UNIVERSITY OF CAMBRIDGE

SCONE: an open-source Monte Carlo neutron transport code for research and teaching

Mikolaj A. Kowalski, **Paul M. Cosgrove**, Valeria Raffuzzi, Nathan Ravoisin, Eugene Shwageraus

23/06/2022

Engineering - Energy, Fluid dynamics and Turbo-machinery

Contents

- What is SCONE?
- Software design
- Validation
- Experience in code development:
  - Masters projects
  - Research projects
- Points for discussion

SCONE

Stochastic
Calculator
Of
Neutron Transport
Equation

- Particle Transport Monte Carlo Code for Nuclear Engineering Applications
- Academic focus -> targeting Masters students and PhDs
- **Development began in 2017**
- Designed for modification: Object-Orientation, well-defined abstractions
- Use: Teaching, New Algorithms Prototyping
- **Prioritise modifiability over performance… ish**

SCONE: software engineering

- Written in Fortran 2008:
  - Easy to learn & read without sacrificing performance
  - Informative compiler errors, easy-to-read standard
  - Reasonably well supported (OpenMP, OpenACC, …)
- Automated testing:
  - Unit and integration tests with pfUnit framework
- Open-source: the only open-source reactor physics code in the UK
- Accessible at **bitbucket.org/Mikolaj_Adam_Kowalski/scone**

The case for SCONE

Why not just use OpenMC?

- Transport() function is not virtual → There is ONE way to do the calculation
- From architecture (it seems):
    - Priority of OpenMC: Fast & Scalable Calculations of Reactor Problems
    - Not a priority of OpenMC: Supporting implementation of "wacky" (often not very useful in practice) ideas
    - E.G. Does not support delta-tracking in its current implementation
- NOTE: Not a criticism of OpenMC, but an observation that its priorities seems to be much different from SCONE's

How does SCONE fit?:

- Goal: Challenging to use, <u>Easy to modify,</u> Somewhat slow to execute
- Expose the user to some gritty details of MC methods in input files (similarly to OpenFOAM)
- Allow maximum flexibility in defining calculation sequences
- Define clear abstraction for interaction with key components (Nuclear Data, Geometry, Tallies).
- Try to optimise for speed of : *Idea → Prototype Implementation;* **not** *Input → Result*

SCONE: structuring

```
!!
!! Returns number of particles produced on average by the reaction
!!
!! Args:
!!   E [in] -> Incident particle energy [MeV]
!!
!! Result:
!!   Average number of particles for an incident energy E.
!!
!! Errors:
!!   If E is invalid (e.g. -ve) or outside of bounds of data table N = 0.0 is returned.
!!
function release(self, E) result(N)
  import :: defReal, uncorrelatedReactionCE
  class(uncorrelatedReactionCE), intent(in) :: self
  real(defReal), intent(in)                 :: E
  real(defReal)                             :: N
end function release
```

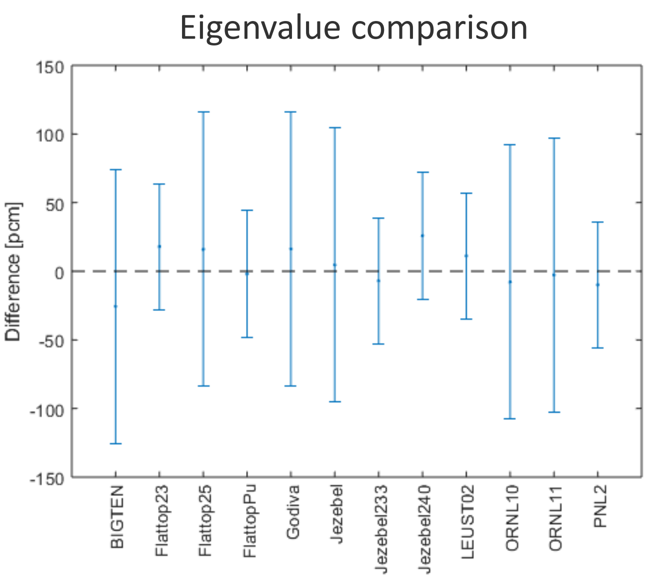
```
├── Apps
├── CollisionOperator
│   └── CollisionProcessors
├── DataStructures
├── Geometry
│   ├── csgRepresentation
│   ├── SurfaceObjects
│   │   ├── Cylinders
│   │   ├── Planes
│   │   └── TruncCylinders
│   └── Universes
├── Tallies
│   ├── TallyClerks
│   ├── TallyClerksOLD
│   │   └── keffClerk
│   ├── TallyFilters
│   ├── TallyMaps
│   └── TallyResponses
...
```

