Python Development Schemes for Monte Carlo Neutronics on High Performance Computing

Jackson P. Morgan & Kyle E. Niemeyer

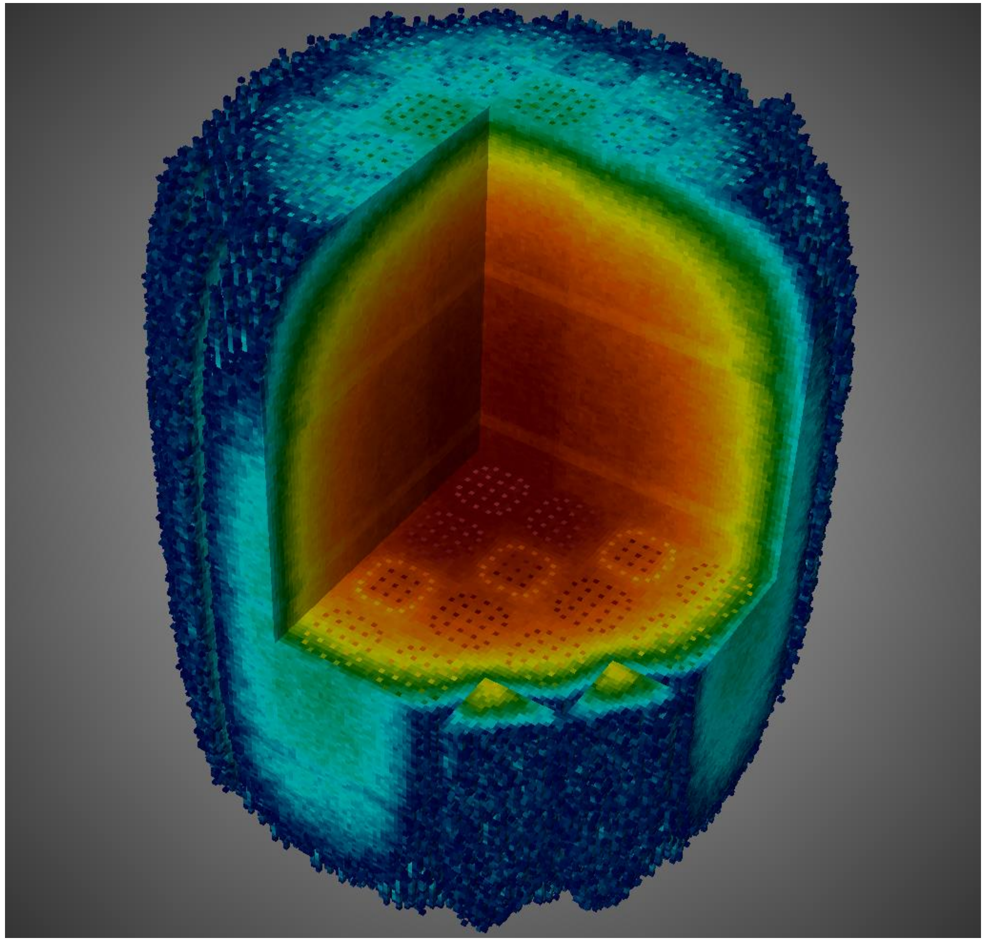
The Center for Exascale Monte Carlo Neutron Transport (CEMeNT)
Oregon State University

SciPy 2022
Austin, TX
July 17th, 2022

This work was supported by the Center for Exascale Monte-Carlo Neutron Transport (CEMeNT) a PSAAP-III project funded by the Department of Energy, grant number: DE-NA003967

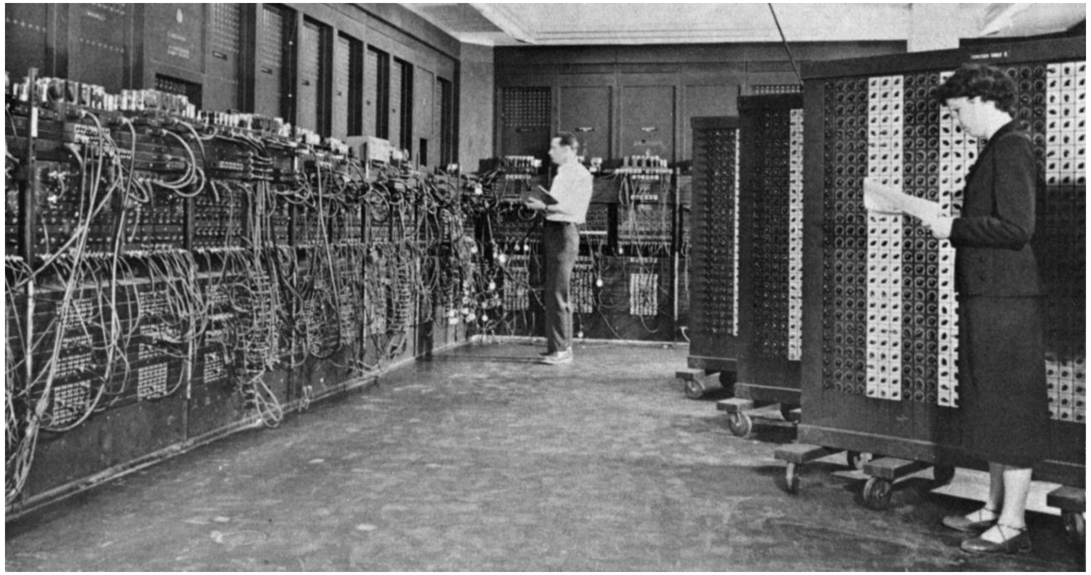
Neutron Transport

- Trying to answer where, how, and when neutrons interact with a domain
- Applications:
  - Cancer radio therapy development
  - Power reactor analysis
  - Other governmental implementations



Neutron Transport and HPC

There at the beginning…



ENIAC – 1946

First general programable computer circa 1946

https://ieeexplore.ieee.org/document/6880250

There now

El Capitan – 2023

Heterogeneous exa-scale machine

Direct Monte Carlo Simulations

Monte Carlo Methods

Can we use statistics and event rates to model a system?

