Porting a Python Markov Chain Monte Carlo method to RAPIDS during the SDSC GPU Hackathon

Team members

Prof. Sreedhar Bharath, Dr. Gil Speyer, Chaitanya Inumella, & **Dr. Jason Yalim**

Arizona State University

W.P. Carey School of Business & Computational Research Accelerator

Mentors

Dr. Oded Green, Dr. Huiwen Ju, Dr. Srivathsan Koundinyan

NVIDIA

OpenACC and Hackathons Summit 2022

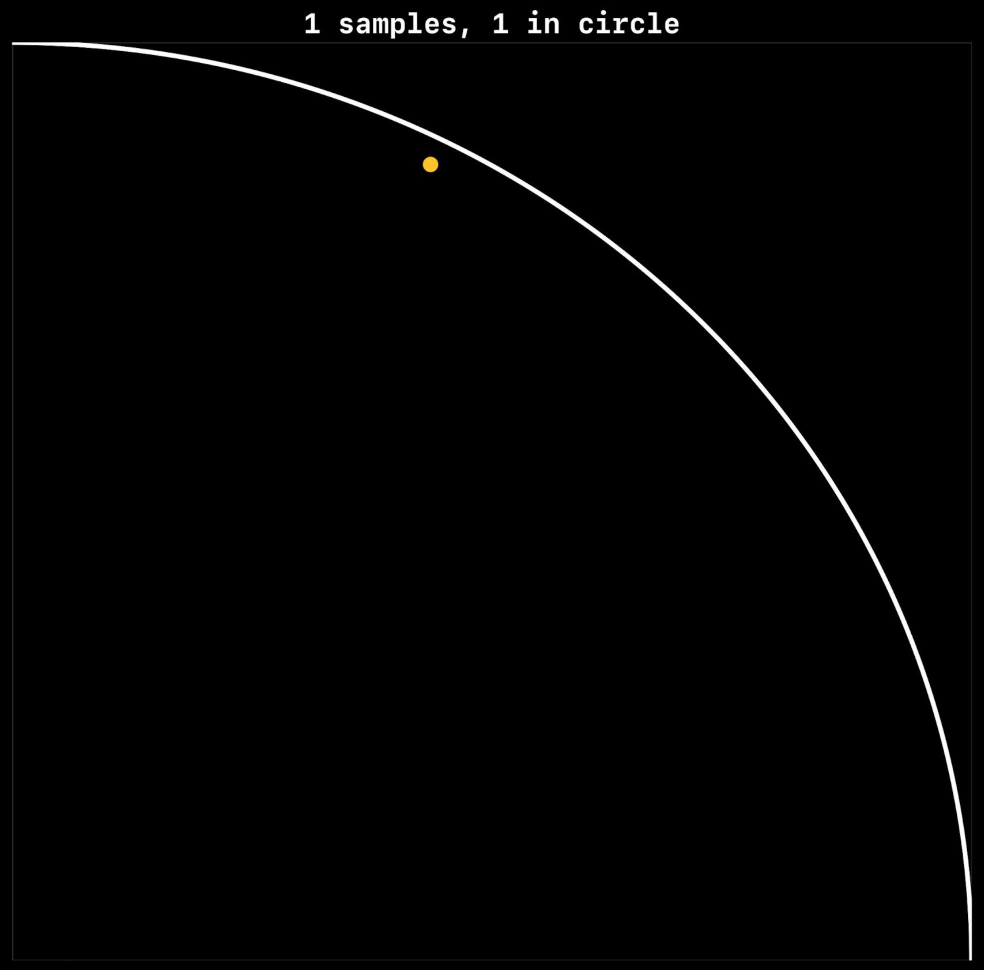
Background

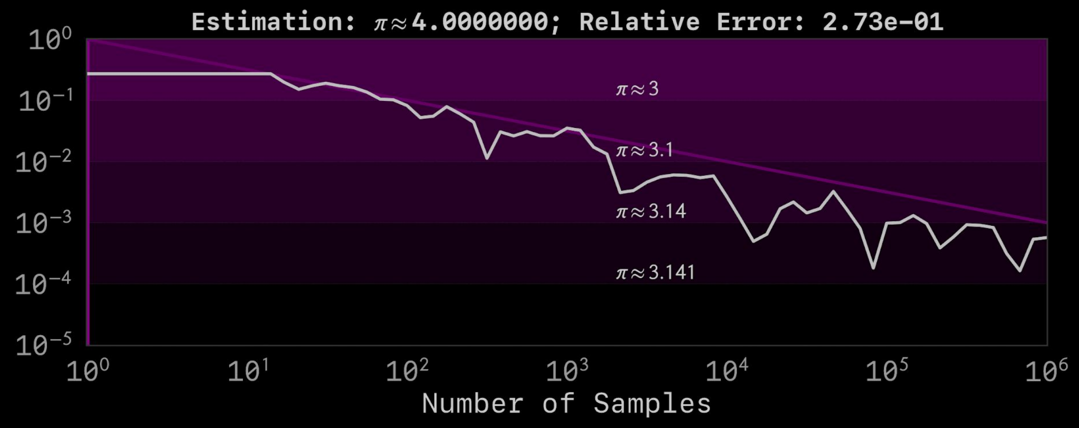
- Asst. Research Prof. at Arizona State University
- Work in Computational Research Accelerator, adjacent to supercomputing Unit
- Approached by Faculty in Economics dept. for help accelerating a Markov Chain Monte Carlo (MCMC) simulation with Python
- Good fortune: accepted into May 2022 SDSC GPU Hackathon
    - Thank you again to our NVIDIA mentors:

      Dr. Oded Green, Dr. Huiwen Ju, Dr. Srivathsan Koundinyan
- Achieved major speed-up overall
- General Outline for the impatient:
    - What is MCMC, what was the application, what were the steps for acceleration, was the Hackathon worth it, how will we continue to progress?

Monte Carlo Simulation

- Example, estimating π = 3.14159…
  - randomly sample a point within a square with a quarter-circle drawn.
  - Count the number of samples inside the quarter-circle
  - Compute ratio to total number of samples, multiply by 4
  - 100x more samples for another digit!
- Other examples
  - Ensembles of Brownian walkers to estimate statistical dynamics
  - Ensembles of weather models that are given randomly perturbed initial conditions

