SNAP: SN (Discrete Ordinates) Application Proxy – An Overview

Joe Zerr & Randy Baker

CCS-2

LA-UR-13-23061

January 8, 2015

UNCLASSIFIED

Outline

- What is SNAP?
- What is discrete ordinates transport?
- What is the parallelization model?
- How can it be tested?
- What comes next?

WHAT IS SNAP?

SN Application Proxy

- Proxy for discrete ordinates particle transport
- Modeled off LANL production code PARTISN
- Update to Sweep3D for hybrid architectures
- Open Source: modifications to physics operators/data → not real transport code
- Reliable approximation of a transport code's data layout, movement, workload

SNAP *vs* PARTISN

SNAP

- Fortran 90/95+MPI +OpenMP
- ~3,000 lines of source code
- Simple data structures
- Fabricated algorithms to set needed constants
- Isolates particular solution algorithm, parallel model

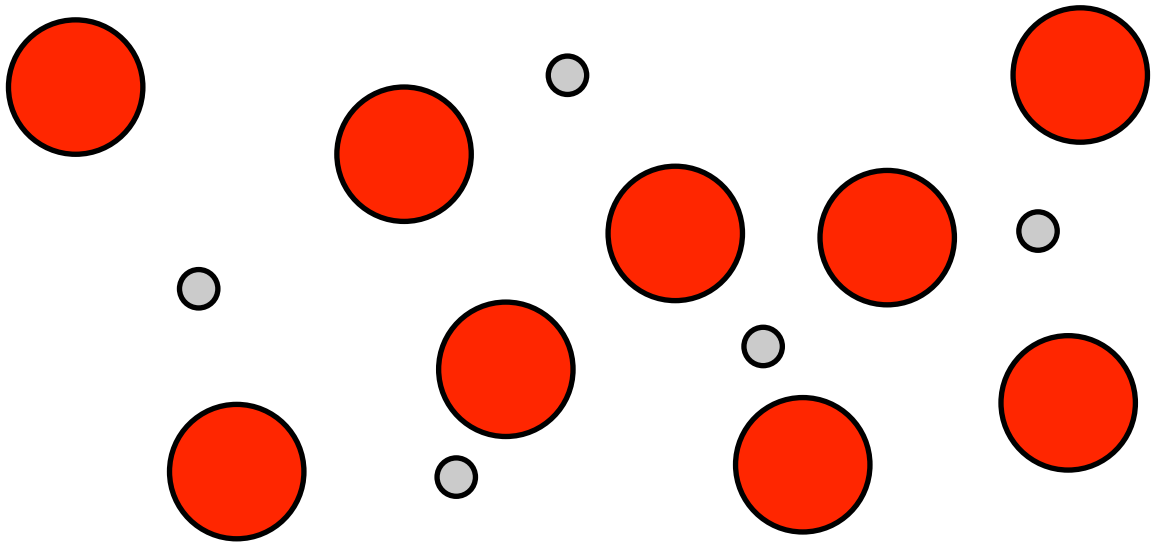
PARTISN

- Fortran 90/95+MPI +OpenMP
- 100,000+ lines of source code
- Derived data types
- Validated nuclear data + flexible user input
- Multiple discretization techniques, solution algorithms, parallel models

WHAT IS DISCRETE ORDINATES TRANSPORT?

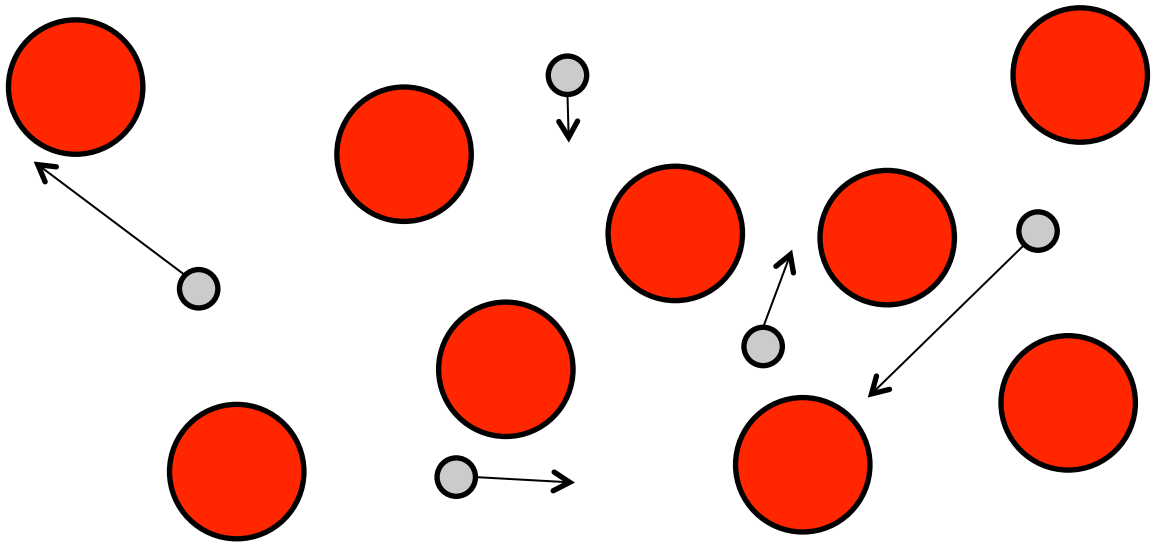
Neutrons interact with atomic nuclei

- Imagine neutrons (blue) moving through space interacting with matter/atoms (red)



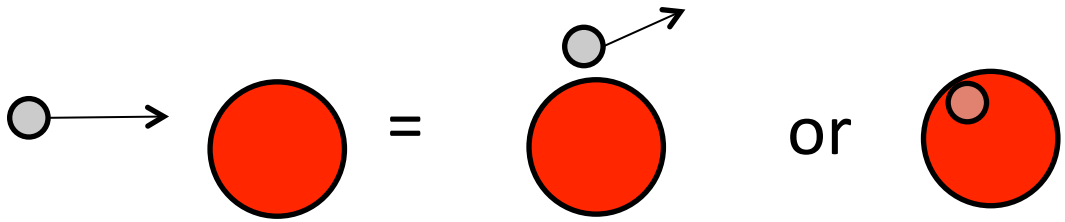
Neutrons interact with atomic nuclei

- Assume atoms are stationary while neutrons move in any direction at different speeds



Neutrons interact with atomic nuclei

- Assume neutrons do not interact with each other but can be scattered or absorbed by atomic nucleus



Linear Boltzmann Equation

- The physical processes can be expressed by the first order partial differential equation
- Solve for "flux", *f*

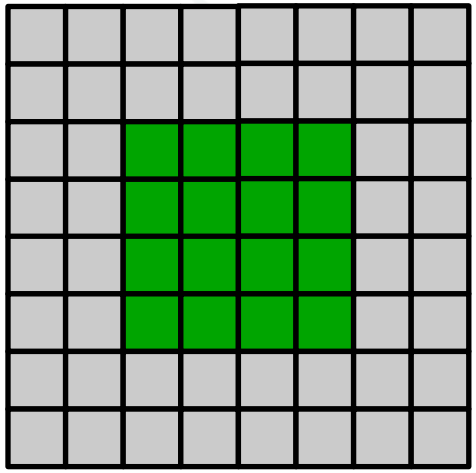
$$\frac{1}{v(E)}\frac{\partial}{\partial t} f\left(\vec{r},\hat{\Omega},E,t\right)+\hat{\Omega}\cdot\vec{\nabla}f+\sigma_t\left(\vec{r},E,t\right)f=\mathbf{S}\left(\vec{r},\hat{\Omega}'\rightarrow\hat{\Omega},E'\rightarrow E,t\right)f+q\left(\vec{r},\hat{\Omega},E,t\right)$$

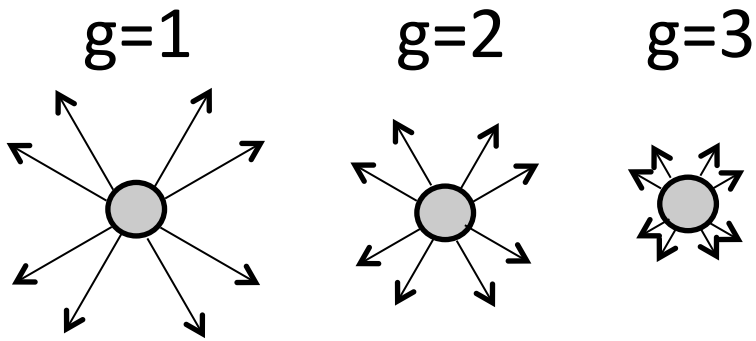
Time rate of change	Particle streaming	Total interaction	Particle re-emission by scattering	Inhomogeneous source

- SNAP: vacuum BCs, zero initial condition

Deterministic solution with discrete ordinates (SN)

- Apply a spatial mesh
- Different materials in spatial cells, (*i,j,k*)
- Particles binned into energy "groups", *g*
- Restrict directions to a pre-determined set of discrete ordinates with associated weights, Ω_n and w_n