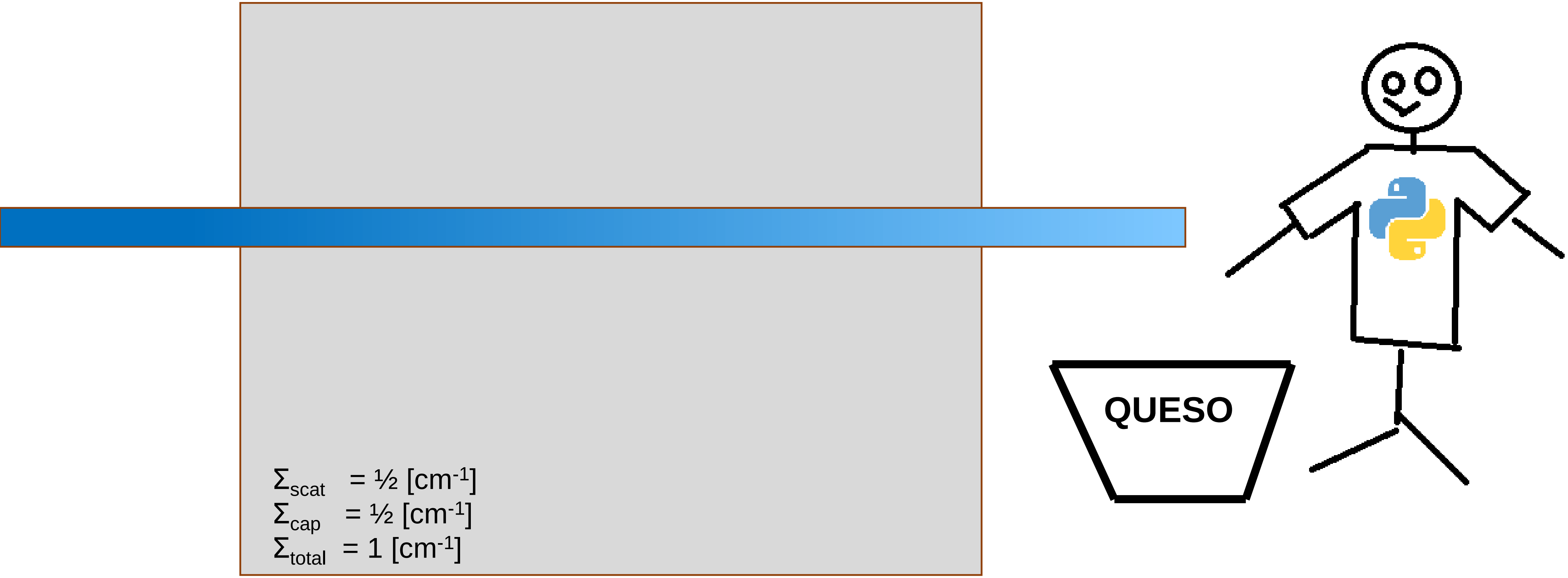
Traditional numerical methods (deterministic methods) get an exact solution to an estimated problem

Monte Carlo gives an estimated solution to an exact problem

Monte Carlo Neutronics

A wild BEAM OF NEUTRONS appeared!

Are we safe?



$\Sigma_{scat} = ½ \text{ [cm}^{-1}]$
$\Sigma_{cap} = ½ \text{ [cm}^{-1}]$
$\Sigma_{total} = 1 \text{ [cm}^{-1}]$