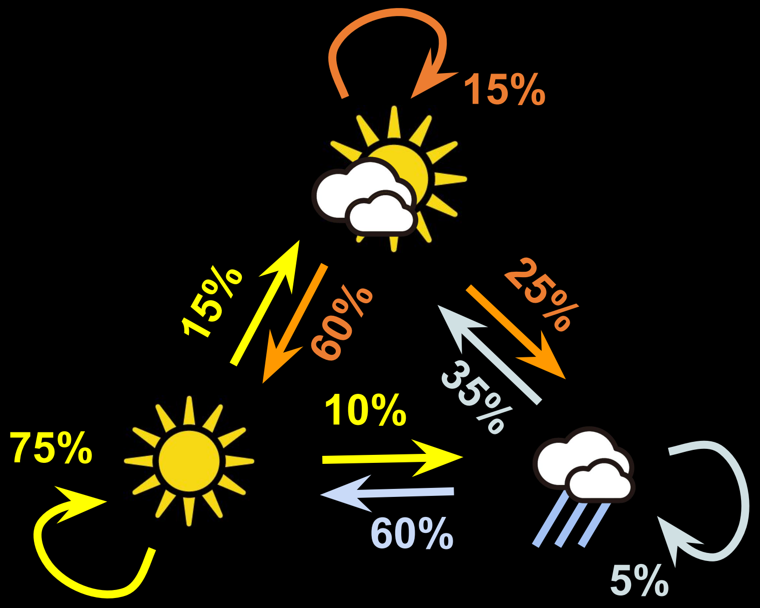




Markov Chains

Observation

Model



	75%	15%	10%
	60%	15%	25%
	60%	35%	5%

Equilibrium Distribution

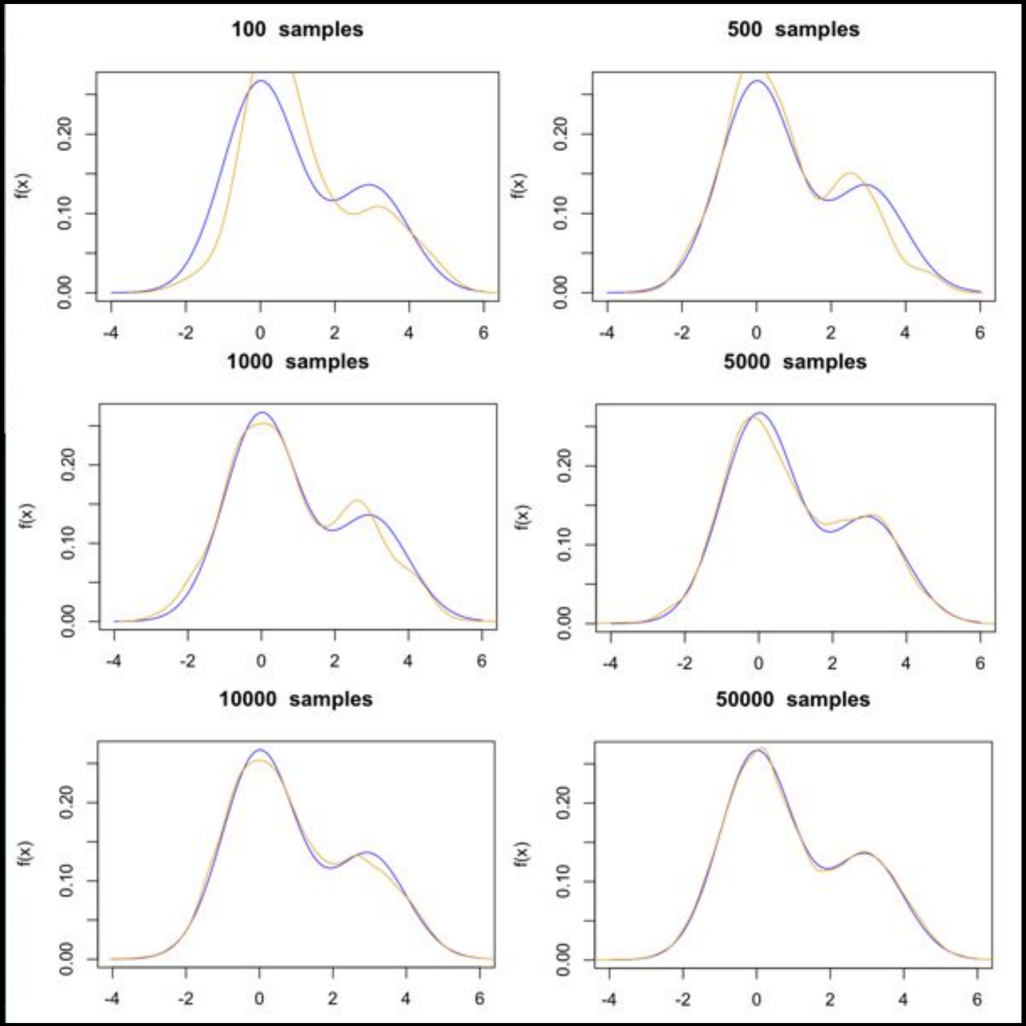
	70.6%
	17.4%
	12%

Markov Chain Monte Carlo (MCMC)

- Created to estimate high-dimensional problems
- In a nutshell, a Monte Carlo simulation that samples and trends towards a Markov Chain equilibrium
- Ensemble random walkers become **autocorrelated**
- Convergence given by (order $N^{-0.5}$)

$$\sigma^2 = (\tau / N)\ \mathrm{Var}_{p(\theta)}[f(\theta)]$$

TAKES 100x more samples for 1 digit of convergence!