- Documentation comment for each procedure and class
- Based on Google docstring style for Python
- Hopefully clearly communicates specification for each code component
- Folder structure follows code structure to ease navigation

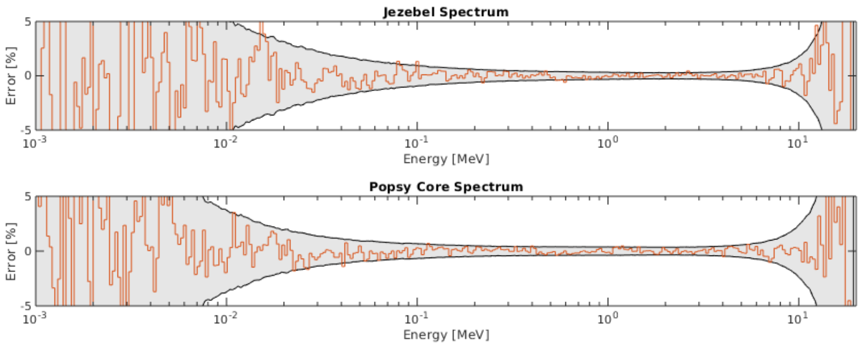
SCONE: validation and performance

Successfully tested on standard MCNP criticality benchmarks: compared to MCNP and/or Serpent reference results

Works on fast, thermal, uranium, plutonium, water, deuterium…



CPU time [min]	**Serpent**	**SCONE**
Flattop23	221	58
FlattopPu	247	70
Jezebel233	40	6
Jezebel240	33	8
LEUST02	615	291
PNL2	223	108



UNIVERSITY OF CAMBRIDGE

Masters projects

Experiences of SCONE Masters projects

- Very successful in short time (3 to 6 months)
- Showed that it is possible for Master's students to contribute to the development
- Positive feedback from the students on SCONE ☺

Lessons learned:

- Students tend to stay quiet: can spend a lot of time struggling with problems easy to correct if they ask for help
- Necessary to enforce good style

Previous projects:

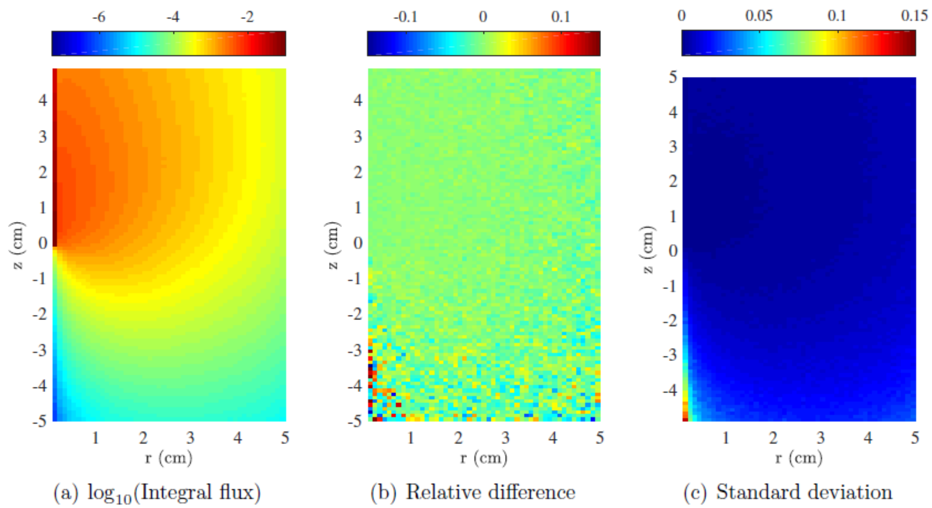
- Photon transport
- Unstructured meshes
- Alpha eigenvalue
- Isotopic depletion
- Photon-neutron coupling
- Implicit Monte Carlo
- Low population systems
- DBRC + OTF Doppler

Upcoming projects:

- CMFD
- Dynamic Monte Carlo

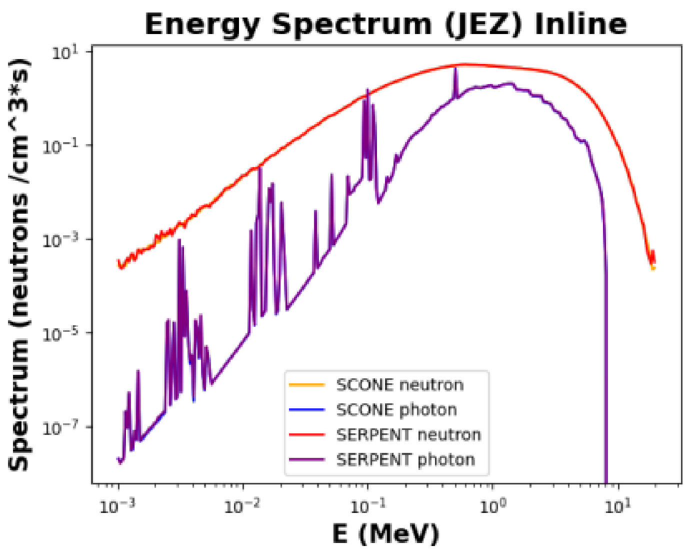
Masters projects

CASE 1: Photon transport



Log$_{10}$(Photon Flux) in an iron cylinder
10 MeV Beam (compared against Serpent)