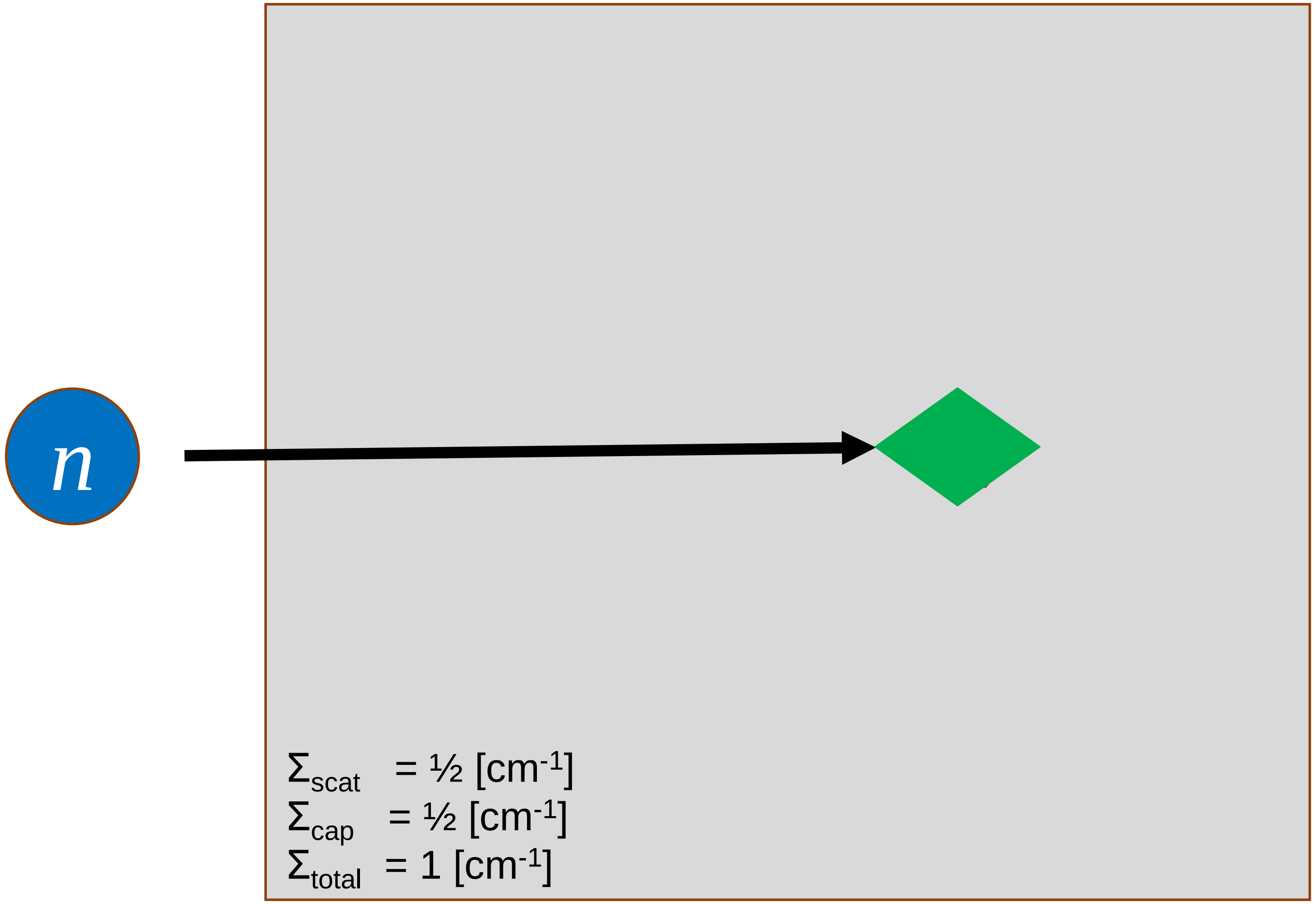
Monte Carlo Neutronics

- Material Data (statistical likelihoods)
- Equations where can relate a distance to a probability
- List of events that could possible happen (scatter, absorption, transmission)

$$S = -\frac{\ln(\xi)}{\Sigma_t}$$

ξ: random number [0,1]
Σ_t: total cross section

Monte Carlo Neutronics



Events:

1. Roll a random number
   `ξ = 0.345`
2. Compute distance to event
   `S = 1.06 cm`
3. Move the particle
4. Roll a new random number
   `ξ = 0.544`
5. Determine new event type
   `ξ > 0.5`
6. Tally
   `Absorption`