Seven-Dimensional Phase Space

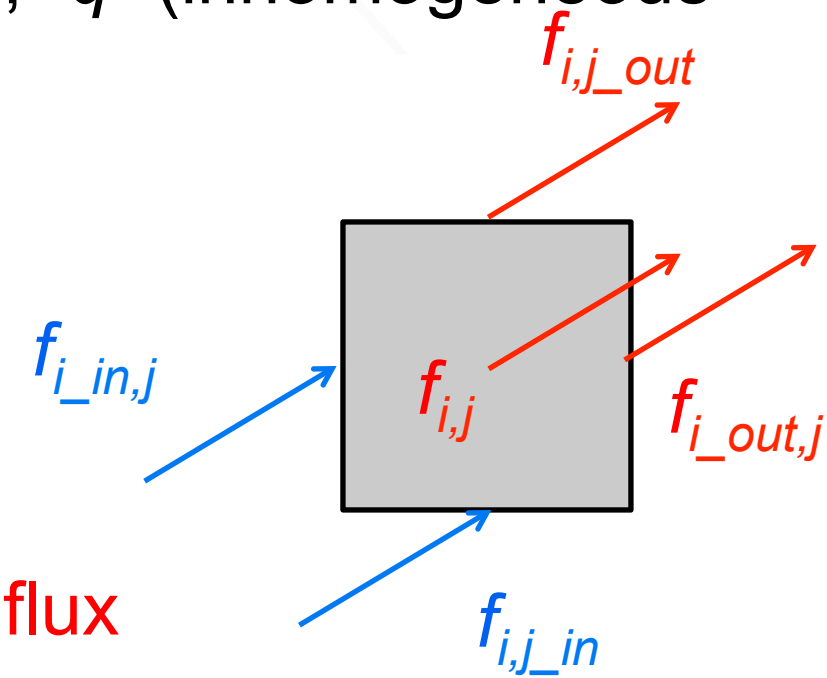
- A 3-D (spatially), time-dependent problem has a 7-D solution, *f*
  - 3 in space: x, y, z
  - 2 in angle: octants, angles
  - 1 in energy: groups
  - 1 in time: time steps

Outer/Inner Solution Strategy

- Each time step is solved quasi-statically
- Outer iterations
  - Scattering process changes the speed of particles in the system
  - Resolves out-of-group sources
- Inner iterations
  - Scattering process also changes the direction while leaving particles in the same group
  - Resolve flux for fixed source + within-group scattering source
  - Compute intensive kernel

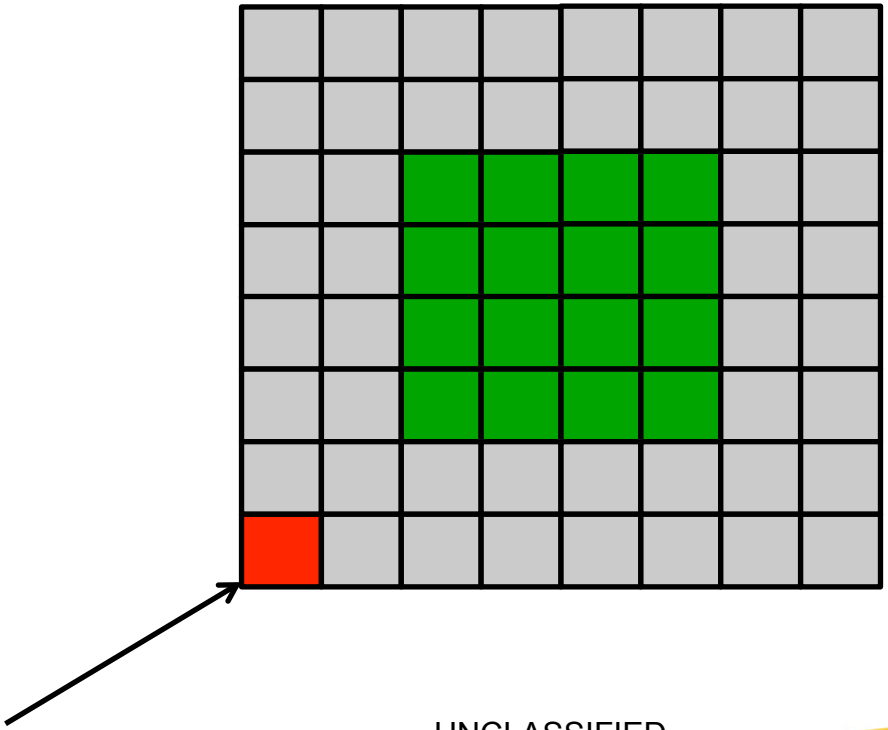
Transport Mesh Sweep

- Solve TE with fixed source, "*q*" (inhomogeneous + out-of-group)
  - For all groups, *g*
  - For all angles, Ω_n
- For a single cell
  - Know incoming flux
  - Solve center + outgoing flux
- Outgoing becomes incoming downstream

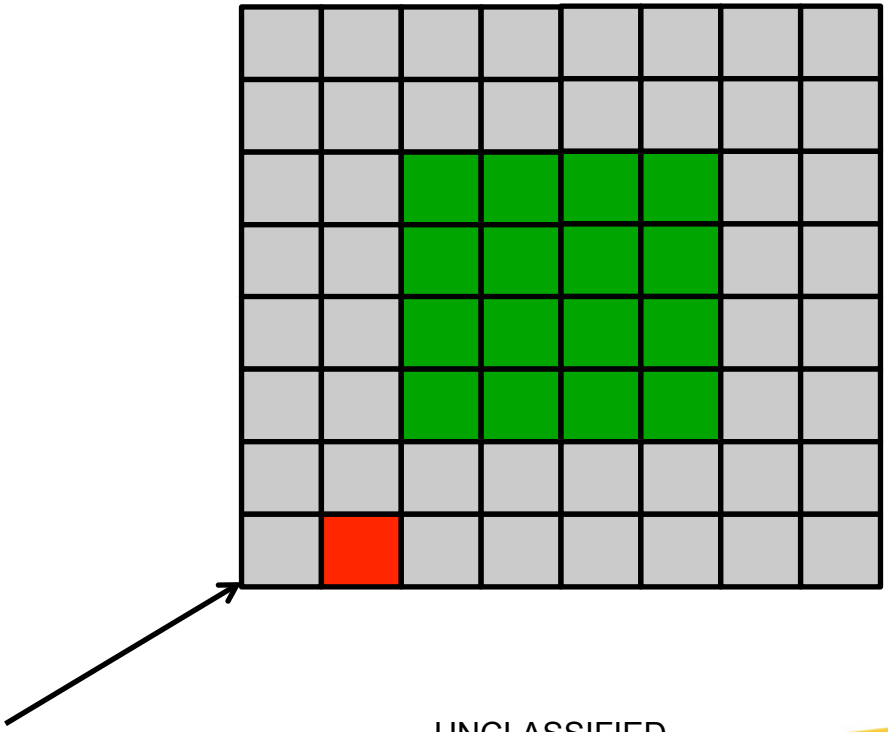


Transport mesh sweep

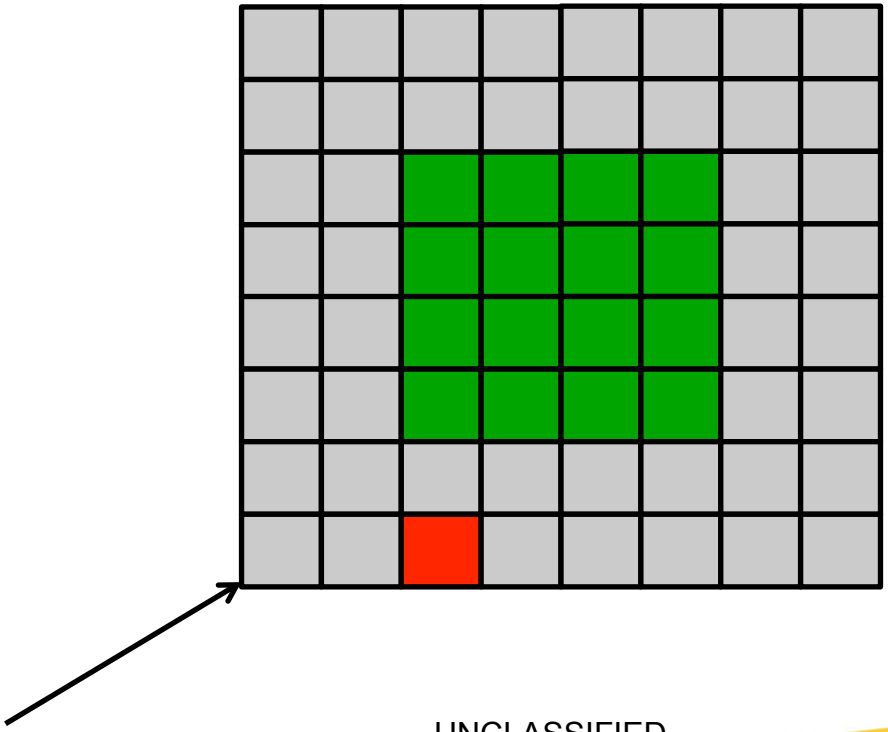
- March across the entire spatial mesh for each group-angle pair



