Hardware Code Generation Techniques for Accelerating Python

Jackson P. Morgan, Todd S. Palmer, and Kyle E. Niemeyer

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

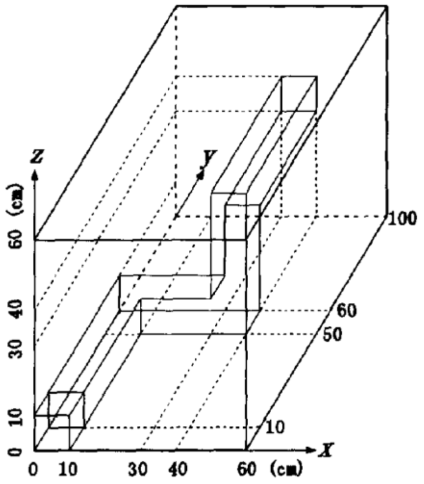
Annual Meeting of the American Nuclear Society, Anaheim, CA
June 16th, 2022

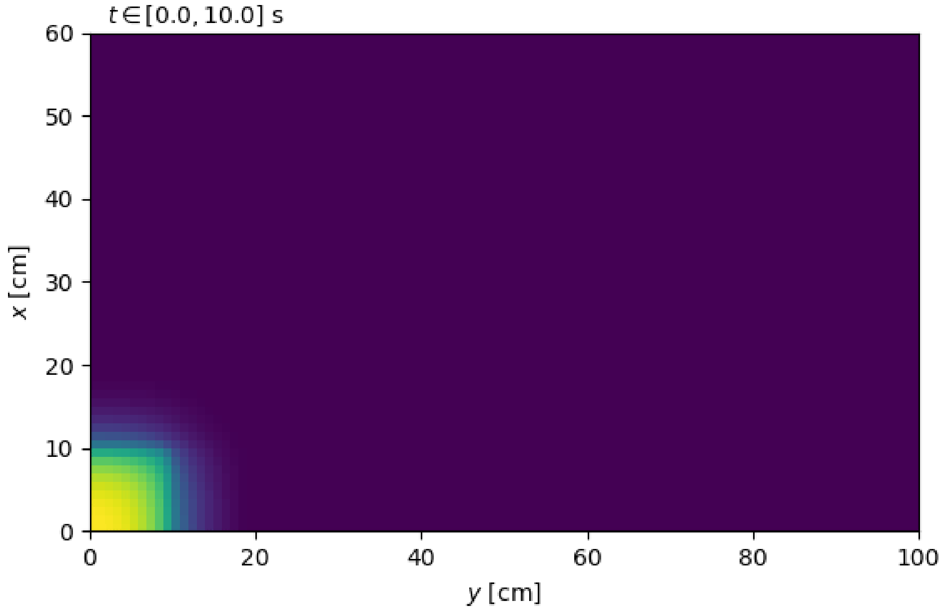
This work was supported by the Center for Exascale Monte-Carlo Neutron Transport (CEMeNT) a PSAAP-III project funded by the Department of Energy, grant number: DE-NA003967

Introduction

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 architectures
- Written in Python



