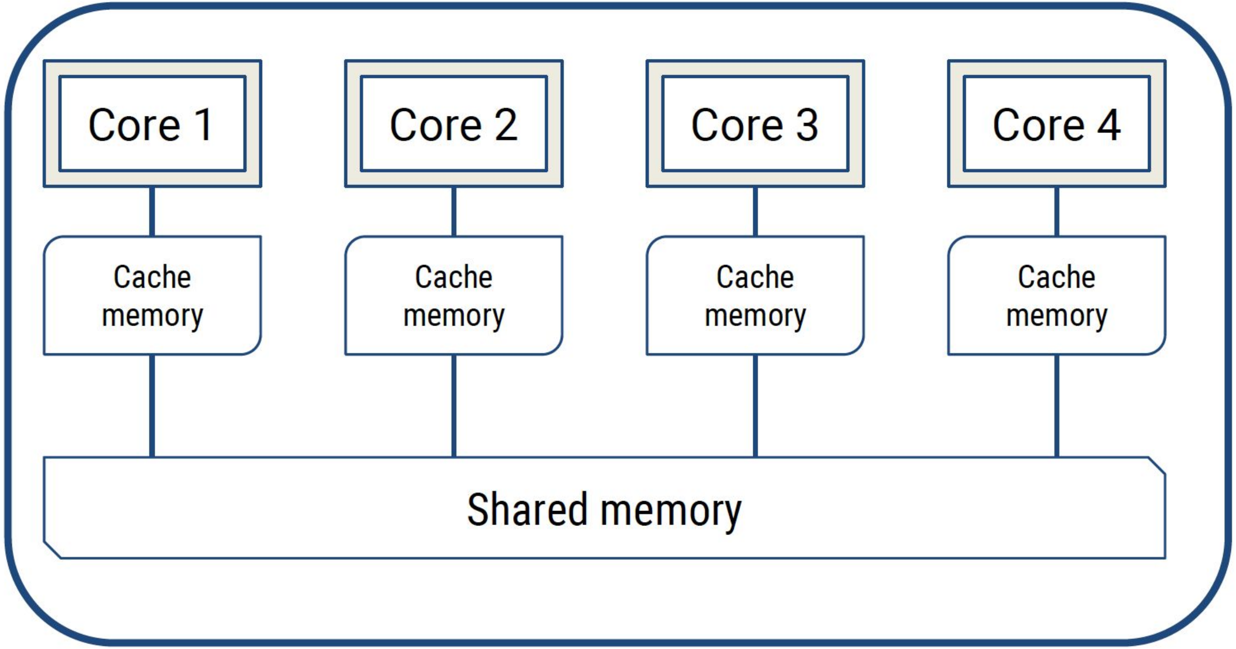
```
Number of runs = 50, t = 0.0301
pi_est_mean = 3.1447520000, pi_est_std = 0.0144816883, darts = 10000
Number of runs = 500, t = 0.2138
pi_est_mean = 3.1417456000, pi_est_std = 0.0169036209, darts = 10000
Number of runs = 5000, t = 2.0944
pi_est_mean = 3.1413668000, pi_est_std = 0.0165346623, darts = 10000
Number of runs = 50000, t = 21.3021
pi_est_mean = 3.1416676240, pi_est_std = 0.0164531550, darts = 10000
```

$\pi = 3.141592653589793$

Why do we want to use parallel processing?