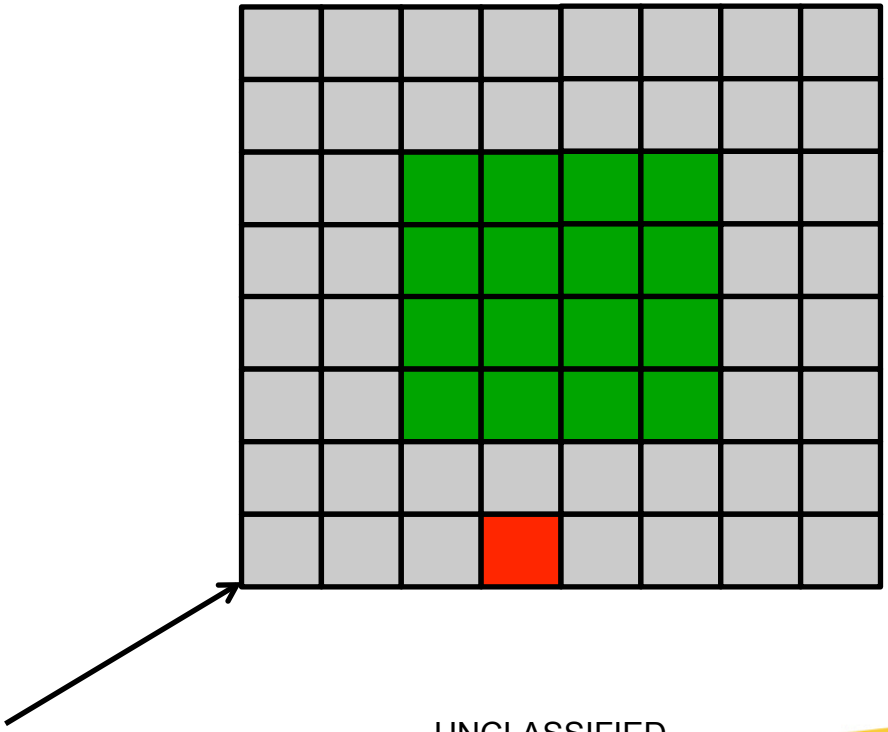
Transport mesh sweep



Transport mesh sweep

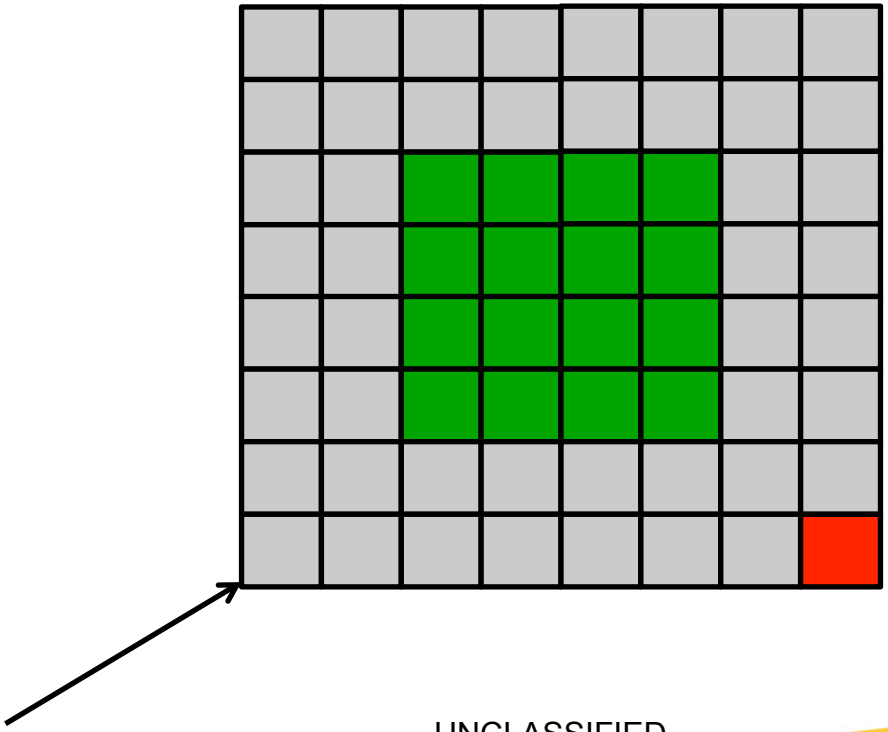


Transport mesh sweep



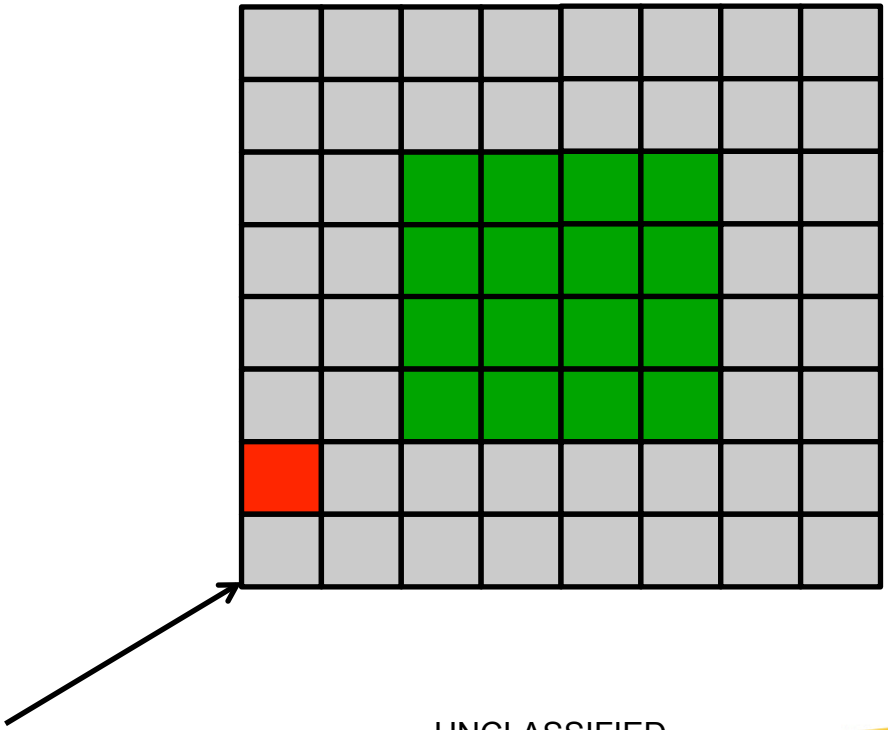
Transport mesh sweep

- … reach the end of the row



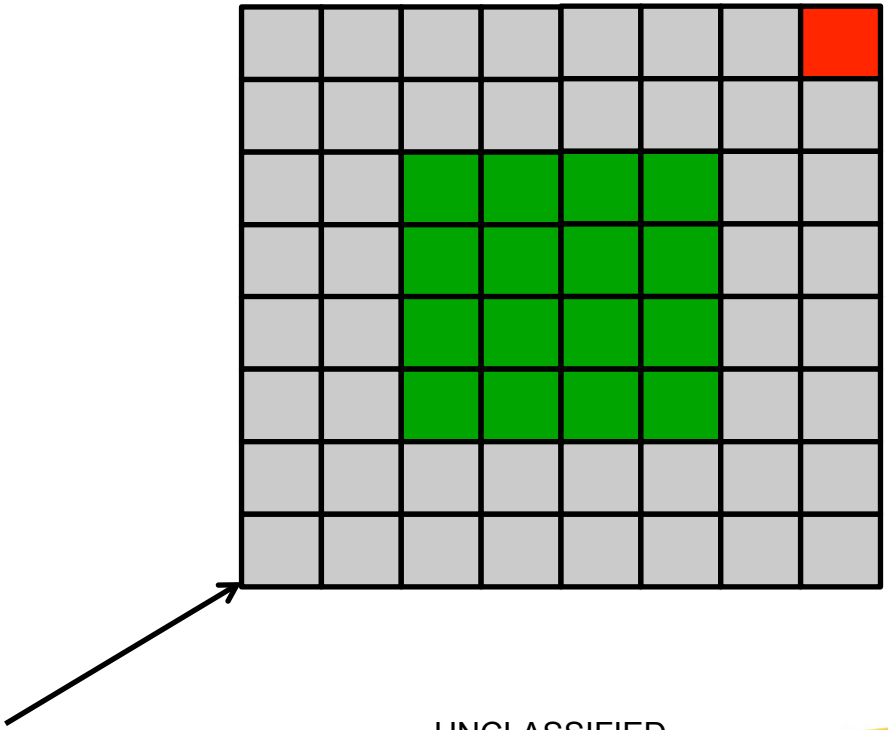
Transport mesh sweep

- start row of next column



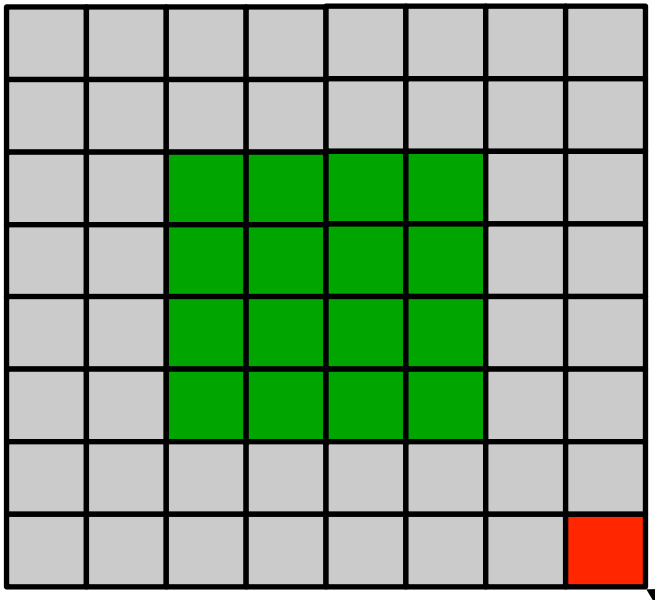
Transport mesh sweep

- … reach end of plane



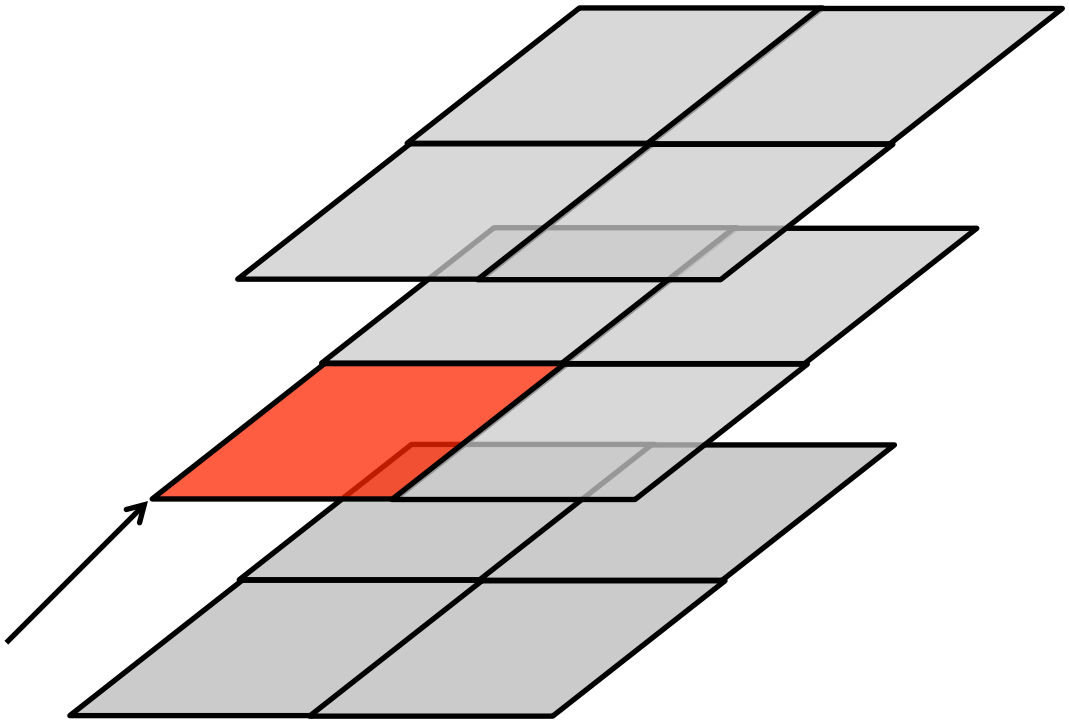
Transport mesh sweep

- In 2-D, start another quadrant
- When quadrants are done, go to next group



Transport mesh sweep

- In 3-D, go to next plane



Flux Fixup

- Discretization can lead to unphysical negative fluxes at mesh boundaries
- Non-linear fix
  - Check for negativity at boundary
  - Set to zero
  - Compute a new, rebalanced center flux
  - Multi-pass: rebalance due to fixup in one dimension requires recomputation in other dimensions

Putting it all together

- Start with initial guess
- Compute out-of-group sources
- Mesh sweep over all angles, groups