Dead Slow

To get a decent solution we will need to do this over

and over…
and over…
and over…
and over…
and over…
and over…

⋮ ⋮

$$\mathcal{O} = \frac{1}{\sqrt{n}}$$

The Problem Fully Defined

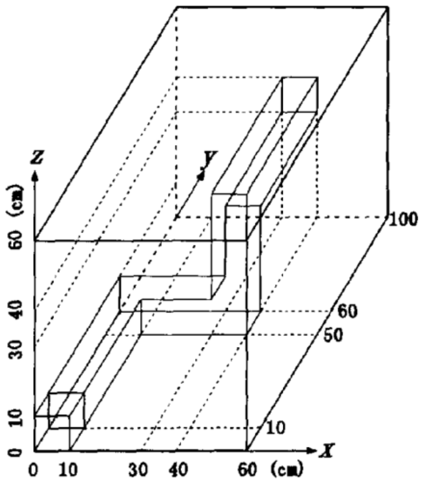
How do we

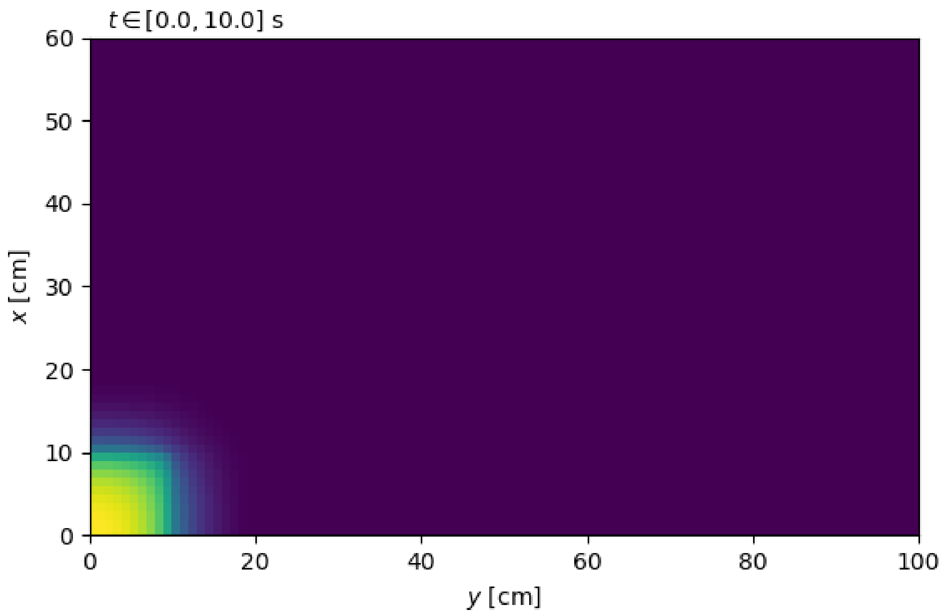
1. *Write high performance compute kernels*
2. *That can't use off the shelf libraries*
3. *For heterogeneous machines*
4. *That isn't too syntactically dense so all can participate*
5. *And maybe even total abstraction of hardware target or at least abstract vector machines from CPUs*

MC/DC

MC/DC: Monte Carlo / Dynamic Code

- Dynamic neutron transport solver made for rapid methods exploration at high performance computing and exa-scale
- Target various hardware architecture
- Currently parallelized with mpi4py





Kobyashi Problem: Image courtesy Ilham Variansyah

GitHub: https://github.com/CEMeNT-PSAAP/MCDC

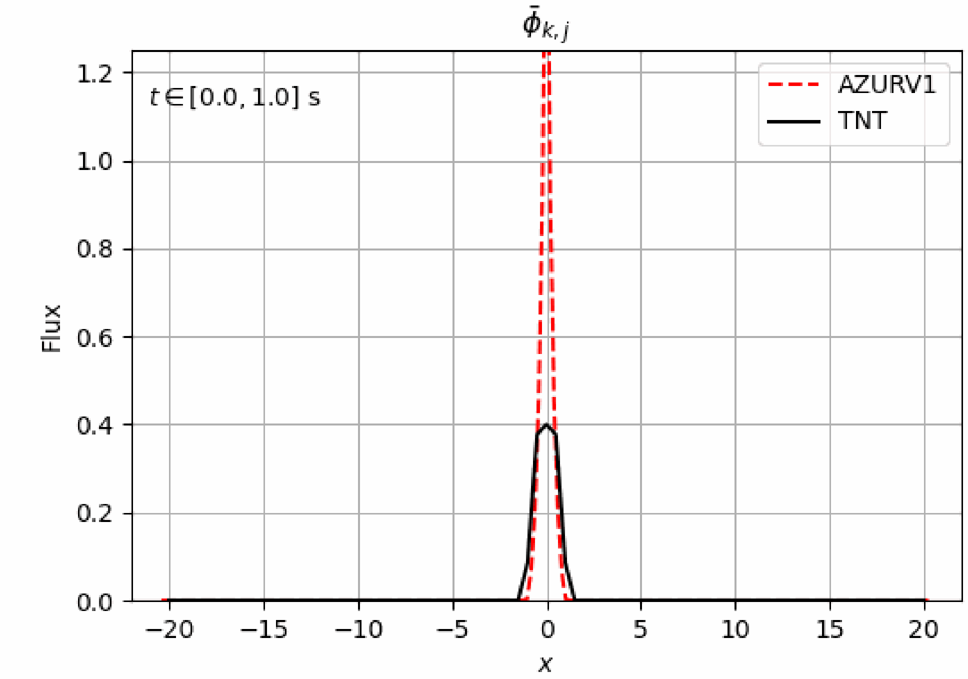
MC/DC–TNT: Toy Neutronics Testbed

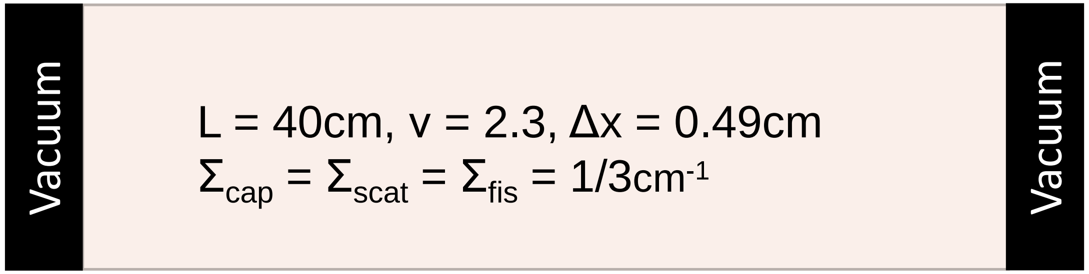
CEMENT

- Mono-energetic, slab-geometry, transient tallies, fission, event-based, with surface tracking

- Architecture targets: Nvidia GPUs and x86 CPUs

- Validated with analytic solutions (AZURV1 [1])