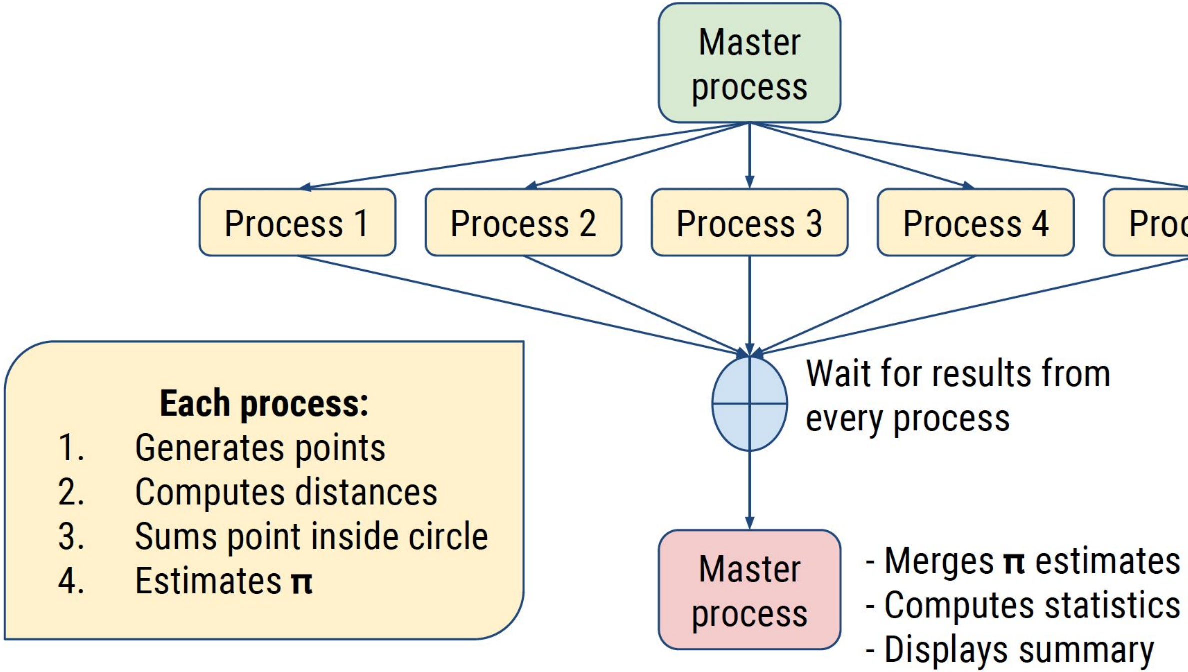
Multi-core processors



Exploiting parallelism in Monte Carlo applications

- Remember, we are trying to model an unknown distribution!
- Works best with a HUGE number of random samples
- Luckily, it's "embarrassingly" parallel
  - Every random sample is independent
- Do the work faster with N parallel tasks
  - Each does 1/N of the total samples
  - Each must use a different seed
  - Merge results at the end

Monte Carlo in Parallel

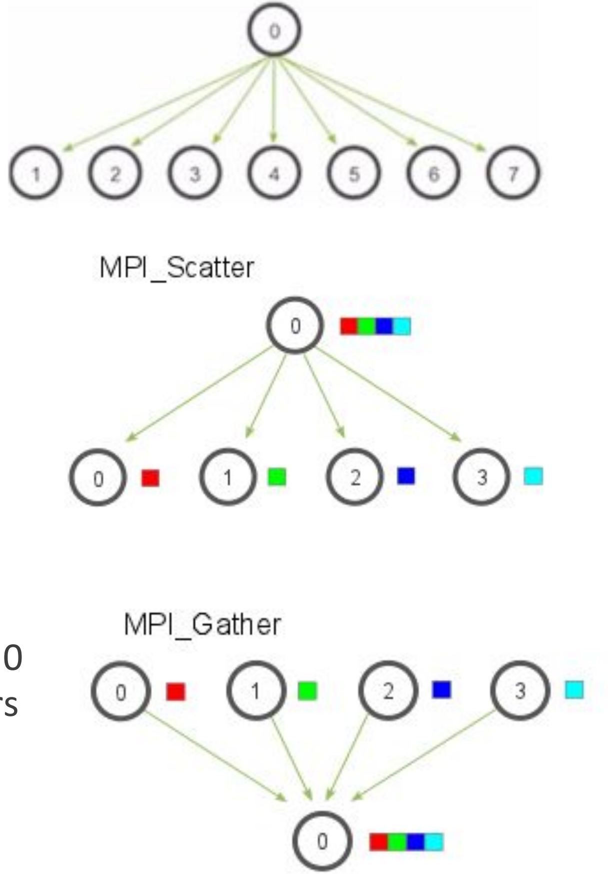


How can we run Python in parallel?

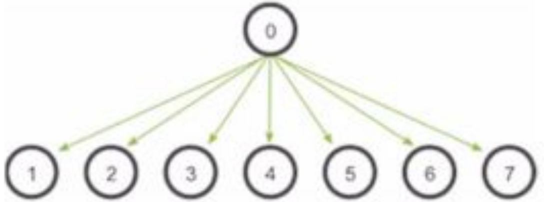
MPI4py - library to implement MPI inside Python

MPI = Message Passing Interface

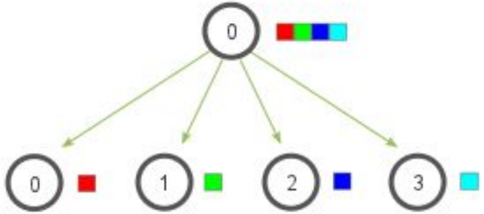
comm = MPI.COMM_WORLD # sets up communication between the processors
size = comm.Get_size() # gives number of ranks in comm
rank = comm.Get_rank() # identifies ranks of the processors, starting with 0
comm.scatter(data) # break up an array and send data out to all processors
comm.gather(data) # gather data from processors