- Update within-group source
- Iterate until converged or limit
- Update out-of-group sources
- Return to mesh sweep + within-group sources
- Iterate until outer source converges or limit

WHAT IS THE PARALLELIZATION MODEL?

Distributing data

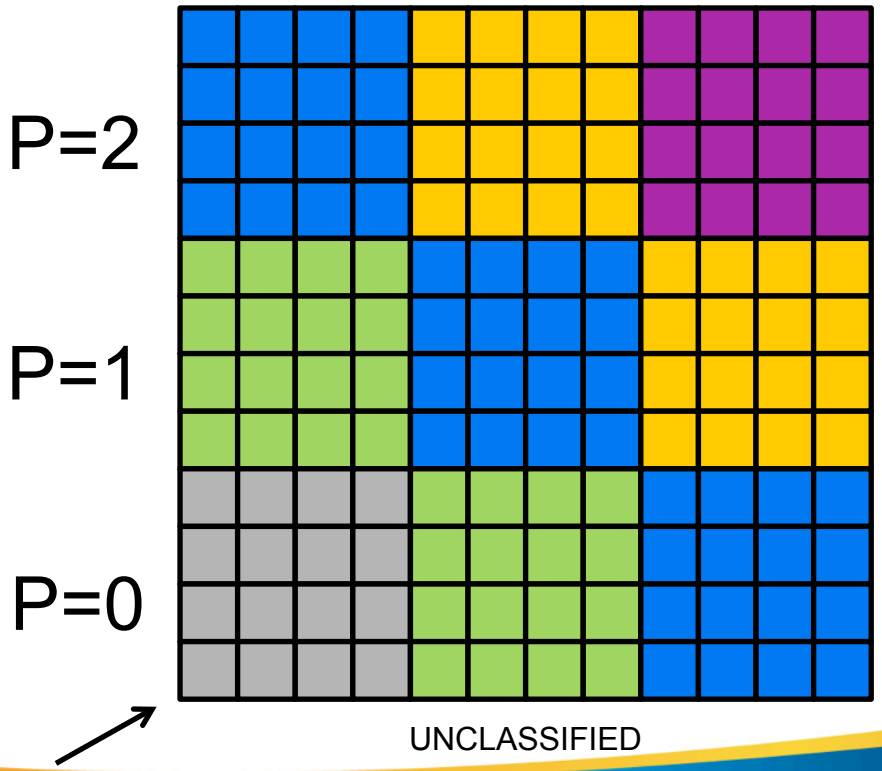
- Problem resolution
  - Tens/Hundreds energy groups
  - Hundreds/Thousands discrete directions
  - Millions of spatial cells
- Use spatial dimension to exploit distributed memory aspect of architecture
- MPI for data communication

KBA mesh sweep

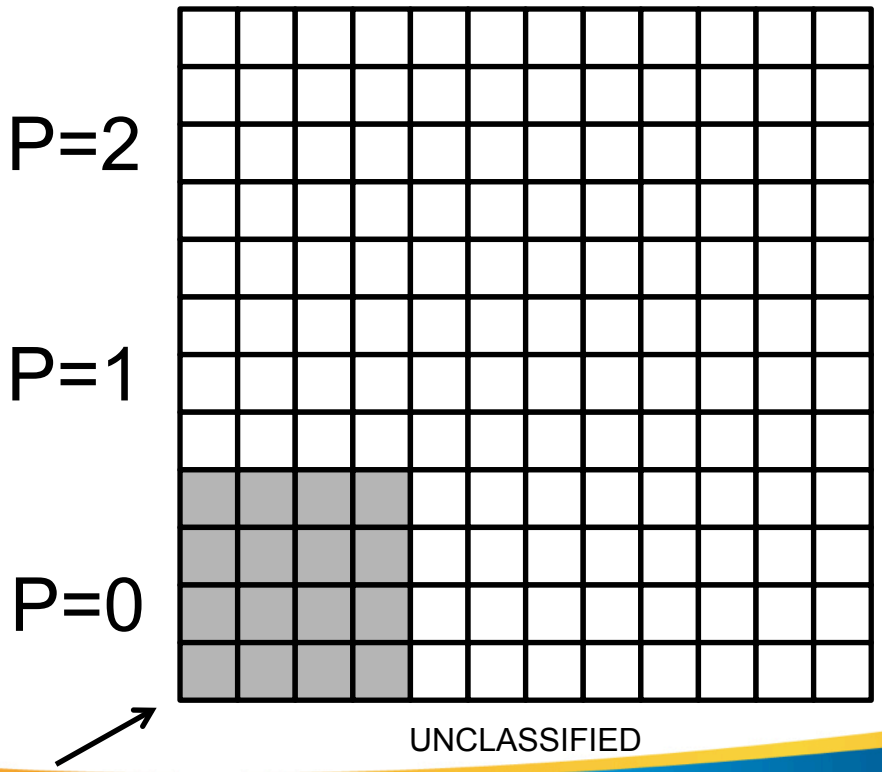
- Apply *n*-1 decomposition to *n*-dimensional (spatially) problem
- Break up non-decomposed axis into work chunks
- Perform transport sweeps for single chunk
- Copy outgoing → incoming flux on process
- Communicate outgoing → incoming between processes
- Startup/shutdown penalties
- Approximate performance: Parallel Computational Efficiency (PCE)

KBA mesh sweep

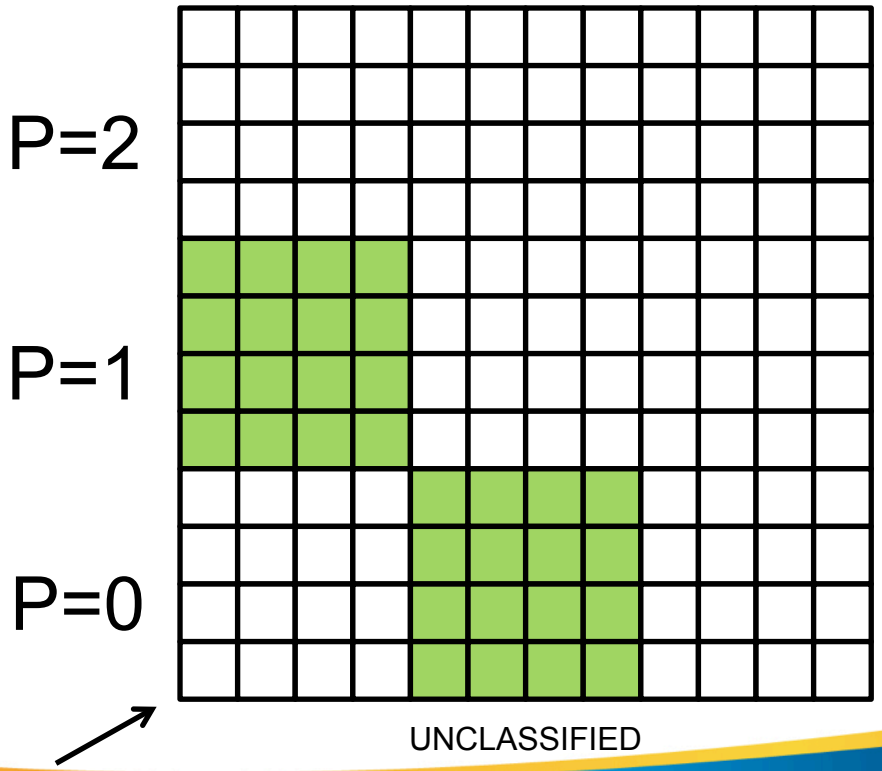
- Work chunks along same diagonal computed concurrently