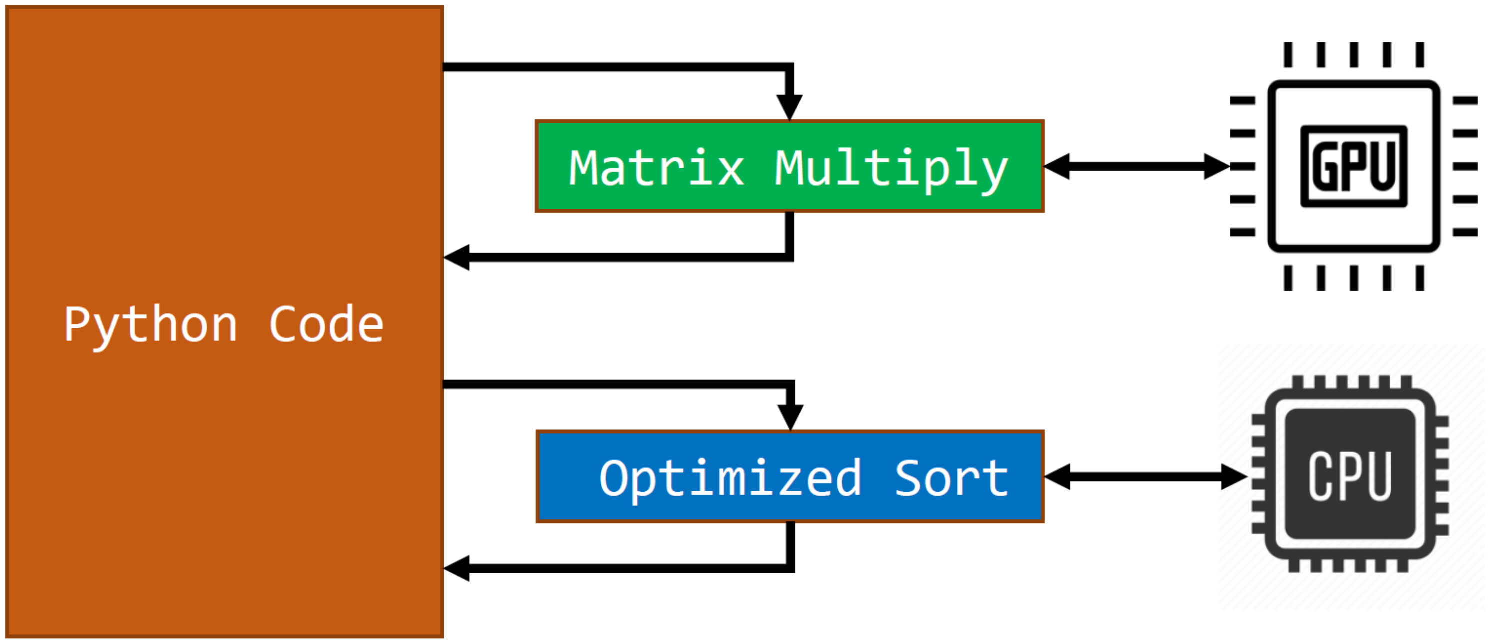


GitHub: https://github.com/CEMeNT-PSAAP/MCDC-TNT

Methods of Acceleration

Heterogeneous Targeting: Python Glue

- Python serves as glue code
- Native Python modules used produce and just-in-time (JIT) schemes
- Can target multiple architecture types



A potential accelerated Python program

PyKokkos

```python
import math
import numpy as np
import pykokkos as pk

@pk.workload
class vecLog:
    def __init__(self, vec, total, N):
        self.vec: pk.View1D[pk.float] = vec
        self.total: pk.View1D[pk.float] = total
        self.N: int = N

    @pk.main
    def run(self):
        pk.parallel_for(self.N, self.vecLog_wu)

    @pk.workunit
    def vecLog_wu(self, i: int):
        self.vec[i] = math.log(self.vec[i])
        pk.atomic_fetch_add(self.total, [0], self.vec[i])


if __name__ == '__main__':
    space = pk.ExecutionSpace.Cuda #pk.ExecutionSpace.OpenMP
    pk.set_default_space(space)

    data_type = np.float32
    N: int = 32

    vec = np.random.random(N).astype(data_type)
    vec_pyk = pk.from_numpy(vec)
    total = np.zeros(1, data_type)
    total_pyk = pk.from_numpy(total)

    pk.execute(space, vecLog(vec, total, N))
```

- Python library that implements parts of Kokkos Portability framework [2]
- Brand new and under active development
- Treats functions as objects to run in pyk commands

Numba + CUDA

- Converts Python code then implements the LLVM compiler [3]
- Industry support and active development
- Often operates on pure Python code
- Experimental full implementation of OpenMP [4]

```python
import numpy as np
import numba as nb
import math
from numba import cuda


@cuda.jit
def vecLog(vec, total):
    i: int = cuda.grid(1)

    vec[i] = math.log(vec[i])
    cuda.atomic.add(total, [0], vec[i])

if __name__ == '__main__':
    data_type = np.float32
    N: int = 32

    vec = np.random.random(N).astype(data_type)
    total = np.zeros(1, data_type)

    vec_cuda = cuda.to_device(vec)
    total_cuda = cuda.to_device(total)

    threadsperblock = 32
    blockspergrid = (N + (threadsperblock - 1)) // threadsperblock

    vecLog[blockspergrid, threadsperblock](vec, total, N)
```

HCGLs (PyCUDA)