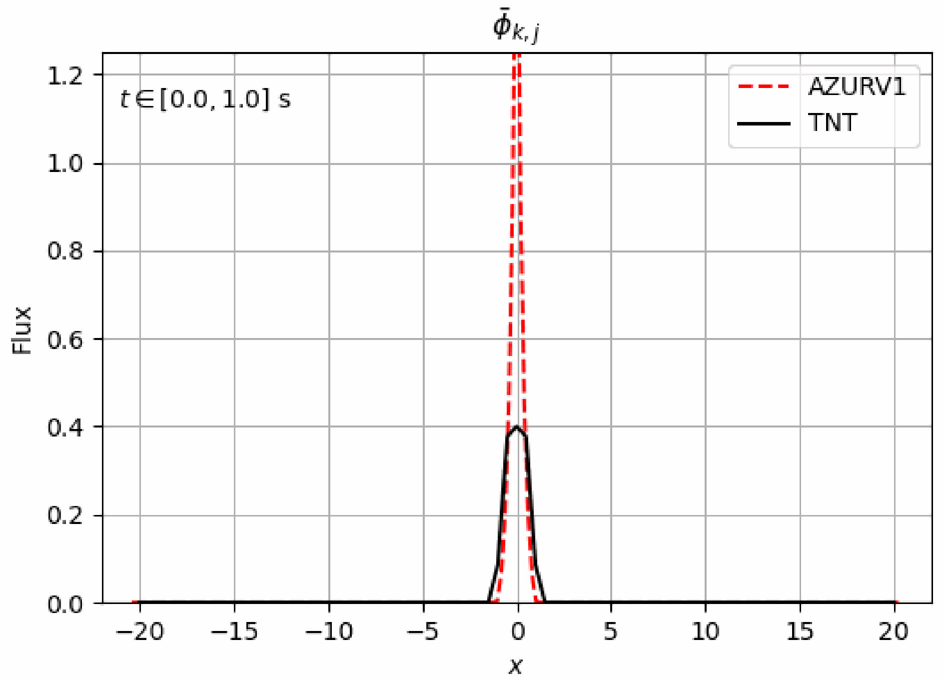


Kobyashi Problem: Image courtesy Ilham Variansyah

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 AZURV1 [1]



Vacuum

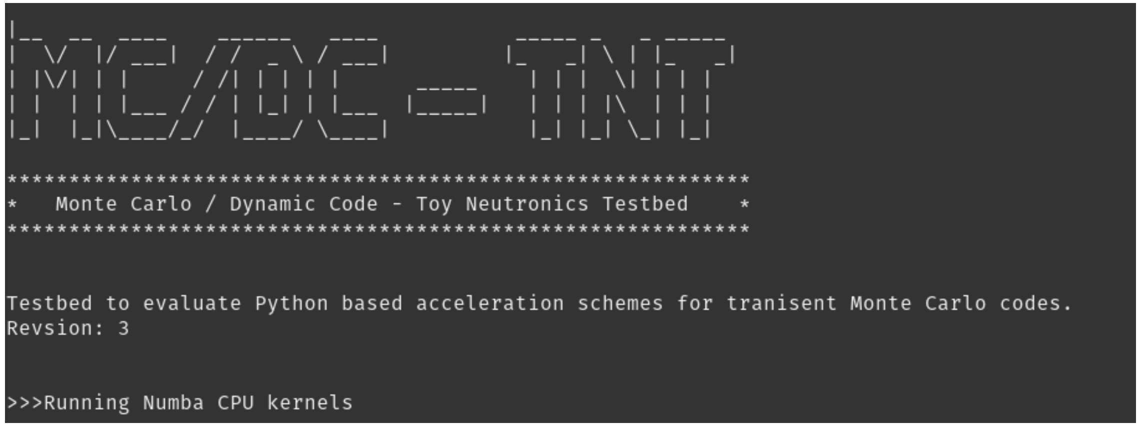
$L = 40$cm, $v = 2.3$, $\Delta x = 0.49$cm
$\Sigma_{cap} = \Sigma_{scat} = \Sigma_{fis} = 1/3 cm^{-1}$

Vacuum

MC/DC–TNT

CEMENT

- Modularity in mind
- 10 accelerated functions
- Advance implemented on hardware target