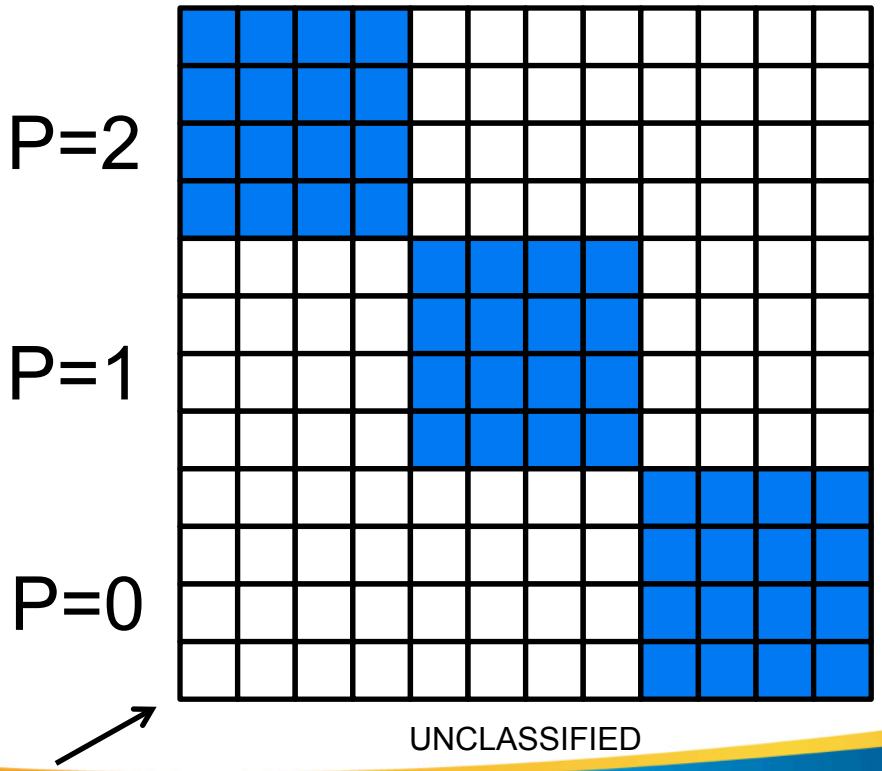
KBA mesh sweep



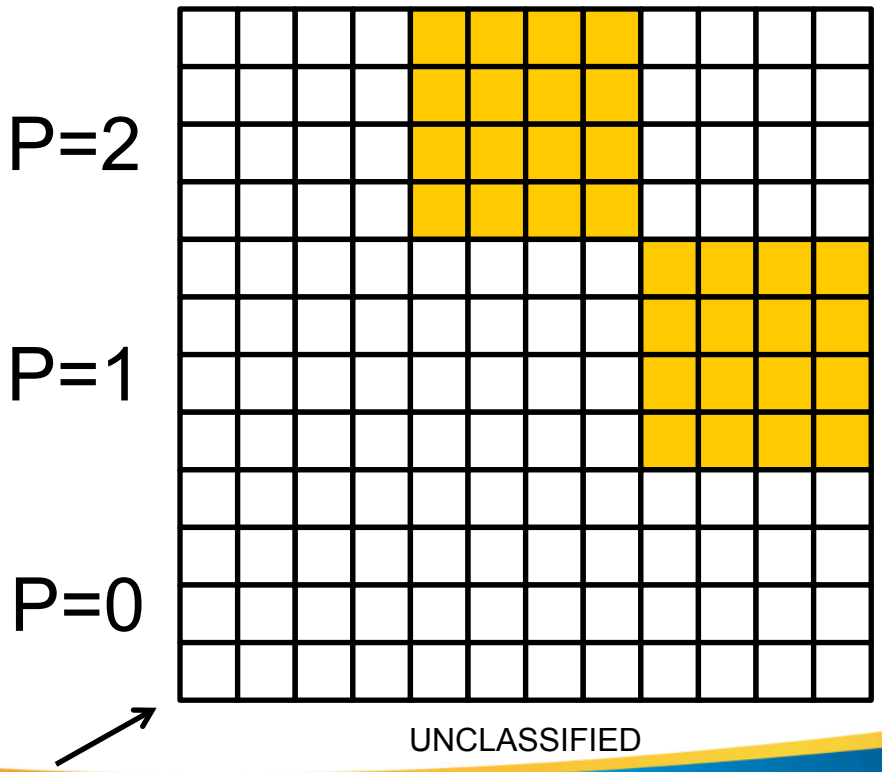
KBA mesh sweep



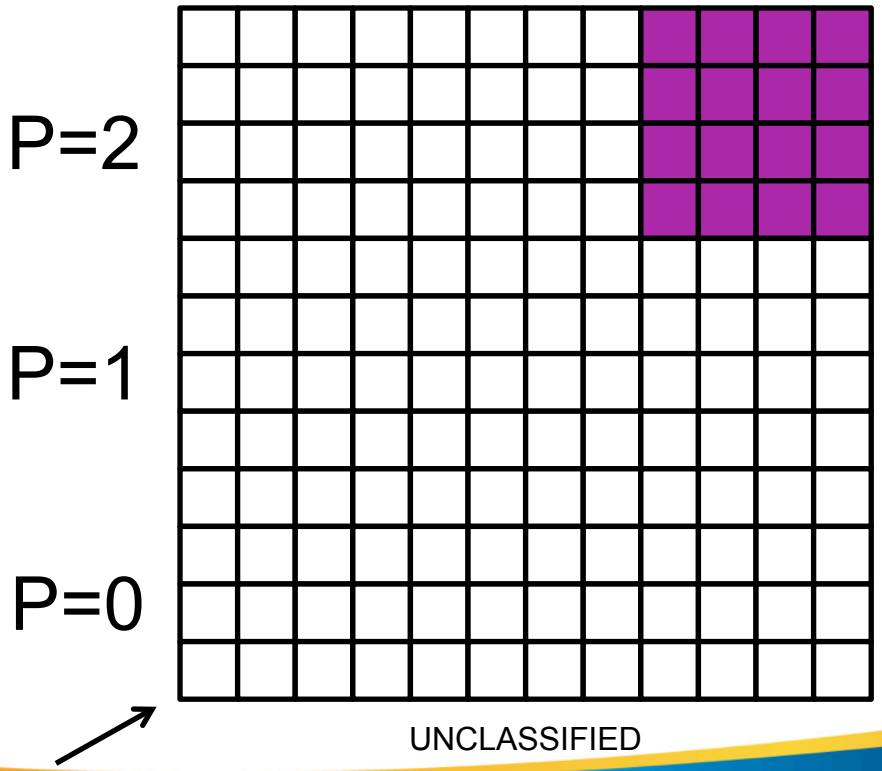
KBA mesh sweep



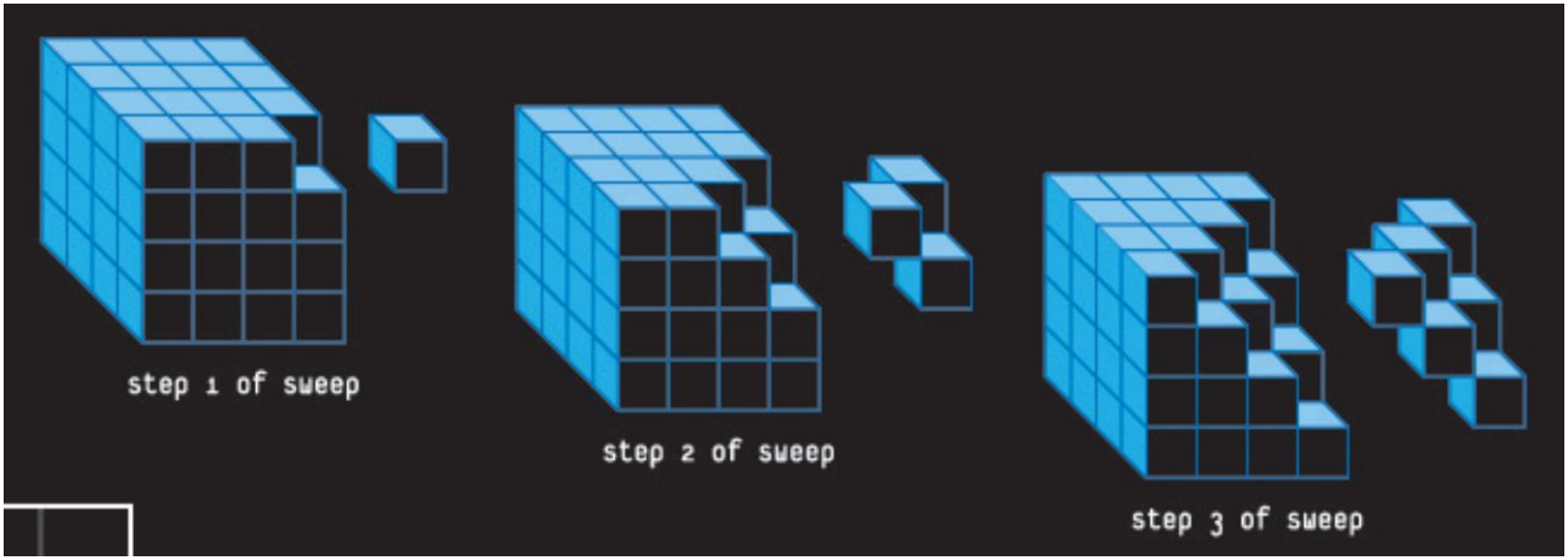
KBA mesh sweep



KBA mesh sweep



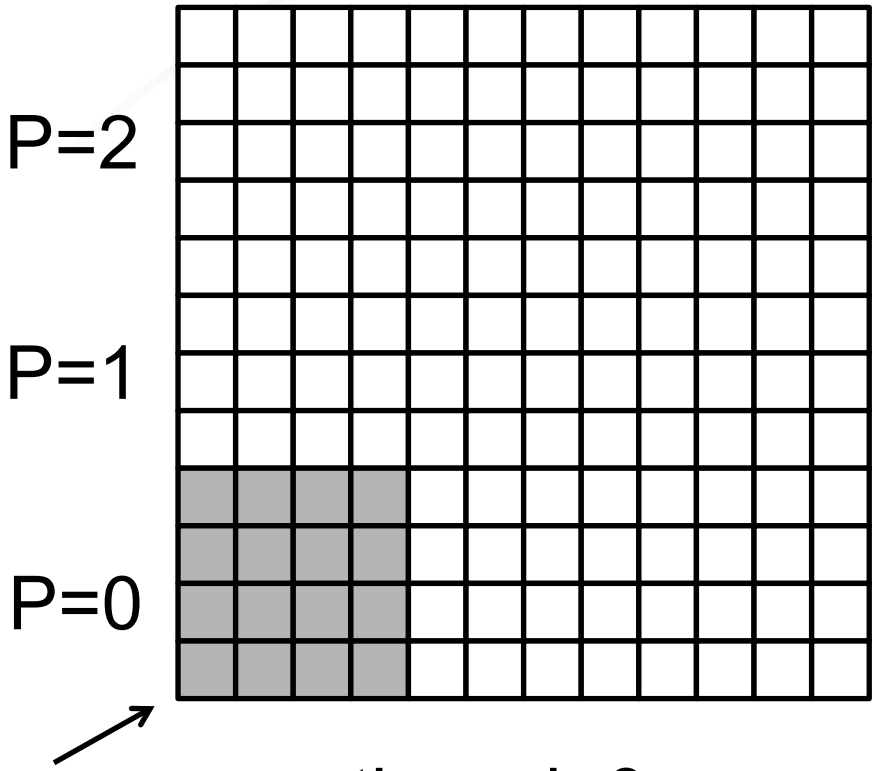
KBA mesh sweep in 3-D



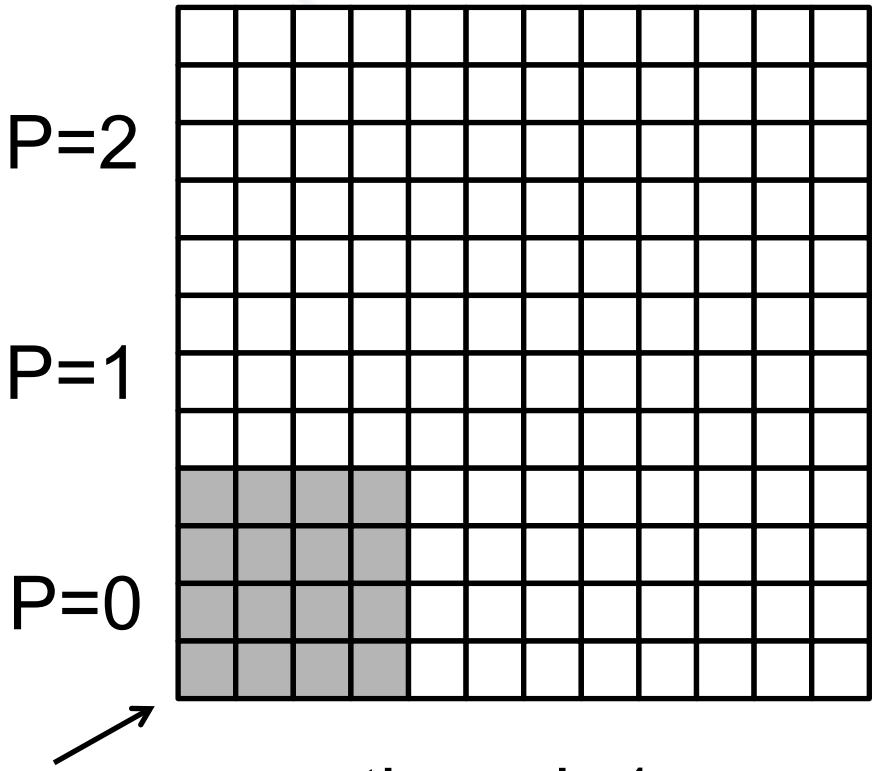
Threading

- Mesh sweeps done for each energy group independently between updates of the out-of-group source in the outer iterations (Jacobi iterations)