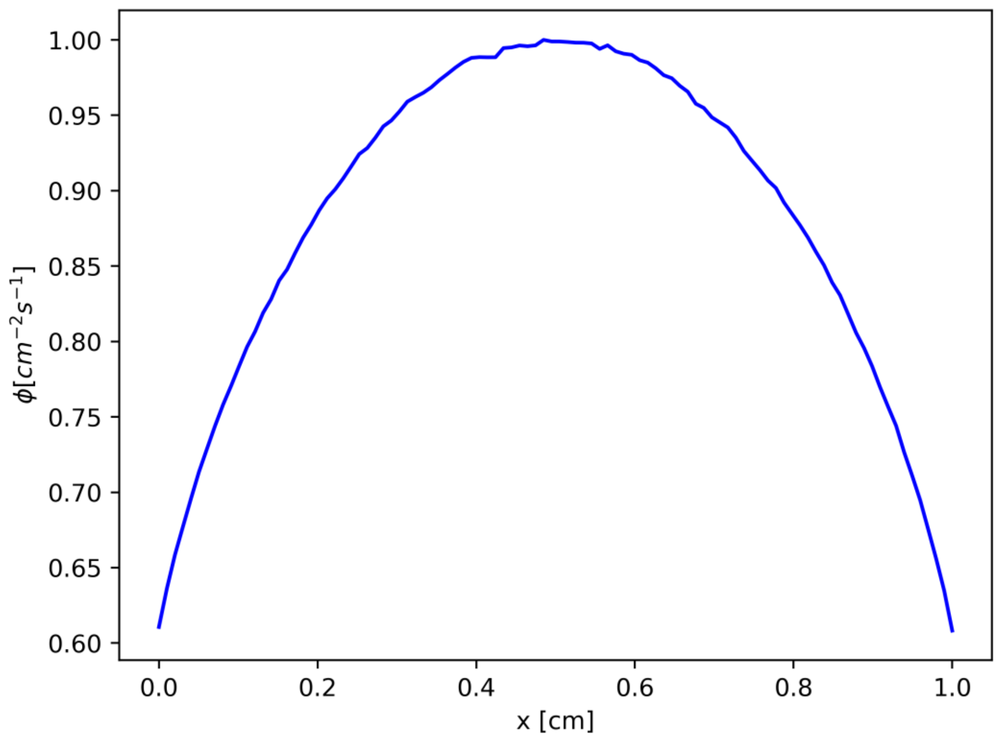
- Implemented on PyFR [5] at petascale [6]
- Code-generating libraries to compile code
- Have to write our own source

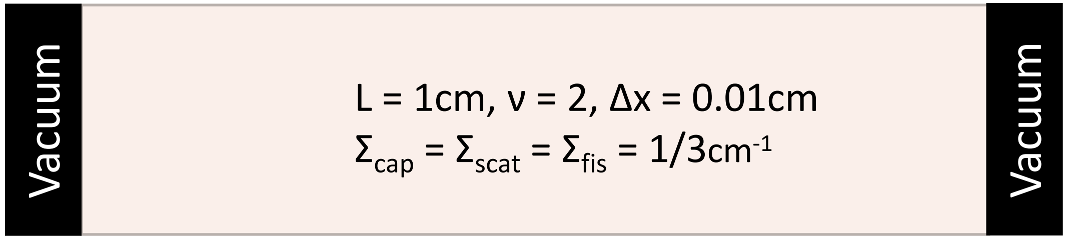
```python
import numpy as np

import pycuda.autoinit
import pycuda.driver as drv
from pycuda.compiler import SourceModule

mod = SourceModule("""
__global__ void vecLog(float *vec, float *total){
    const int i = threadIdx.x + blockIdx.x * blockDim.x;

    float vec[i] = log(vec[i]);
    atomicAdd(&total[0], vec[i]);

}""")



if __name__ == '__main__':
    data_type = np.float32
    N: int = 32

    vec = np.random.random(N).astype(data_type)
    total = np.zeros(1, data_type)

    threadsperblock = 32
    blockspergrid = (N + (threadsperblock - 1)) // threadsperblock

    vecLog = mod.get_function("vecLog")

    vecLog(drv.InOut(vec), drv.InOut(total),
           block=(threadsperblock, 1, 1), grid=(blockspergrid, 1))
```

Results

Runtime Test Problem

- Sub-critical slab with initial population of $1{\times}10^8$ particles
- Validated with MC/DC
- Follow particles till death





Performance: CPU

Integration test problem: $L = 1\text{cm}$, $\Delta x = 0.01\text{cm}$, $\Sigma_f = \Sigma_c = \Sigma_s = 1/3\ \text{cm}^{-1}$, $\nu = 2$, vacuum boundary conditions on LHS and RHS w/ 1×10^8 Initial particles

Method of Implementation	Compile Time [s]	Run Time [s]
Pure Python*	N/A	52970
Numba (Native threading)	5.28	232.3
Numba PyOmp	5.66	382.3
PyKokkos	37.50	158.4

16 threads on an i7-10875H CPU
*one thread

Performance: GPU Implementation

CEMENT

Integration test problem: L = 1cm, Δx = 0.01cm, $\Sigma_f = \Sigma_c = \Sigma_s$ = 1/3 cm^{-1}, ν = 2, vacuum boundary conditions on LHS and RHS w/ 1×10^8 Initial particles

Method of Implementation	Compile Time [s]	Run Time [s]
Numba	6.25	179.36
PyKokkos	39.72	385.24
HCGL (PyCUDA)	2.45	160.53

1 single GPU (NVIDIA TeslaV100 at 1530MHz w/ 16GB) on 1 Lassen node

Conclusions and Future Work

Difficulty of Implementation

- Numba is simple
- Pykokkos is more difficult
- HCGL is very difficult but more performant

Future Work

- Complete transient tally implementation for all methods
- Test deployment on new hardware
- Accelerated as compered to what
- Implement Numba on MC/DC*
- *And much much much more!*

How to Slay the Dragons

- Data type hygiene
- Keep an errors diary
- Actually implement testing and run your tests after EVERY commit
- Use CONDA for everything possible
- Log build commands

Acknowledgments

Special thanks to:

- Post Docs: Ilham Variansyah; Aaron Reynolds
- CEMeNT Team and Associated Folks!
- *All the packages, their developers and the open science community*

Please Reach Out!