The Application: MCMC Staggered Board

- Economics/finance, models the “staggered board”, designed to prevent rapid change and encourage higher risk projects, e.g., Jet Blue immediately changing Spirit Airlines’ management
- Model quantifying structure utility involves evaluating high-dimensional integral with Markov Chain Monte Carlo (MCMC) method
- 110 Hours = 4 days 14 hours => 1,000,000 iterations : ~15 months!

```
ORIGINAL
100%|██████████| 10000/10000 [113:34:14<00:00, 40.89s/it]
100%|██████████| 10000/10000 [106:43:54<00:00, 38.42s/it]
100%|██████████| 10000/10000 [111:34:52<00:00, 40.17s/it]
100%|██████████| 10000/10000 [107:37:34<00:00, 38.75s/it]
```

Variable Caching

```python
def T_omega(omega,d,e):
    Q=np.shape(omega)[0]
    omega=omega.reshape(Q,1)
    mu=omega_seed-(alpha0+alpha1*omega+alpha2*d+alpha4*e+alpha5*d*omega+\
                   alpha6*e*omega+alpha7*d*e*omega)
    T=normal_pdf(mu,sig_o)/np.sum(normal_pdf(mu,sig_o))
    return T

def T_X(X_seed,x,eta,const,sigx):
    Q=np.shape(x)[0]
    mu=X_seed-eta*x.reshape(Q,1)-const
    T=normal_pdf(mu,sigx)/np.sum(normal_pdf(mu,sigx))
    return T
```

```python
def T_omega(omega,d,e):
    Q=np.shape(omega)[0]
    omega=omega.reshape(Q,1)
    mu=omega_seed-(alpha0+alpha1*omega+alpha2*d+alpha4*e+alpha5*d*omega+\
                   alpha6*e*omega+alpha7*d*e*omega)
    npdf = normal_pdf(mu,sig_o)
    T=npdf/npdf.sum()
    return T

def T_X(X_seed,x,eta,const,sigx):
    Q=np.shape(x)[0]
    mu=X_seed-eta*x.reshape(Q,1)-const
    npdf = normal_pdf(mu,sigx)
    T=npdf/npdf.sum()
    return T
```

~2x speed up

```
ORIGINAL
100%|██████████| 10000/10000 [113:34:14<00:00, 40.89s/it]
100%|██████████| 10000/10000 [106:43:54<00:00, 38.42s/it]
100%|██████████| 10000/10000 [111:34:52<00:00, 40.17s/it]
100%|██████████| 10000/10000 [107:37:34<00:00, 38.75s/it]
```

Einstein Summation

```
def value_function(omega,asset,cash,v, beta_da, beta_dc, beta_sa,
                   beta_sc, u_d, u_s):
## function 10-13, page 7
    T11=T_omega(omega,1,1)
    T10=T_omega(omega,1,0)
    T01=T_omega(omega,0,1)
    T00=T_omega(omega,0,0)
    T_asset=T_X(asset_seed,asset,eta_a,const_a,sig_a)
    T_cash=T_X(cash_seed,cash,eta_c,const_c, sig_c)

    V11=np.sum(T11*T_asset*T_cash*v, axis=1)
    V01=np.sum(T01*T_asset*T_cash*v, axis=1)
    V10=np.sum(T10*T_asset*T_cash*v, axis=1)
    V00=np.sum(T00*T_asset*T_cash*v, axis=1)
```

```
def build_T_tensor(omega):
    T  = np.tensordot(E00,T_omega(omega,0,0),axes=0)
    T += np.tensordot(E01,T_omega(omega,0,1),axes=0)
    T += np.tensordot(E10,T_omega(omega,1,0),axes=0)
    T += np.tensordot(E11,T_omega(omega,1,1),axes=0)
    return T
```

```
def value_function(omega,asset,cash,v, beta_da, beta_dc, beta_sa,
                   beta_sc, u_d, u_s):
## function 10-13, page 7
    # T shape (2,2,M,M), v is shape (M,)
    T  = build_T_tensor(omega)
    T_asset=T_X(asset_seed,asset,eta_a,const_a,sig_a)
    T_cash=T_X(cash_seed,cash,eta_c,const_c, sig_c)
    V  = np.einsum('ijkl,kl->ijk',T,T_asset*T_cash*v)
```

```
ORIGINAL
100%|          | 10000/10000 [113:34:14<00:00, 40.89s/it]
100%|          | 10000/10000 [106:43:54<00:00, 38.42s/it]
100%|          | 10000/10000 [111:34:52<00:00, 40.17s/it]
100%|          | 10000/10000 [107:37:34<00:00, 38.75s/it]
PERFORMANCE
100%|          | 10000/10000 [6:44:44<00:00,  2.43s/it]
100%|          | 10000/10000 [6:36:03<00:00,  2.38s/it]
100%|          | 10000/10000 [5:24:42<00:00,  1.95s/it]
100%|          | 10000/10000 [6:35:03<00:00,  2.37s/it]
```