- Exploit shared memory at the NUMA node level
- Inner iteration group mesh sweeps performed concurrently via OpenMP threads

Energy threading



thread=0,
g=1

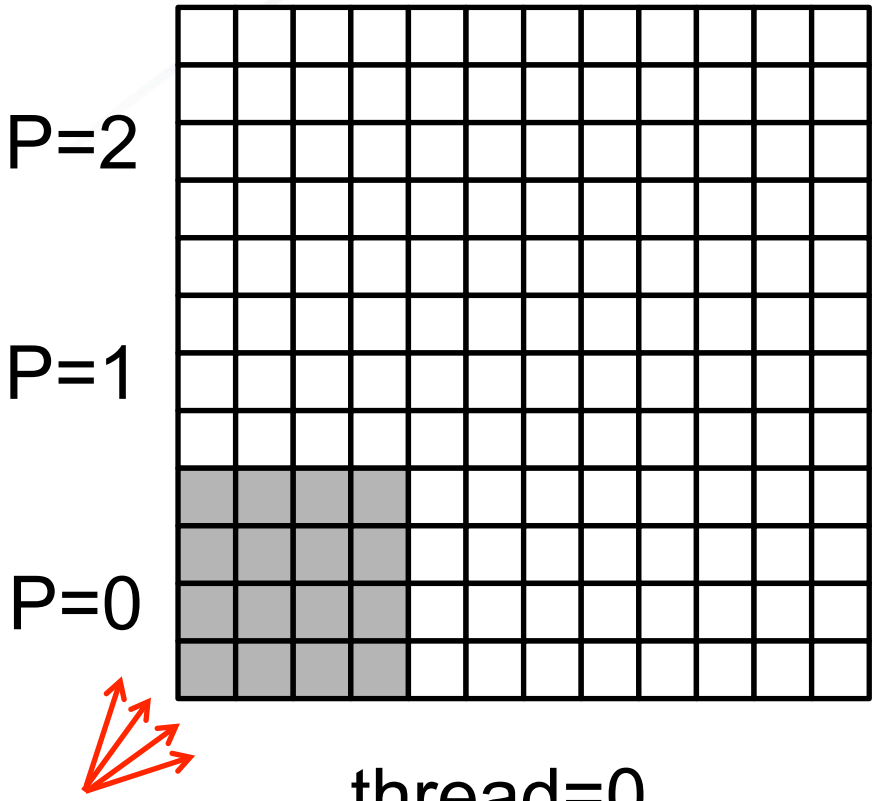


thread=1,
g=2

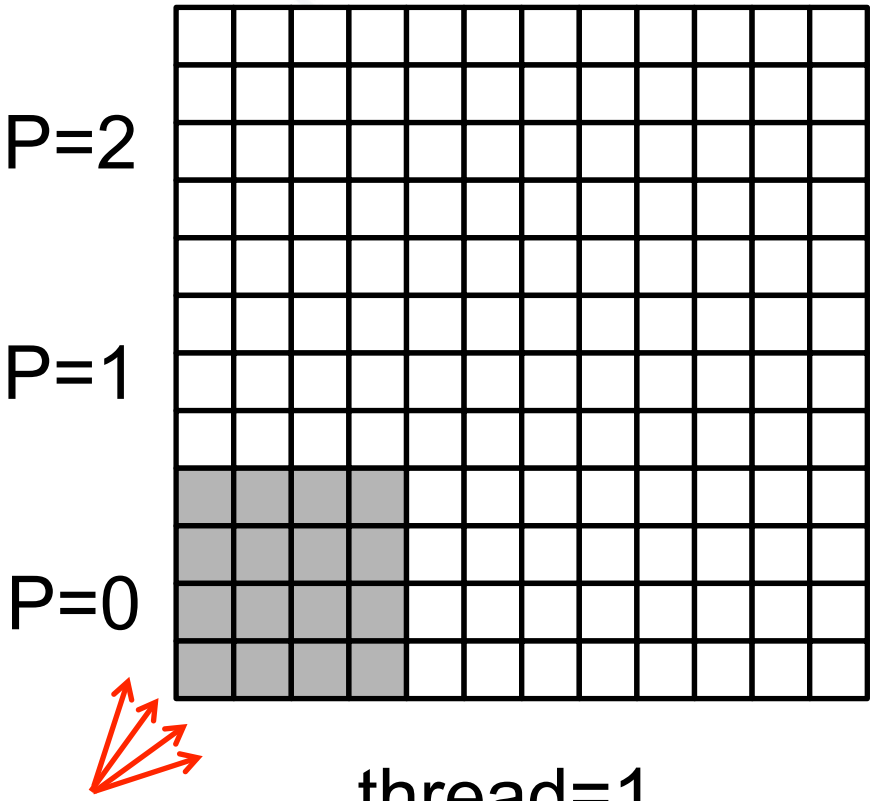
Vectorization

- Mesh sweeps for each angle
- Angles of given octant undergo same set of instructions in same order
- Exploit vectorization resources on chip by vectorizing operations over the angular dimension per octant