Contact

- Discord: `jpmorgan34#9493`
- Email: morgjack@oregonstate.edu
- Slack!

Repos

- MC/DC: https://github.com/CEMeNT-PSAAP/MCDC
- MC/DC – TNT: https://github.com/CEMeNT-PSAAP/MCDC-TNT

Citations

CEMENT

[1] Ganapol B.D., Baker, R. S., Dahl, J. A., & Alcouffe, R. E. (2001). Homogeneous Infinite Media Time-Dependent Analytical Benchmarks. *International Meeting on Mathematical Methods for Nuclear Applications*, *836*(December), 1–4.

[2] Awar, N. Al, Zhu, S., Biros, G., & Gligoric, M. (2021). A performance portability framework for python. *Proceedings of the International Conference on Supercomputing*, 467–478. https://doi.org/10.1145/3447818.3460376

[3] Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A LLVM-Based Python JIT Compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. https://doi.org/10.1145/2833157.2833162

[4] T. G. Mattson, T. A. Anderson, G. Georgakoudis, K. Hinsen, and A. Dubey, “PyOMP: Multithreaded Parallel Programming in Python,” *Comput. Sci. Eng.*, vol. 23, no. 6, pp. 77–80, Nov. 2021, doi: 10.1109/MCSE.2021.3128806.

[5] Witherden, F. D., Farrington, A. M., & Vincent, P. E. (2014). PyFR: An open source framework for solving advection-diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications*, *185*(11), 3028–3040. https://doi.org/10.1016/j.cpc.2014.07.011

[6] Witherden, F. (2021). Python at petascale with PyFR or: how I learned to stop worrying and love the snake. *Computing in Science & Engineering*, *9615*(c), 1–1. https://doi.org/10.1109/mcse.2021.3080126

Backmatter Slides

Neutron Transport Equation

CEMENT

Time Rate of Change

Streaming through space

Collision

$$\left(\frac{1}{v(E)}\frac{\partial}{\partial t} + \hat{\mathbf{\Omega}} \cdot \nabla + \Sigma_t(\mathbf{r}, E, t)\right)\psi(\mathbf{r}, E, \hat{\mathbf{\Omega}}, t) =$$

Particles produced from delayed fission

Particles produced from prompt fission

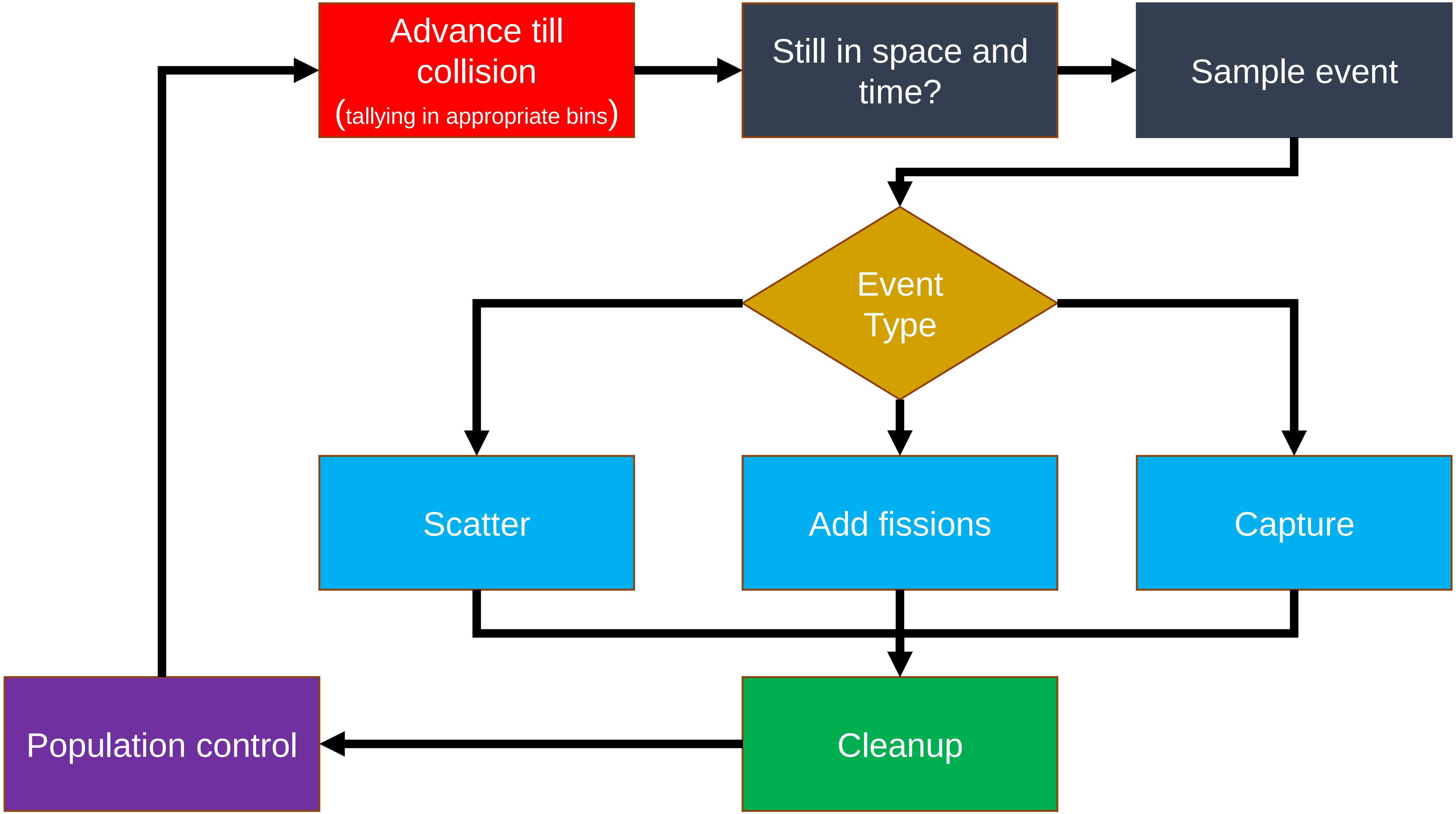
$$\frac{\chi_p(E)}{4\pi}\int_0^\infty dE'\,\nu_p(E')\,\Sigma_f(\mathbf{r}, E', t)\,\phi(\mathbf{r}, E', t) + \sum_{i=1}^{N}\frac{\chi_{di}(E)}{4\pi}\lambda_i C_i(\mathbf{r}, t) +$$