~20x speed up

Identified Computational Costs Pre GPU

- Kernel function (likelihood function) evaluation major computational cost, involves 900x40,000 matrix spanning 6D-space.
- Greatest computational costs evaluating Gaussian & Einsum
- GPU can be utilized with NVIDIA RAPIDS

```
def normal_pdf(mu,sigx):
  # N(\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp{-0.5\mu^2/\sigma^2}
  # 1/sqrt(2*pi) = 0.3989422804014327794
  inv_sig = 1/sigx
  return np.exp(-0.5*(mu*inv_sig)**2)*inv_sig*0.3989422804014327794

  V         = np.einsum('ijkl,kl->ijk',T,T_asset*T_cash*v)
```

Starting Profile

```
         1390038 function calls (1366989 primitive calls) in 444.548 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1518  138.072    0.091  138.072    0.091 {built-in method numpy.core._multiarray_umath.c_einsum}
     6078  133.203    0.022  133.203    0.022 /home/sbharath/StaggeredBoard/sample-runs/likelifun1.py:107(normal_pdf)
      506   81.711    0.161  428.634    0.847 /home/sbharath/StaggeredBoard/sample-runs/likelifun1.py:190(likeli_fun)
22688/18041   32.452    0.001  170.980    0.009 {built-in method numpy.core._multiarray_umath.implement_array_function}
     4052   17.887    0.004  110.666    0.027 /home/sbharath/StaggeredBoard/sample-runs/likelifun1.py:116(T_omega)
     1013   16.351    0.016  159.721    0.158 /home/sbharath/StaggeredBoard/sample-runs/likelifun1.py:132(build_T_tensor)
     2026    8.307    0.004   51.033    0.025 /home/sbharath/StaggeredBoard/sample-runs/likelifun1.py:125(T_X)
      506    4.396    0.009  214.907    0.425 /home/sbharath/StaggeredBoard/sample-runs/likelifun1.py:173(v_fun)
     1012    3.177    0.003  210.472    0.208 /home/sbharath/StaggeredBoard/sample-runs/likelifun1.py:153(value_function)
```

Day 2 Profile

carefully: `import cupy as cp`

Hardware: A100 with 80GB of RAM

```
         11436749 function calls (11405607 primitive calls) in 289.174 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  1036489  142.153    0.000  147.037    0.000 /home/uvxj3hd/.local/opt/conda/envs/rapids-22.04/lib/python3.9/site-packages/rmm/rmm.py:198(rmm_cupy_allocator)
    21798   45.640    0.002  121.219    0.006 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:138(normal_pdf)
     3633   25.862    0.007  161.191    0.044 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:168(build_T_tensor)
     3632   14.894    0.004  246.737    0.068 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:184(value_function)
     5448    9.286    0.002   14.376    0.003 {built-in method cupy._core._routines_linalg.matmul}
    14532    8.507    0.001  121.621    0.008 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:147(T_omega)
     1816    6.271    0.003  253.678    0.140 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:204(v_fun)
     7266    4.812    0.001   54.922    0.008 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:156(T_X)
     1816    4.276    0.002  276.191    0.152 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:221(likeli_fun)
```