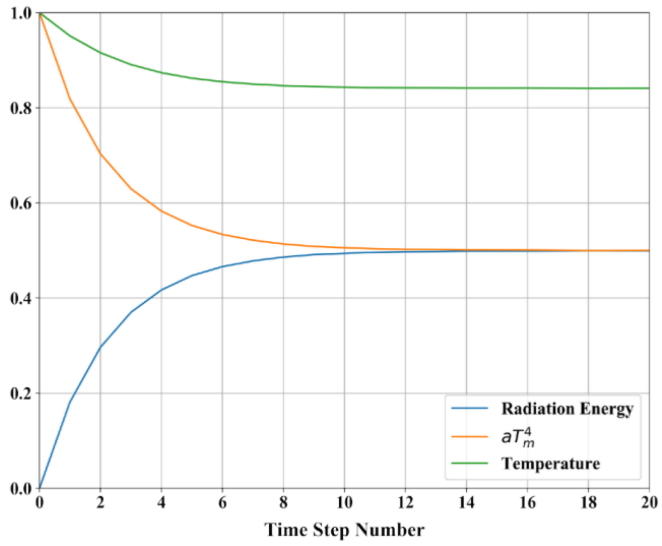
CASE 2: Photon-neutron coupling



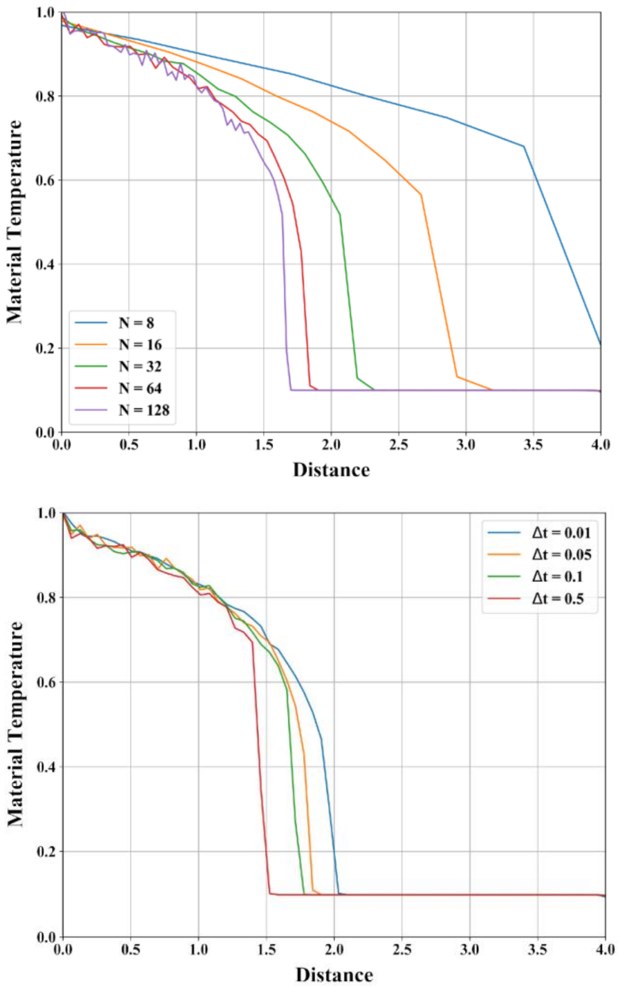
Masters projects

CASE 3: Thermal Radiative Transfer

Thermal equilibrium in an infinite problem



Teleportation error in a Marshak wave problem

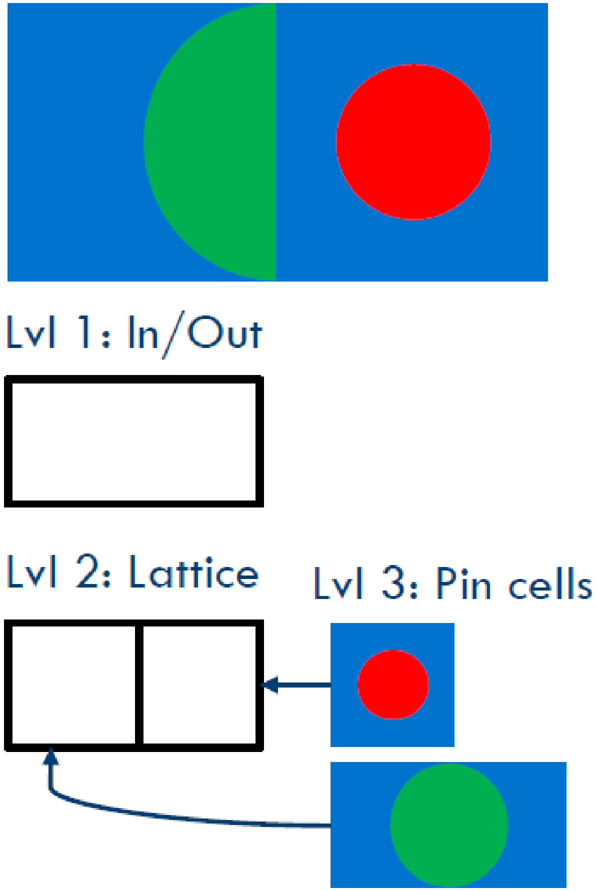


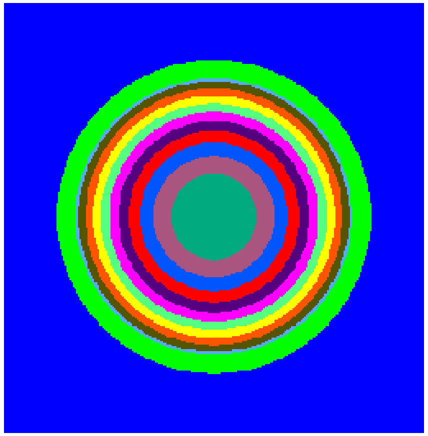
Research projects

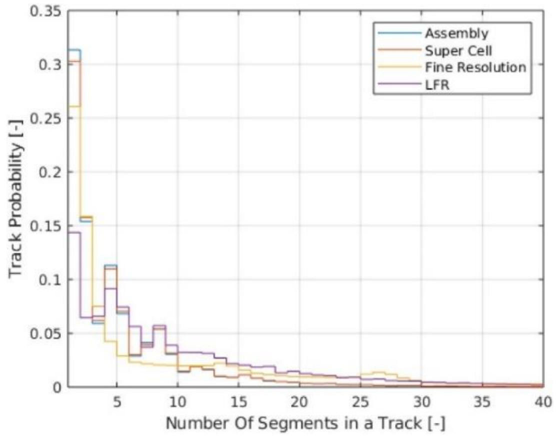
- PhD projects – acceleration methods
  - **Surface tracking distance caching**
  - MG – CE variable fidelity geometry: our motivating problem
  - Functional representation of cross sections
  - **Source convergence acceleration using MG**
- Also less conventional stuff
  - **Tramm's Random Ray Method**

Surface tracking distance caching

- Surface tracking demands checking the distance to the boundary of **every** universe at **every** particle flight
- Monte Carlo geometries usually composed of multiple nested 'universes'
- In reactor geometries, particles may cross many surfaces before colliding







UNIVERSITY OF CAMBRIDGE

Surface tracking distance caching

- Remember the distance to the boundary at higher universe levels and decrement them each flight
- Two days to implement and test
- Due to abstracting movement to a geometry function
- Easy to add a ‘move_withCache’ by duplicating, adding another argument and caching logic
- Also had an extra conditional in the transport operator
- A few tricks to handle FP error accumulation: periodic cache reset or Kahan summation