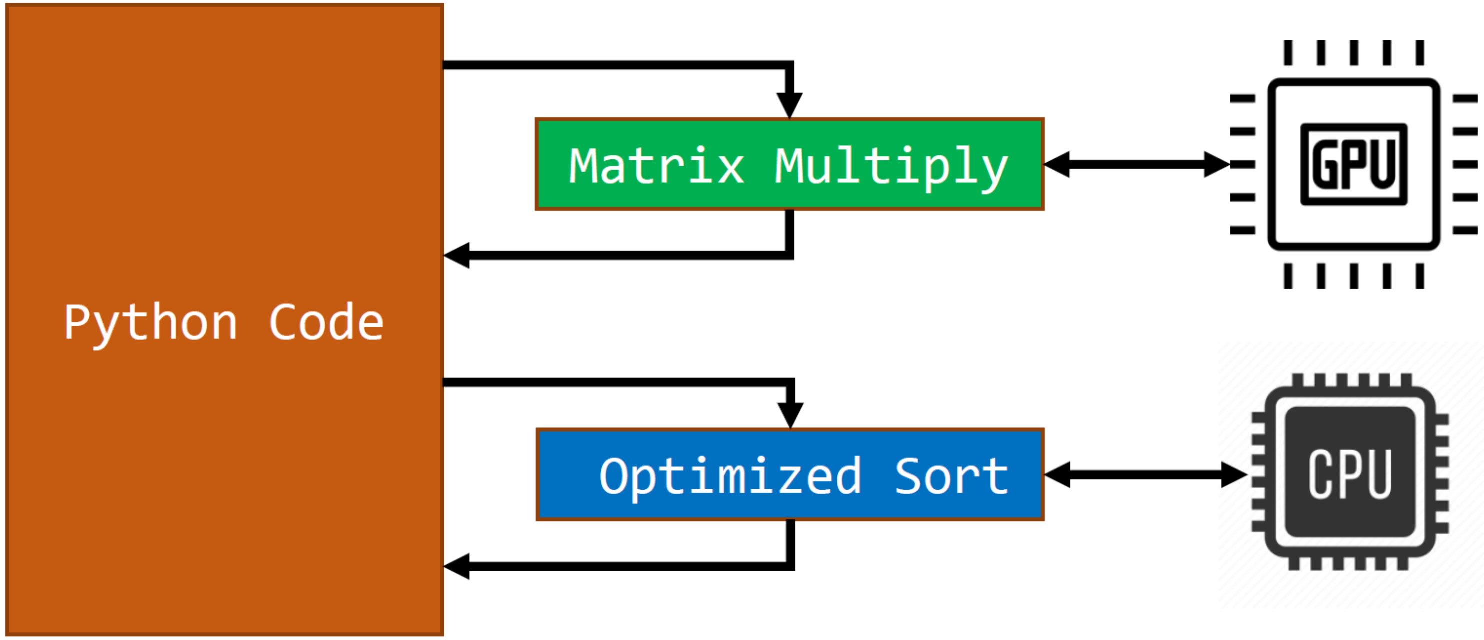



```
.
└── mcdc_tnt
    ├── numba_kernels
    │   ├── cpu
    │   ├── pyopenmp
    │   └── cuda
    ├── pk_kernels
    ├── hcgl_kernels
    └── pure_py_kernels
```