Final Profile

Hardware: A100 with 80GB of RAM

```
        11436749 function calls (11405607 primitive calls) in 289.174 seconds

  Ordered by: internal time

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 1036489  142.153    0.000  147.037    0.000 /home/uvxj3hd/.local/opt/conda/envs/rapids-22.04/lib/python3.9/site-packages/rmm/rmm.py:198(rmm_cupy_allocator)
   21798   45.640    0.002  121.219    0.006 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:138(normal_pdf)
    3633   25.862    0.007  161.191    0.044 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:163(build_T_tensor)
    3632   14.894    0.004  246.737    0.068 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:184(value_function)
    5448    9.286    0.002   14.376    0.003 {built-in method cupy._core._routines_linalg.matmul}
   14532    8.507    0.001  121.621    0.008 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:147(T_omega)
    1816    6.271    0.003  253.678    0.140 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:204(v_fun)
    7266    4.812    0.001   54.922    0.008 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:156(T_X)
    1816    4.276    0.002  276.191    0.152 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:221(likeli_fun)
```

```python
import rmm
import cupy as cp
cp.cuda.set_allocator(rmm.rmm_cupy_allocator)
#rmm.reinitialize(pool_allocator=True)
mempool = rmm.mr.PoolMemoryResource(
  rmm.mr.CudaMemoryResource(),
  initial_pool_size = 2**33,
  maximum_pool_size = 2**35,
)
rmm.mr.set_current_device_resource(mempool)
```

```
        34761666 function calls (34723630 primitive calls) in 91.516 seconds

  Ordered by: internal time

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 3436730   15.938    0.000   24.827    0.000 /home/uvxj3hd/.local/opt/conda/envs/rapids-22.04/lib/python3.9/site-packages/rmm/rmm.py:198(rmm_cupy_allocator)
   50748   12.221    0.000   28.972    0.001 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:157(T_omega)
   76122    6.995    0.000   11.647    0.000 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:148(normal_pdf)
   24024    6.953    0.000   84.218    0.004 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:261(likeli_fun0)
   12686    6.883    0.001   61.595    0.005 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:195(value_function)
    6343    4.864    0.001   76.765    0.012 /home/uvxj3hd/.local/src/mcmc-staggered-board/run/gpu-v0.1/likelifun1.py:236(likeli_fun)
```

Results

```
ORIGINAL
100%|          | 10000/10000 [113:34:14<00:00, 40.89s/it]
100%|          | 10000/10000 [106:43:54<00:00, 38.42s/it]
100%|          | 10000/10000 [111:34:52<00:00, 40.17s/it]
100%|          | 10000/10000 [107:37:34<00:00, 38.75s/it]
PERFORMANCE
100%|          | 10000/10000 [6:44:44<00:00,  2.43s/it]
100%|          | 10000/10000 [6:36:03<00:00,  2.38s/it]
100%|          | 10000/10000 [5:24:42<00:00,  1.95s/it]
100%|          | 10000/10000 [6:35:03<00:00,  2.37s/it]

DAY 2
100%|          | 200/200 [04:33<00:00,  1.37s/it]

DAY 3
100%|          | 10000/10000 [12:25<00:00, 13.42it/s]
```

~540x/2150x

~30x

~20x

Results

```
ORIGINAL
100%|██████████| 10000/10000 [113:34:14<00:00, 40.89s/it]
100%|██████████| 10000/10000 [106:43:54<00:00, 38.42s/it]
100%|██████████| 10000/10000 [111:34:52<00:00, 40.17s/it]
100%|██████████| 10000/10000 [107:37:34<00:00, 38.75s/it]
PERFORMANCE
100%|██████████| 10000/10000 [6:44:44<00:00,  2.43s/it]
100%|██████████| 10000/10000 [6:36:03<00:00,  2.38s/it]
100%|██████████| 10000/10000 [5:24:42<00:00,  1.95s/it]
100%|██████████| 10000/10000 [6:35:03<00:00,  2.37s/it]

DAY 2
100%|██████████| 200/200 [04:33<00:00,  1.37s/it]

DAY 3
100%|██████████| 10000/10000 [12:25<00:00, 13.42it/s]
```

~540x/2150x

~30x

~20x

1,000,000 iterations takes ~15 months! → 1,000,000 iterations takes ~20 hours!