Without Cache

```
23.56%  __aceneutronnuclide_class_MOD_search
 6.76%  __aceneutrondatabase_class_MOD_updatemicroxss
 6.63%  __squarecylinder_class_MOD_distance
 5.74%  __ieee754_log_fma
 5.65%  __latuniverse_class_MOD_distance
 5.27%  __cylinder_class_MOD_distance
 2.91%  __particle_class_MOD_particlestate_fromparticle
```

With Cache

```
26.21%  __aceneutronnuclide_class_MOD_search
 7.08%  __aceneutrondatabase_class_MOD_updatemicroxss
 6.78%  __ieee754_log_fma
 5.88%  __cylinder_class_MOD_distance
 3.32%  __neutroncestd_class_MOD_scatterfrommoving
 3.12%  __particle_class_MOD_particlestate_fromparticle
 2.89%  __aceneutrondatabase_class_MOD_updatetotalmatxs
```

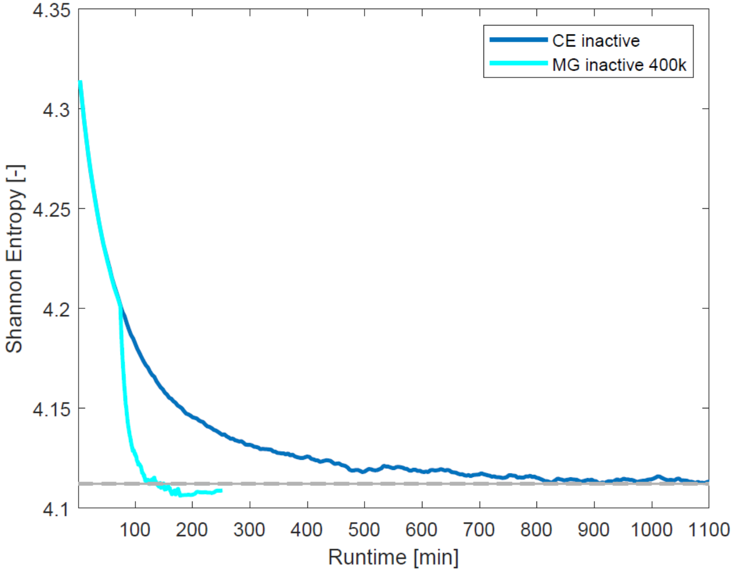
Only cylinder remains in the profile

Source convergence acceleration with MG inactive cycles

Converge the fission source with multi-group (MG) cross sections during the inactive cycles, and tally results with continuous energy (CE) cross sections during the active cycles.

SCONE already had support for both continuous energy (CE) and multi-group (MG) nuclear data (also at the same time): very quick to implement!

- Adding a tally to compute MG cross sections
- Adding subroutines to the physics package
  - Switch from CE to MG: initialise the material objects with the calculated MG cross sections, and convert the source neutron energy into an energy group
  - Switch from MG to CE: samples the source neutron energy from a CE distribution



The Random Ray Method

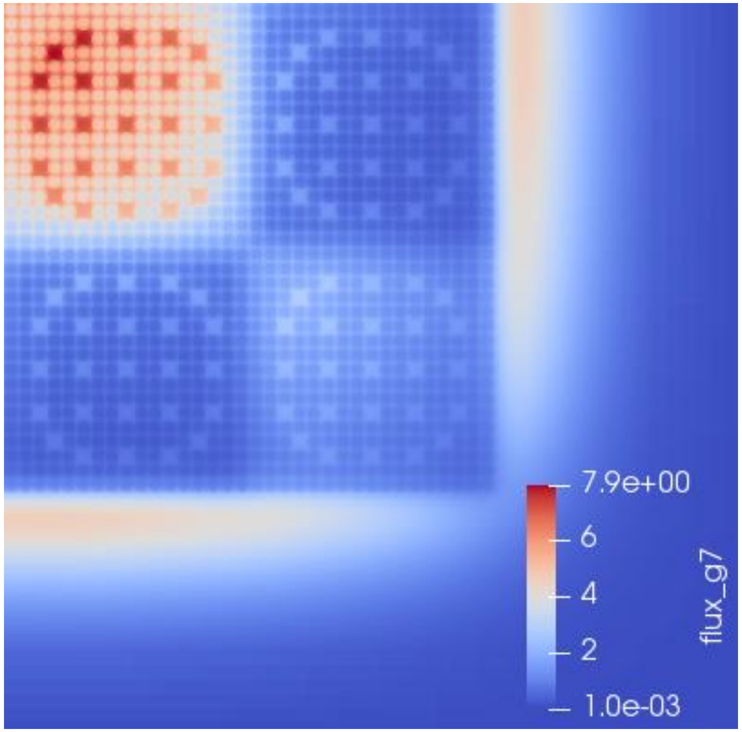
- Method of Characteristics transport solver (but stochastic)
- Changes to SCONE:

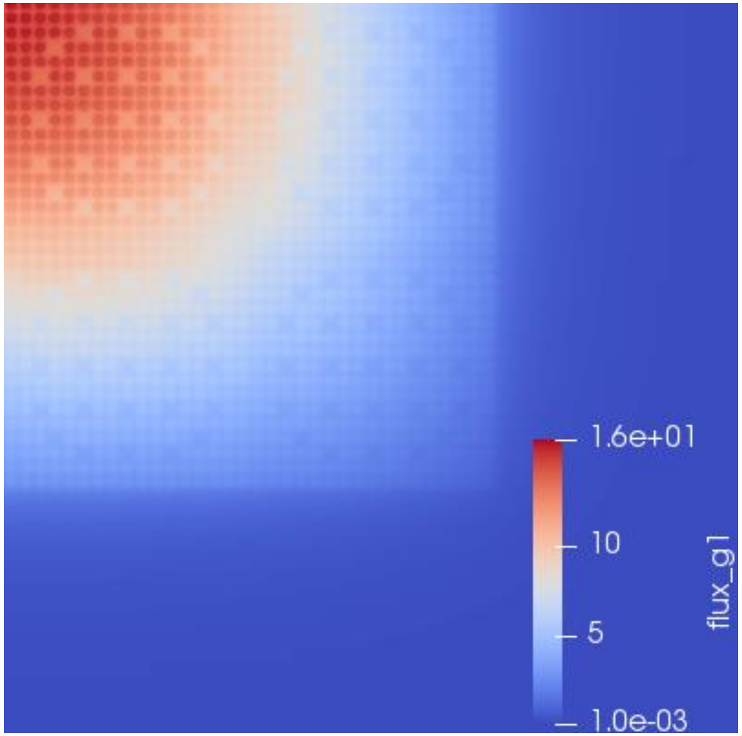
$$\Psi_{k,g}(s'') = \Psi_{k,g}(s')e^{-\tau_{k,i,g}} + \frac{Q_{i,g}}{\Sigma^T_{i,g}}(1 - e^{-\tau_{k,i,g}})$$

  - Copy paste a Physics Package, particle
  - Remove most of both while adding in flux vectors and the (very simple) Random Ray algorithm
  - Add a move subroutine with different logic for a vacuum boundary hit
  - Add Colin Josey's exponential evaluator
  - Make some long-overdue upgrades to **pin universes (azimuthal division)** and visualisation (easy plotting of flux maps)
  - Optional: mess around with distance caching to see if it helps
- Two weeks (thanks to plenty of guidance from John Tramm)
- Also shows limits of SCONE: not desirable to abstract everything away all the time

The Random Ray Method

- Result: pretty C5G7 flux plots and 3ns/integration (and a conference trip)
- Obviously not novel – but now we can research TRRM!





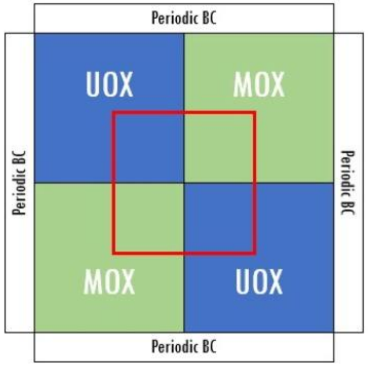
Discussion

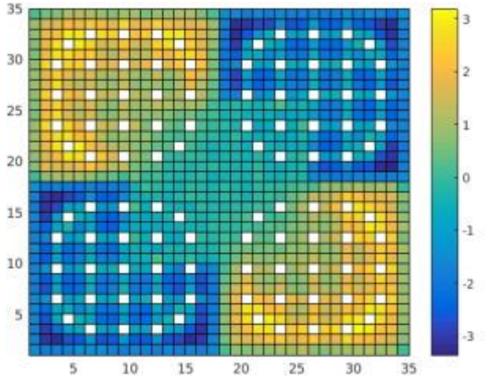
- Has anyone run a criticality calculation with SCONE?
  - Is SCONE easy to use? To understand?
  - Long term Fortran compiler support? Fortran tools and code reuse.
- Does anyone set, e.g., ‘write a Dancoff factor tally’ as a student assignment?
- How can we make SCONE more attractive to the research community?
- What experiences do others have of student code development projects?