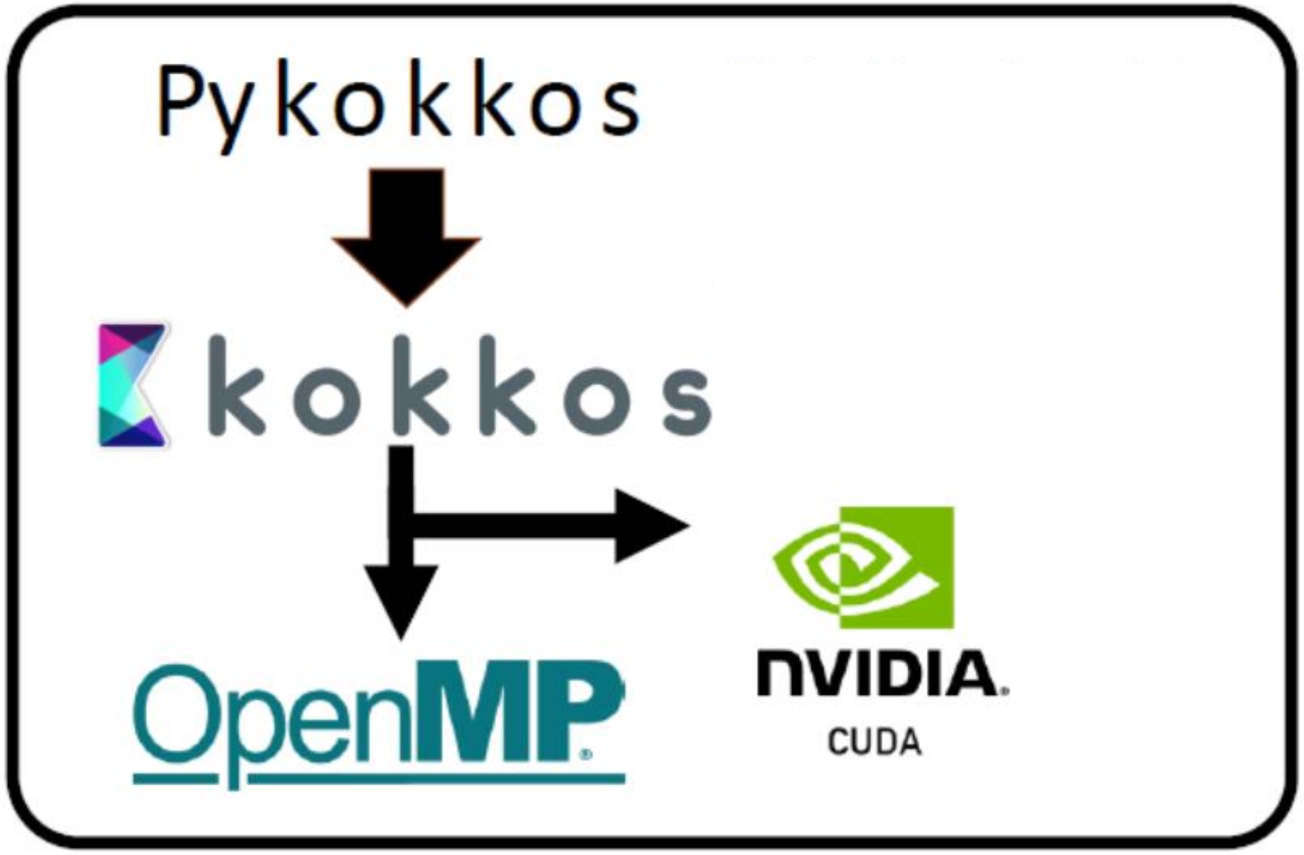
Methods of Acceleration

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



PyKokkos



- Python library that implements parts of Kokkos Portability framework [2]
- Brand new and under active development
- Currently building out HIP functionality

Numba + CUDA

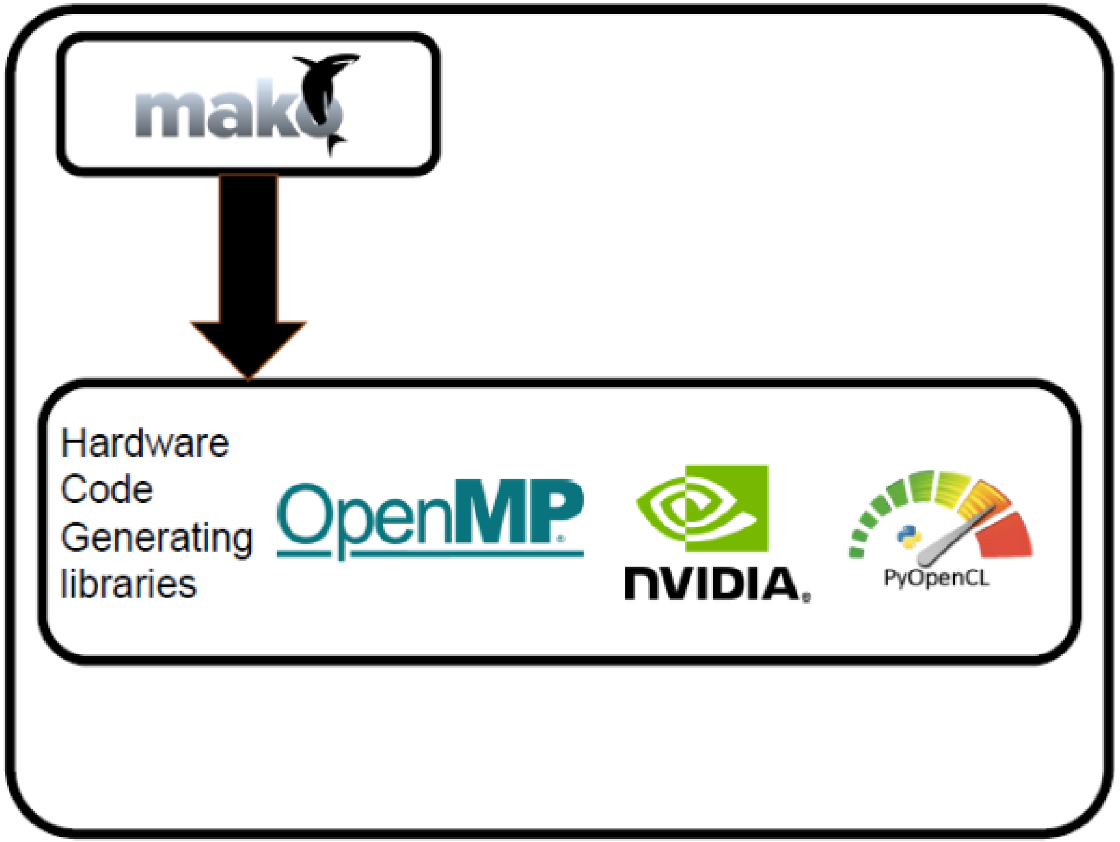
CEMENT

- 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]