Angular vectorization



thread=0,
g=1

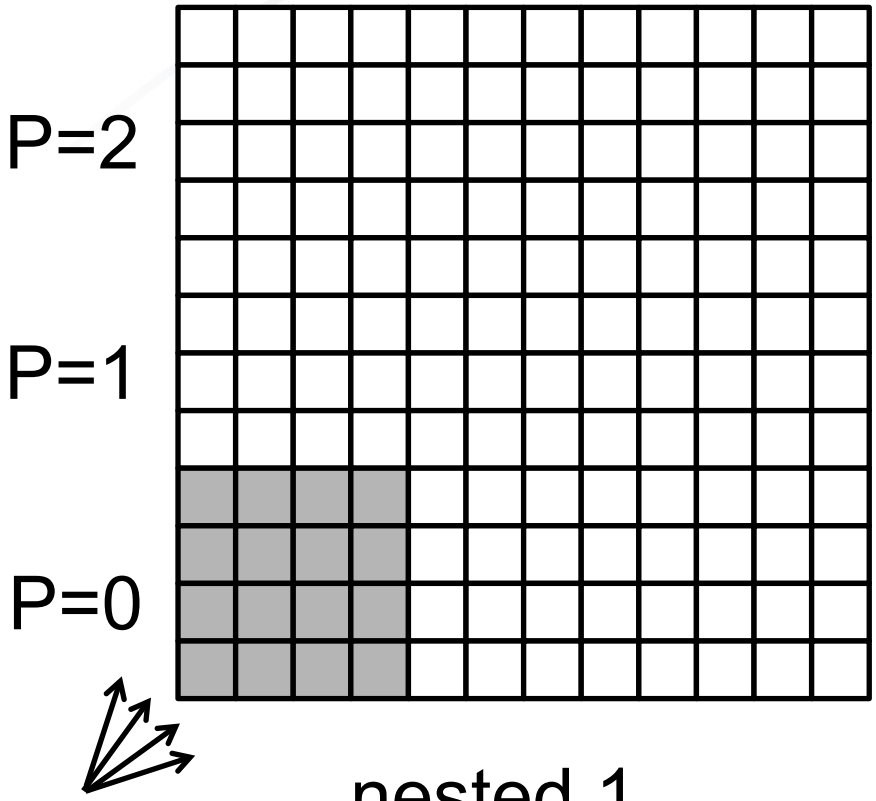


thread=1,
g=2

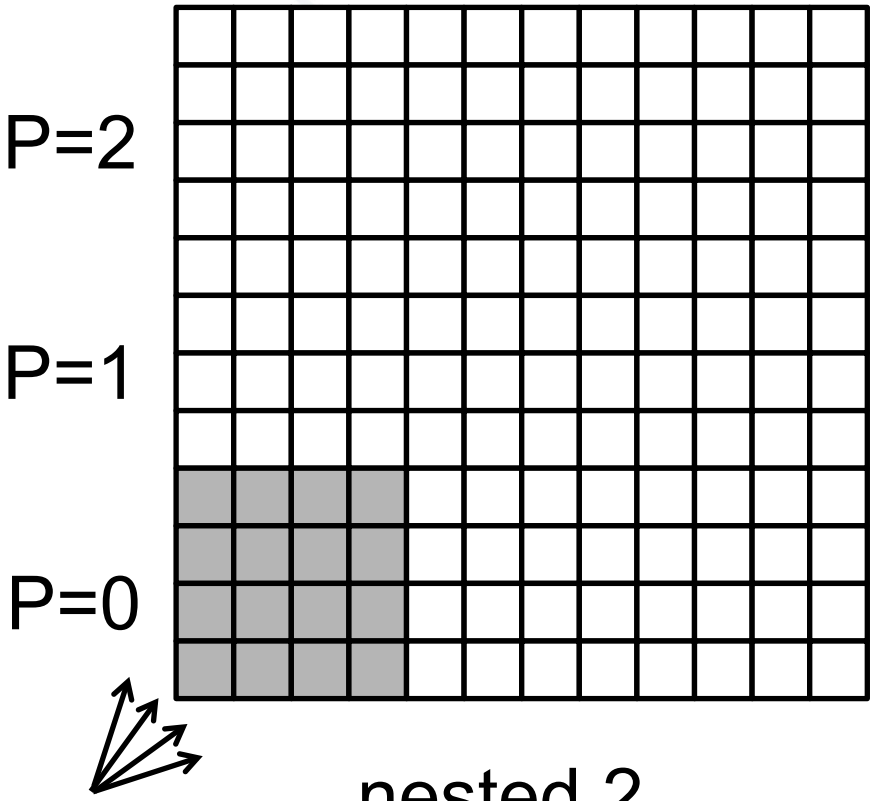
Nested threading

- Attempt to speed up the sweep operations for a single work chunk
- Options:
  - Additional threading for group sweeps
  - Angular work chunking
  - “mini-KBA” sweeps
  - *1/14/15 update:* concurrent octant mesh sweeps
  - *1/14/15 update:* task-based parallelism