$$\int_{4\pi} d\Omega' \int_0^\infty dE'\,\Sigma_s(\mathbf{r}, E' \to E, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, t)\psi(\mathbf{r}, E', \hat{\mathbf{\Omega}}', t) + s(\mathbf{r}, E, \hat{\mathbf{\Omega}}, t)$$

In scattering

Direct source

MC Transport Flow Chart



Other Acceleration Techniques in Python

- Cython (able to use C++ standard parallelism)
- MPI4Py (Does not accelerate code, only runs more of it)
- DASK
- Python CUDA
- Pure Numpy / SciPy implementations (C under the hood)
- Build your own! (PyBind11, SciKit-Build)

Planed Explorations within MC/DC

- Fully transient Monte Carlo
- Intrusive UQ
- Dynamic Quasi Monte Carlo
- Dynamic Weight Windows
- Population Control Methods
- Python Based Parallelization
- Asynchronous GPU scheduling
- Machine Learning MPI scheduling

Future Development Path of MC/DC

CEMENT

1. Address Numba issues in MC/DC
   - Replace JITClass with Numba structured array
   - Runtime and memory profiling
2. Write event-based MC/DC (pure Python + MPI4Py)
   - Reuse and exploit existing MC/DC (history-based) modules
     - with Python decorator
3. Integrate findings from MC/DC-TNT
   - PyKokkos, Numba, PyOMP, Mako templating

AVURV Benchmark Description

We can simulate fission by having c>1

$$\Phi(x,t) = \frac{e^{-t}}{2t}\left[1 + \frac{c\,t}{4\pi}\left(1-\eta^2\right)\int_0^{\pi} \sec^2\left(\frac{u}{2}\right)\Re\left(\xi^2 e^{\frac{c\,t}{2}\left(1-\eta^2\right)\xi}\right)\,du\right] H(1-|\eta|)$$

NTE with initial source

$$\left[\frac{\partial}{\partial t} + \mu\frac{\partial}{\partial x} + 1\right]\Psi(x,\mu,t) = \frac{c}{2}\int_{-1}^{1}\Psi(x,\mu',t)\,d\mu + \frac{1}{2}\delta(x)\,\delta(t)$$

Science Python & HPC: Bigger Picture

- Enables rapid methods development for complex systems [7]
- Off the shelf codes for science applications available [8]
- There ***is*** a trade off in performance in benchmarks [9]
- A rich environment or high productivity in science [10]
- Allows nuclear folks to better interface with other fields!
- Can alleviate the need for C++ testbeds as initial performance analysis of methods can be examined

[7] Barba, L. A., Klockner, A., Ramachandran, P., & Thomas, R. (2021). Scientific Computing With Python on High-Performance Heterogeneous Systems. *Computing in Science & Engineering*. https://doi.org/10.1109/MCSE.2021.3088549
[8] Bogdan Opanchuk, Daniel Ringwalt, Lev E. Givon, & SyamGadde. (2021). *Reikna*(0.7.4). http://reikna.publicfields.net/en/latest/
[9] Oden, L. (2020). Lessons learned from comparing C-CUDA and Python-Numbafor GPU-Computing. *Proceedings -2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2020*, 216–223. https://doi.org/10.1109/PDP50117.2020.00041
[10] L. A. Barba, "The Python/Jupyter Ecosystem: Today's Problem-Solving Environment for Computational Science," in Computing in Science & Engineering, vol. 23, no. 3, pp. 5-9, 1 May-June 2021, doi: 10.1109/MCSE.2021.3074693.