HCGL and Mako Templating Engine

CEMENT

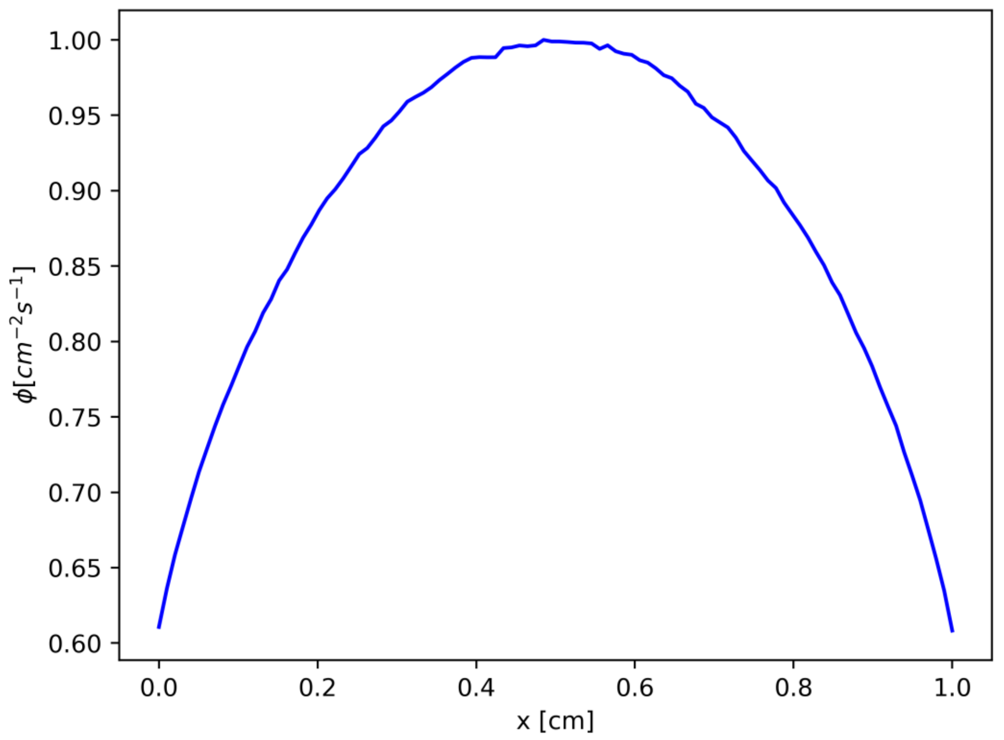
- Implemented on PyFR [5] at petascale [6]
- Code-generating libraries to compile templated code
- Our implementation plugs into PyCUDA and g++

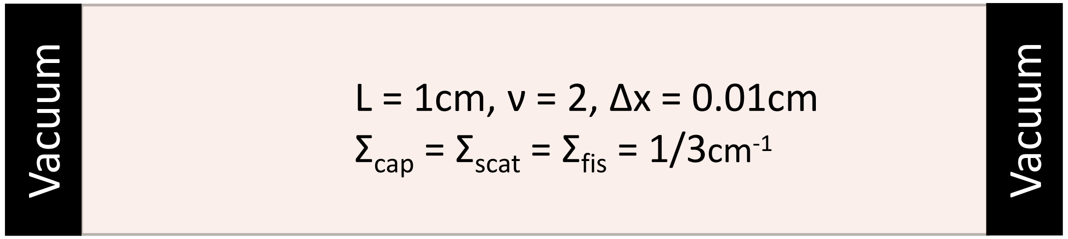


Results

Runtime Test Problem

- Sub-critical slab with initial population of 1×10^8 particles
- Validated with MC/DC
- Follow particles till death





Performance: CPU

Integration test problem: $L = 1cm$, $\Delta x = 0.01cm$, $\Sigma_f = \Sigma_c = \Sigma_s = 1/3\ cm^{-1}$, $\nu = 2$, 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

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

Performance

- Ease of use inversely proportional to max performance
- Every method is very performant! (relative to Python)
- Diminishing returns for more difficult implementations

Future Work

- Complete transient tally implementation for all methods
- Test deployment on new hardware
- Implement testbed in C++
- Implement Numba on MC/DC to accelerate pure Python code and gain *fine grain parallelism*

Acknowledgments

Special thanks to:

- Ilham Variansyah
- Aaron Reynolds
- CEMeNT Team and Associated Folks!