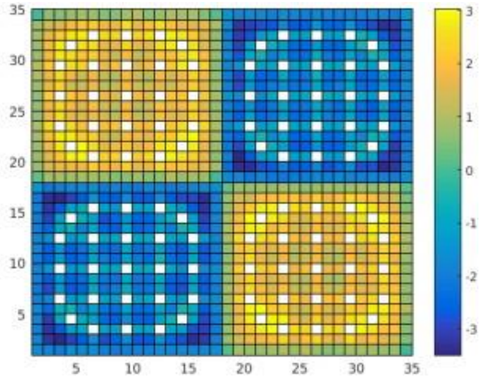
THANK YOU FOR YOUR ATTENTION

MG – CE variable fidelity geometry

- Different data types used in different geometrical regions
- Requires a clever fission source normalisation, different in the two regions!
- Heavily reduces computational time







Source convergence acceleration

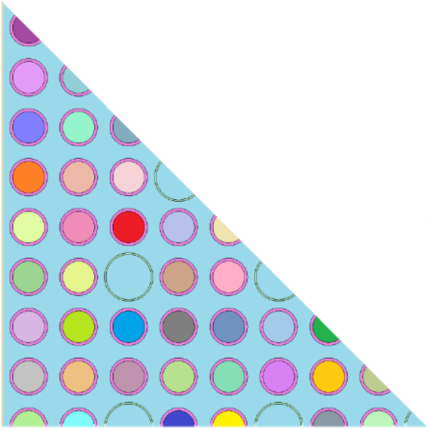
- Monte Carlo needs inactive and active cycles

  Source convergence — Tallying results

- The simulation takes long to converge in problems with high dominance ratio!

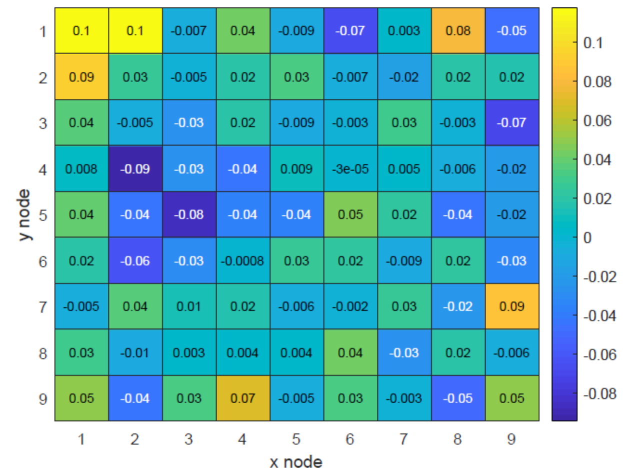
- Calculation route:
  - Calculate MG cross sections on-the-fly during few CE cycles
  - Switch to multi-group cross sections for the rest of the inactive cycles
  - Switch back to continuous energy for all the active cycles (to maintain full fidelity)