Was it worth it?

- **Yes**.
  - Access to incredible resources and expertise
  - 10,000 iterations in 110 hours -> 1,000,000 iterations in 20 hours
    - Monte Carlo methods converge $O(\sqrt{N})$

    -> additional digit of convergence!
  - Broader impact for Economics community and all those that use MCMC (i.e. STEM in general)
    - Use of high-level language, Python, to achieve state-of-the-art convergence in model!
- Continued development:
  - Default floating-point type is 64-bit. Decrease to 32-bit is realistic.
  - Numba for expensive routines

Wishlist

- Interoperability with multiprocessing/MPIpool for Process parallelization (emcee/cuda conflicts)
- Blackbox support for the Gaussian function
  - Potentially cupy may support scipy.special.eval_hermite [doc]
- Improved documentation on nsys command and Python API

Problems Encountered

- multiprocess level parallelization not possible without careful refactor of python dependency “emcee”
- `__pycache__/` from previous benchmarking attempts was throttling performance (easy fix, `rm -rf __pycache__`)
- Curiosity
  - Greater stability (jobs persisted but processes would hang up)
  - Provisioning throughout Hackathon, not just during Hackathon hours
  - Profiling required alternative nodes (dgx01 & dgx09 had issues, FS related)

```
FATAL ERROR: Throw location unknown (consider using BOOST_THROW_EXCEPTION)
Dynamic exception type: boost::exception_detail::clone_impl<boost::exception_d
etail::current_exception_std_exception_wrapper<std::runtime_error> >
std::exception::what: boost::filesystem::file_size
boost::filesystem::filesystem_error
```

Conclusion

Application Background

- Markov Chain Monte Carlo (MCMC) estimator for six-dimensional integral evaluation.
- Models “staggered board”, a managerial practice to prevent rapid change to leadership (staggered contract expiration)
- Measures benefits to R&D with such a structure
- Application has broader impact as MCMC is widely utilized for Bayesian inference.

Hackathon Objectives & Approach

- Python code, heavily written with `numpy`.
- cProfile revealed majority of time evaluating tensor product and normalized Gaussian.
- Incoming strategy was to use NVIDIA’s Rapids, i.e. `cupy`, ideally as drop-in replacement for CPU-based `numpy`.

Results

```
ORIGINAL
100%|██████████| 10000/10000 [113:34:14<00:00, 40.89s/it]
100%|██████████| 10000/10000 [106:43:54<00:00, 38.42s/it]
100%|██████████| 10000/10000 [111:34:52<00:00, 40.17s/it]
100%|██████████| 10000/10000 [107:37:34<00:00, 38.75s/it]
```

Original code took ~110 hours for 10,000 iterations (~40 seconds per iteration), occupying full workstation and ran 4 parametrizations (total of ~440 hours, **nearly 3 weeks**)

```
100%|██████████| 10000/10000 [12:25<00:00, 13.42it/s]
```

New code is ~540 times faster per iteration, and 40GB A100 allows all 4 cases to run simultaneously (**~2150x speedup**).

Technical Achievements & Impact

- Relative to original code, iterations are about 540 times faster, and ability to run multiple processes on single GPU (with 4 cases to run) provides about a 2150 factor speed-up.
- By carefully understanding the linear algebra, and the software, optimal routines were used and then accelerated with the GPU
- State-of-the-art convergence with MCMC enabled.

Thanks!

Next session begins at 9:10 AM PT