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

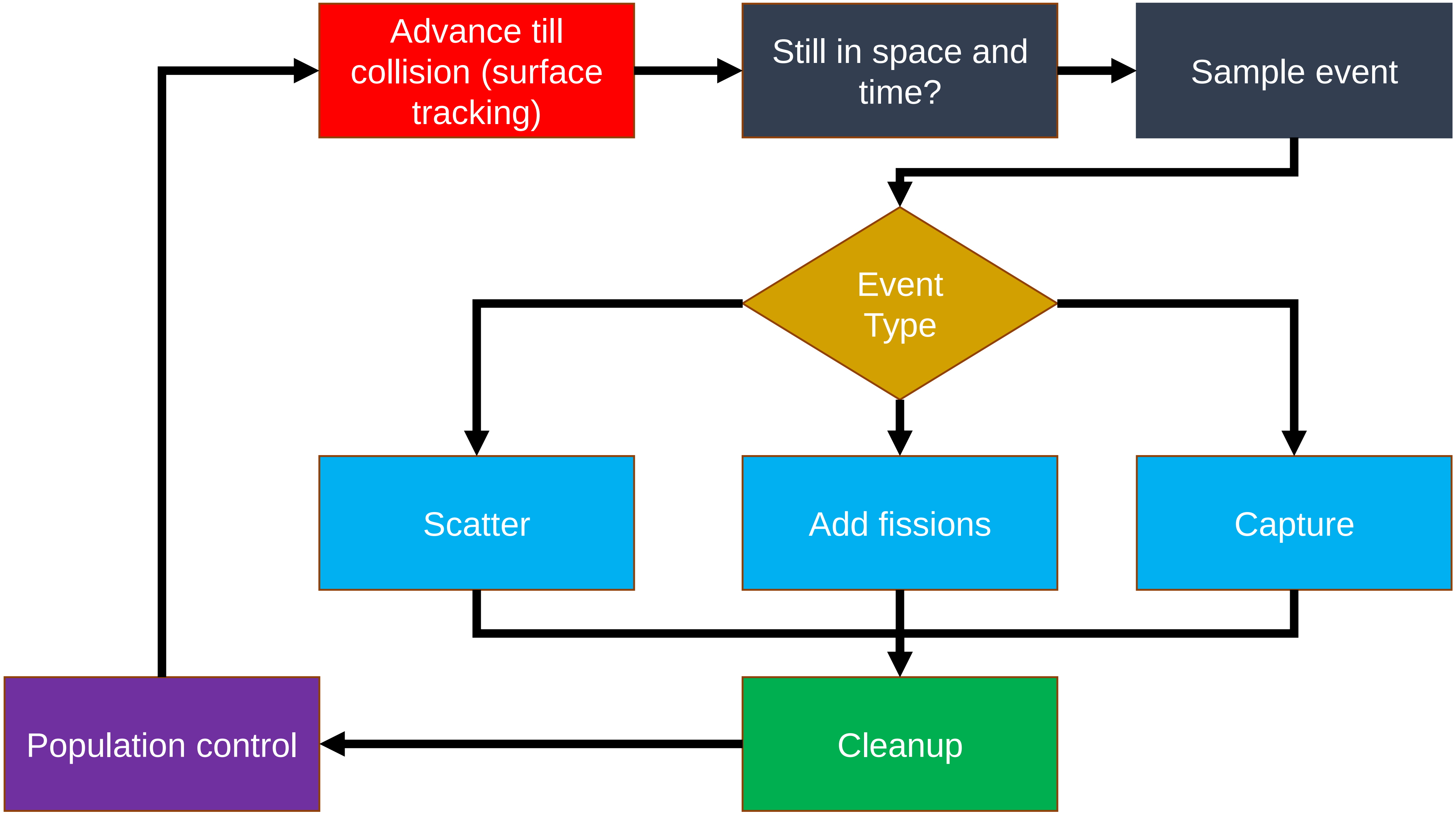
[4] T. G. Mattson, T. A. Anderson, G. Georgakoudis, K. Hinsen, and A. Dubey, “PyOMP: Multithreaded Parallel Programming in Python,” *Comput. Sci. Eng.*, vol. 23, no. 6, pp. 77–80, Nov. 2021, doi: 10.1109/MCSE.2021.3128806.

[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

Event-Based MC Transport Flow Chart



Other Acceleration Techniques in Python

- Cython (able to use C++ standard parallelism)
- PyCUDA and PyOpenCL (used but not directly)
- MPI4Py (Does not accelerate code, only runs more of it)
- Python CUDA
- Pure Numba / SciPy implementations (C under the hood)
- Build your own!

Planed Explorations within MC/DC

CEMENT

- 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

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 Descirption

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

CEMENT

- 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.