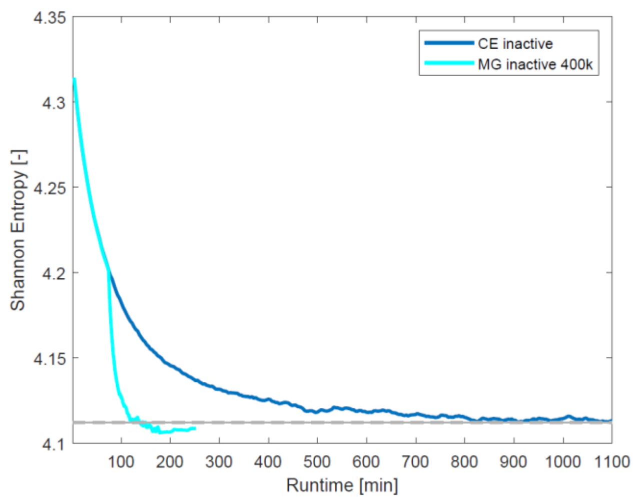
Burnt PWR assembly test case

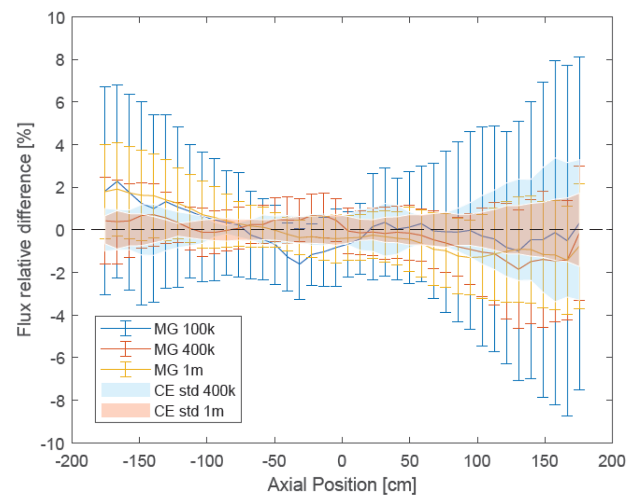


Source convergence acceleration

- Speed-up convergence by a factor of 5
- Memory usage doesn't grow substantially
- Final results are generally unaffected







Transport loop in eigenPhysicsPackage

```
neutron % pRNG => pRNG
call neutron % pRNG % stride(n)

! Obtain particle current cycle dungeon
call self % thisCycle % copy(neutron, n)

bufferLoop: do
  call self % geom % placeCoord(neutron % coords)

  ! Set k-eff for normalisation in the particle
  neutron % k_eff = k_new

  ! Save state
  call neutron % savePreHistory()

  ! Transport particle untill its death
  history: do
    call transOp % transport(neutron, tally, buffer, self % nextCycle)
    if(neutron % isDead) exit history

    call collOp % collide(neutron, tally, buffer, self % nextCycle)
    if(neutron % isDead) exit history
  end do history

  ! Clear out buffer
  if (buffer % isEmpty()) then
    exit bufferLoop
  else
    call buffer % release(neutron)
  end if

end do bufferLoop
```

Transport operator

```
!!   -> Assumes that particle moves without any external forces (assumes that particle
!!      moves along straight lines between colisions)
!!
!! Public interface:
!!   transport(p, tally, thisCycle, nextCycle) -> given particle, tally and particle dungeons
!!     for particles in this and next cycle performs movement of a particle in the geometry.
!!     Sends transistion report to the tally. Sends history report as well if particle dies.
!!   init(dict, geom) -> initialises transport operator from a dictionary and pointer to a
!!                       geometry
!!
!! Customisable procedures or transport actions
!!   transit(p, tally, thisCycle, nextCycle) -> implements movement from collision to collision
!!
type, abstract, public :: transportOperator
  !! Nuclear Data block pointer -> public so it can be used by subclasses (protected member)
  class(nuclearDatabase), pointer :: xsData => null()

  !! Geometry pointer -> public so it can be used by subclasses (protected member)
  class(geometry), pointer         :: geom        => null()

contains
  ! Public interface
  procedure, non_overridable :: transport

  ! Extentable initialisation and deconstruction procedure
  procedure :: init
  procedure :: kill

  ! Customisable deferred procedures
  procedure(transit), deferred :: transit

end type transportOperator
```

Delta tracking implementation

```
  !!
  !! Transport operator that moves a particle with delta tracking
  !!
  type, public, extends(transportOperator) :: transportOperatorDT
  contains
    procedure :: transit => deltaTracking
  end type transportOperatorDT

contains

  subroutine deltaTracking(self, p, tally, thisCycle, nextCycle)
    class(transportOperatorDT), intent(inout) :: self
    class(particle), intent(inout)            :: p
    type(tallyAdmin), intent(inout)           :: tally
    class(particleDungeon), intent(inout)     :: thisCycle
    class(particleDungeon), intent(inout)     :: nextCycle
    real(defReal)                             :: majorant_inv, sigmaT, distance
    character(100), parameter :: Here = 'deltaTracking (transportOIperatorDT_class.f90)'

    ! Get majornat XS inverse: 1/Sigma_majorant
    majorant_inv = ONE / self % xsData % getMajorantXS(p)

    DTLoop:do
      distance = -log( p% pRNG % get() ) * majorant_inv

      ! Move partice in the geometry
      call self % geom % teleport(p % coords, distance)

      ! If particle has leaked exit
      if (p % matIdx() == OUTSIDE_FILL) then
```