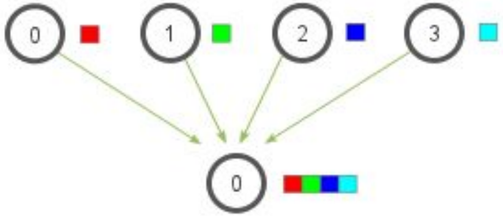
```python
def estimate_pi_in_parallel(comm, N):
    if rank == 0:
        data = [N for i in range(size)]
    else:
        data = None
    data = comm.scatter(data, root=0)
    pi_est = estimate_pi(N)
    pi_estimates = comm.gather(pi_est, root=0)
    if rank == 0:
        return pi_estimates
```



MPI_Scatter



MPI_Gather



High Performance Computing (HPC) Terminology

- Cluster

  A group of computers (nodes) connected by a high-speed network, forming a supercomputer

- Node

  Equivalent to a high-end workstation, part of a cluster

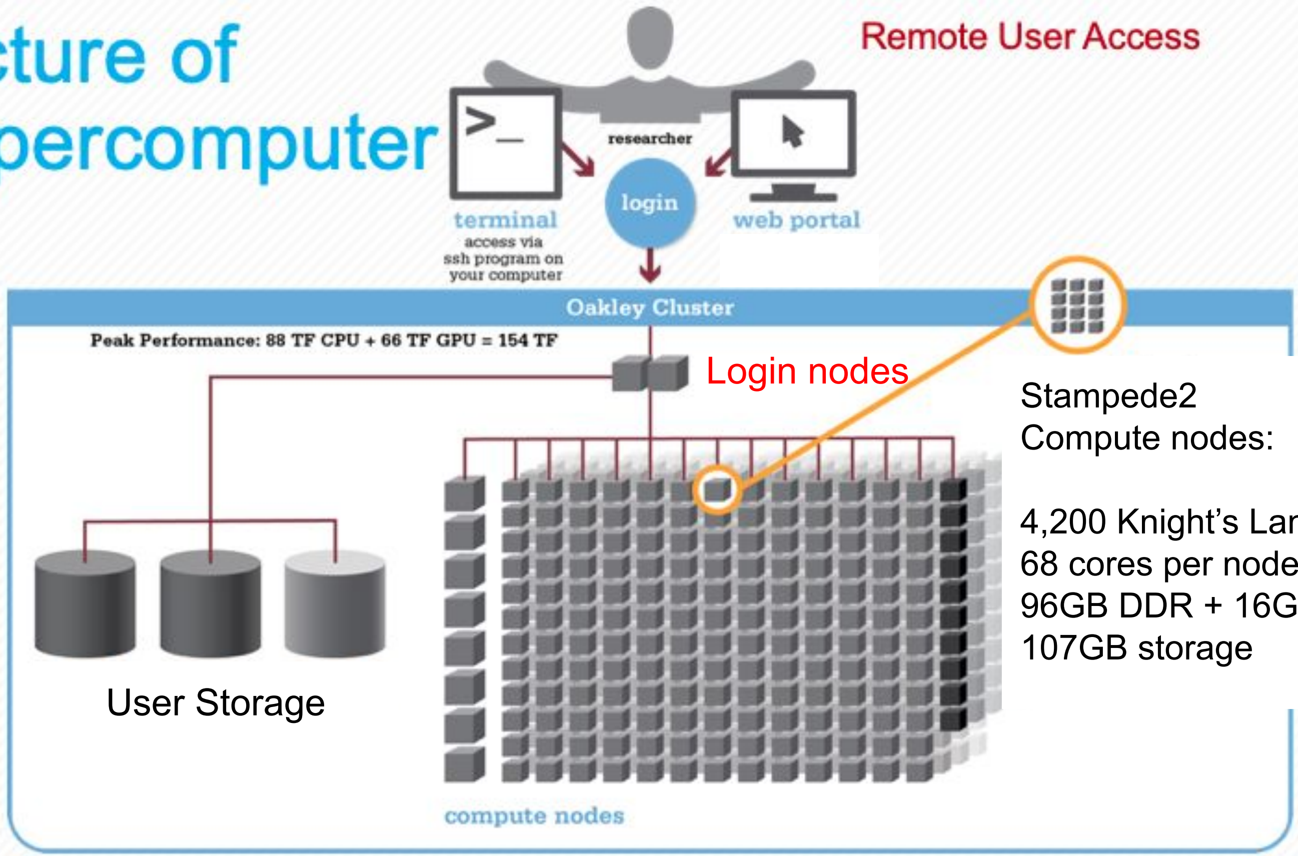
- Core

  A processor (CPU), multiple cores per processor chip

- FLOPS

  "Floating-point Operations (calculations) Per Second"

Structure of a Supercomputer



Stampede2
Compute nodes:

4,200 Knight’s Landing nodes
68 cores per node
96GB DDR + 16GB memory
107GB storage

Today’s Agenda

Log in to Stampede2

Transfer files

Run parallel Monte Carlo code with different core counts

Check timing

Examine output

Worksheet can be used for reference, we will work in terminal