Nested threading: additional group threads

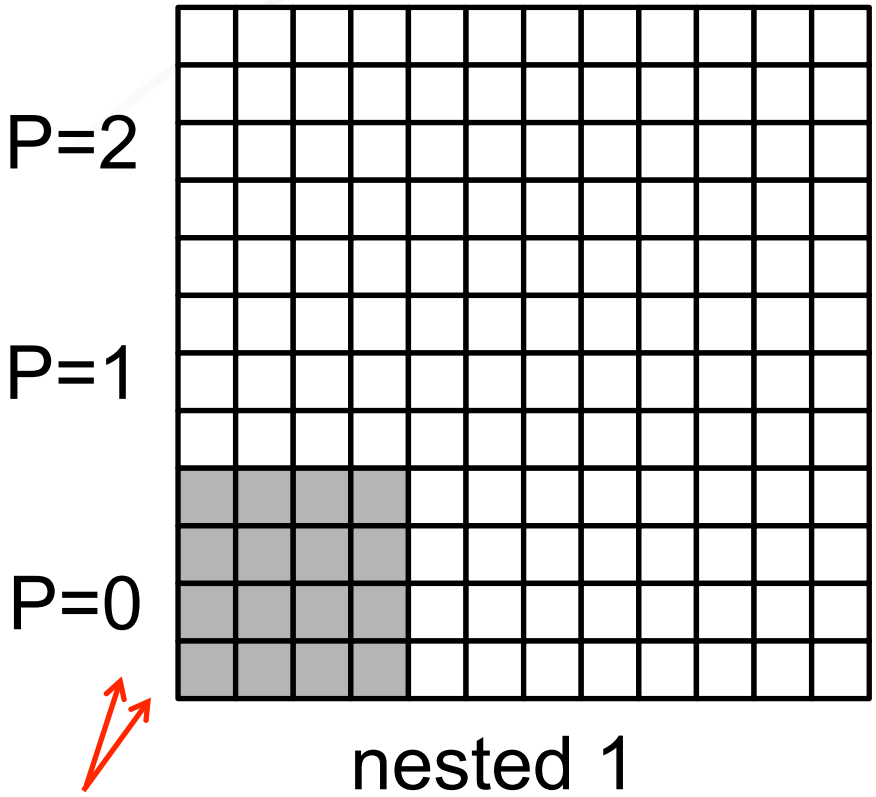


nested 1,
g=1

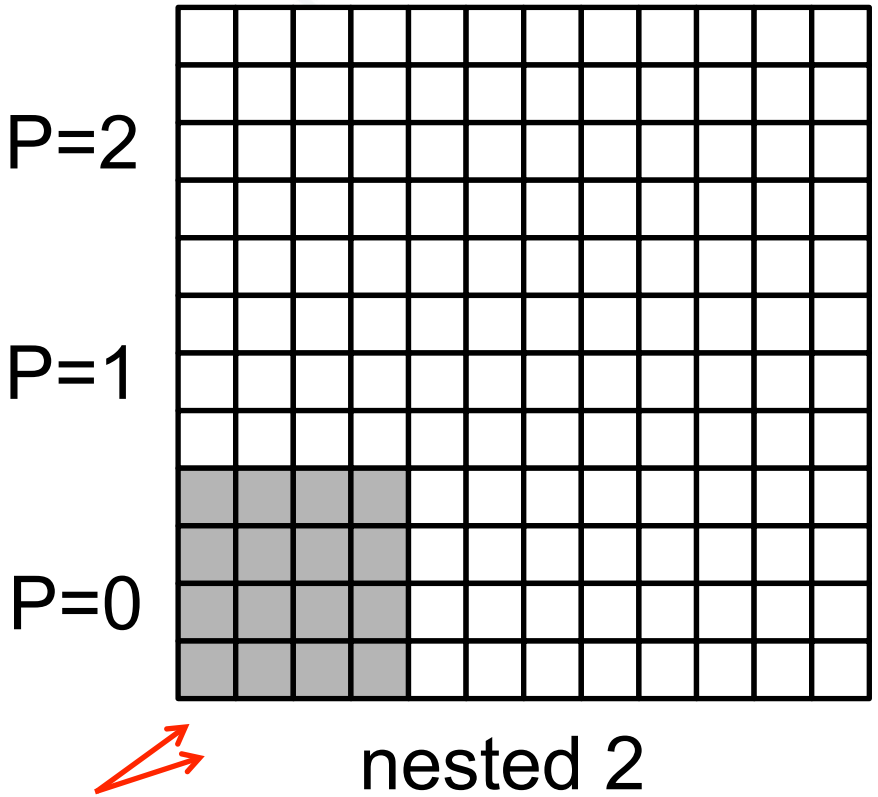


nested 2,
g=2

Nested threading: angular work chunks



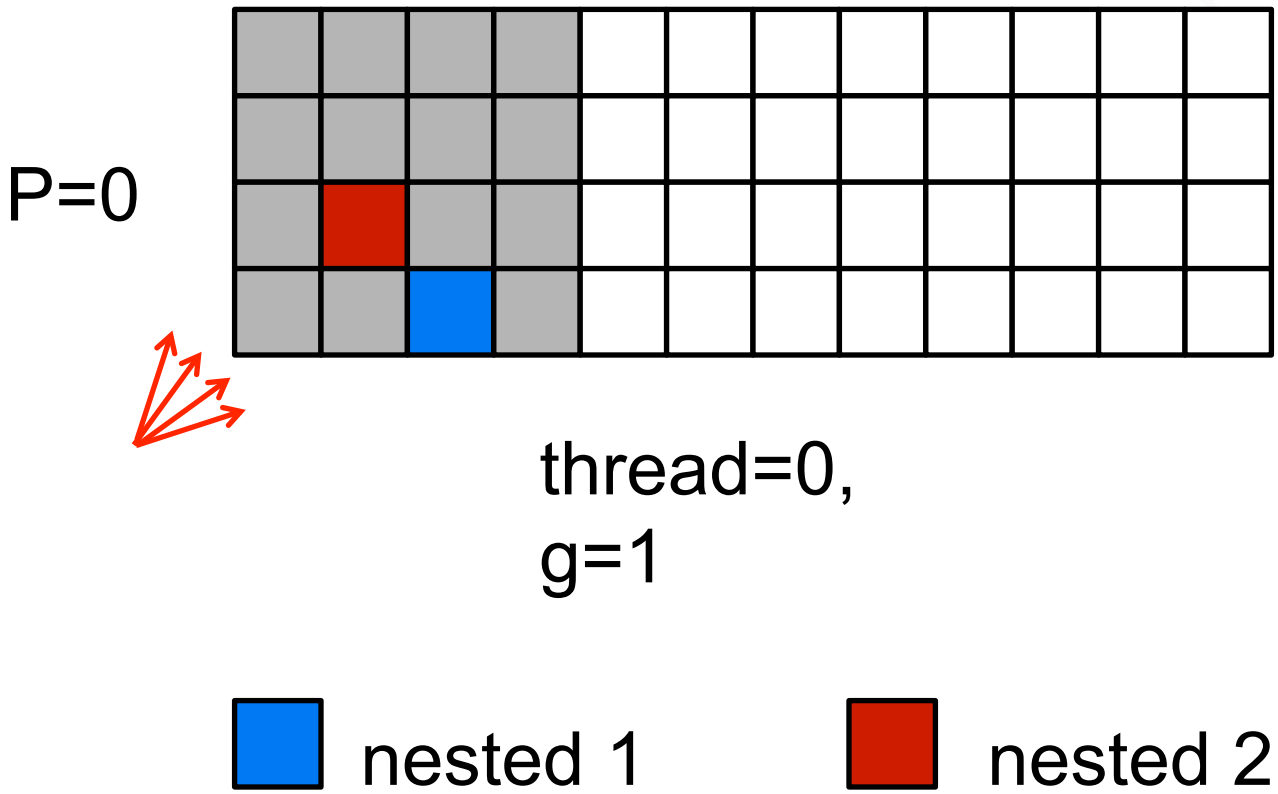
nested 1



nested 2

all angles in octant

Nested threading: mini-KBA



WHAT COMES NEXT?

Current State

- Re-synchronization with PARTISN
  - Apply nested threads to groups
  - Re-ordered source calculations
  - Better group→ thread assignment routines
  - Re-organized operations to better encapsulate the work of a single task
- Embedding previous ideas as options
  - Angular work chunks
  - Mini-KBA sweeps with nested threads

Future additions

- Concurrent octant mesh sweeps
  - Starts KBA sweeps for different octants simultaneously
  - Handle wavefront collisions
  - Incorporate threading
- Nested threading: how to do best?

Final thoughts

- Play with SNAP
  - Complicated problem
  - Lots of avenues for testing
  - Different parameters, knobs affect performance
- Ask questions
- Share with friends
  - Open source for absolute flexibility

Contact us

- SNAP Developers
  - Joe Zerr: rzerr@lanl.gov
  - Randy Baker: rsb@lanl.gov

QUESTIONS?

HOW CAN IT BE TESTED?

Testing considerations

- Typical problem SNAP is expected to see
  - Tens/Hundreds of groups, `ng`
  - Hundreds/Thousands of angles, `noct*nang`
  - Millions of spatial cells, `nx*ny*nz`
  - Time-dependent
- On a single NUMA node
  - Must have all energy/angle information
  - Have spatial sub-domain of full problem

Testing considerations

- Think about how much memory is available
  - Time-dependent: 2 discrete solution copies, *f*
  - Memory: `2*(nx*ny*nz)*ng*(nang*noct)`
    - `noct=2, 4, 8` → 1-D, 2-D, 3-D spatially
  - E.g., 8 GB per NUMA node
    - `ng=80`
    - `nang*noct=1200`
    - `nx*ny*nz=3000`
    - 576M doubles → ~4.3 GB

Scaling studies

- Angles – increase `nang`
- Groups – increase `ng` and/or `nthreads`
- Spatial mesh
  - Strong: create a problem with lots of cells that can fit on single process, then increase processes
    - Sacrifice `ng`, `nang`, or `timedep`
  - Weak: create a problem that fits on single process (NUMA node) and scale `nx*ny*nz`
- Fixup: expensive addition to sweeps

Caveats to testing

- SNAP is not real transport
  - Modifications made to operators and discretization (P_l, directions chosen)
  - Issues:
    - Convergence challenges
    - MMS: “Max Diff” between reference/computed
- BUT number of operations very similar to model PARTISN

Testing suggestions

- Create a regression test suite
  - Exercise all elements of the solution algorithm AND parallel model
  - Use base version of SNAP to generate reference solution
  - Compare modified versions of SNAP
    - Solution output
    - MMS differences
- Comfortable modified SNAP does not affect solution?
  - Perform scaling studies
  - Set solution aside, focus on timings

Those are merely suggestions…

- I can help…
  - Small test suite currently available from repo
    - Goes up to 16 tasks (task=MPI ranks * OpenMP threads)
  - Having trouble creating a test suite?
  - Want a larger testing suite?
  - Want suite specific to some study?

OPERATOR DETAILS

Anisotropic scattering

- When a particle scatters off a nucleus, different probability for each possible outgoing discrete direction
- Mathematically approximate scattering probability as a finite series of basis functions and coefficients

$$\mathbf{S}\left(\hat{\Omega}' \rightarrow \hat{\Omega}\right) \rightarrow \sigma_s\left(\hat{\Omega}' \rightarrow \hat{\Omega}\right) = \sum_{l=1}^{L} P_l\left(\hat{\Omega}' \cdot \hat{\Omega}\right) \sigma_{s,mat,l}$$

Group-to-group scattering

- When a particle scatters off a nucleus, different probability for each outgoing discrete speed
- Different properties for each material
- Neglect time-dependence

$$\mathbf{S}\left(mat,\hat{\Omega}'\rightarrow\hat{\Omega},E'\rightarrow E\right)\rightarrow\sigma_s\left(mat,\hat{\Omega}'\rightarrow\hat{\Omega},E'\rightarrow E\right)=\sum_{g'=1}^{G}\sum_{l=1}^{L}P_l\left(\hat{\Omega}'\cdot\hat{\Omega}\right)\widetilde{\sigma}_{s,mat,l,g'\rightarrow g}$$

Flux Transformations

- Typically interested in storing flux moments, F—sums of discrete fluxes, f, over all directions integrated with scattering basis functions P_l
- Use moments to compute group-to-group sources: $f \rightarrow F$
- Use discrete fluxes to compute source in particular direction during sweep: $F \rightarrow f$

HOW DOES ONE USE SNAP?

Building SNAP

- Use the Makefile
- Standard code (O3) → `make`
- Debugging code → `make OPT=no`
- Easily modifiable for different compilers
- Automatically includes MPI and OpenMP
- Utilities
  - `make count` (omits blank lines and comments)
  - `make clean`

Running SNAP

- Command line needs
  - MPI execution
  - Number of MPI processes/ranks
  - Number of threads AND resources available to threads
  - Path/Executable name
  - Input and Output file names

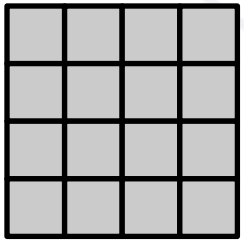
```
mpirun –cpus-per-proc # -np # [path]/snap inp out
```

Fixed data and SNAP limitations

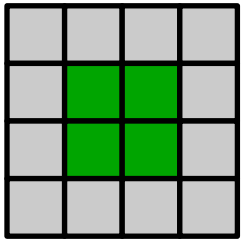
- SNAP data is setup with pre-fabricated algorithms not rooted in physical reality or mathematical consistency
- Diverges SNAP from a true transport code
- Data affected
  - Group speeds: v
  - Interaction probabilities: σ_t, σ_s
  - Angles, weights: Ω, w
  - Scattering expansion functions, P_l

mat_opt

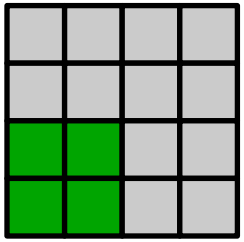
- `mat_opt=0`
  - Homogeneous



- `mat_opt=1`
  - Material 2 in center

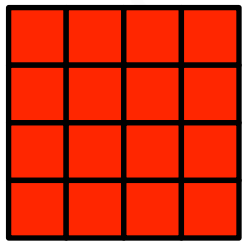


- `mat_opt=2`
  - Material 2 in corner

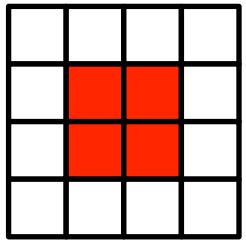


src_opt

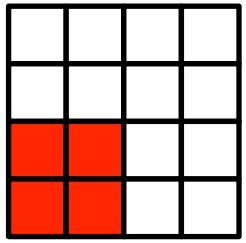
- Isotropic source, strength=1.0
- `src_opt=0`
  - Source everywhere



- `src_opt=1`
  - Center source



- `src_opt=2`
  - Corner source



Method of manufactured solutions

- Start with an analytic solution, *f**
- Plug into analytic equation → *q*
- Apply *q* to discretized system and solve for computed *f*
- Compare *f** with *f*
- In SNAP, `mms_opt=3`

$$f^* = tg\sin(ax)\sin(by)\sin(cz)$$

SNAP MMS

- Compute the reference solution (flux moments) over the spatial mesh
- Use where applicable to compute the manufactured inhomogeneous source
  - Time-independent component
  - Time-dependent component—modified each time step
  - Source is angularly dependent (not moments)
- Compare reference/computed solutions

Solution output

- Output file
  - Input echo
  - Setup echo
  - Iteration details (controlled by `it_det`)
  - Flux solution (controlled by `soloutp`)
  - MMS comparison (if available): Max, Min (Avg)
  - Timing summary
- "`flux`" file (controlled by `fluxp`)
- "`slgg`" file (controlled by `scatp`)

Timing summary

- Input timing
- Setup timing
- Solution timing
  - Transport sweeps – vast majority of solution time
  - Source calculation times
- Output time
- Grind time = solution time/(phase space size * iterations)
  - Measures how fast single unknown can be computed + penalties due to parallelism

ABSTRACT

- A new proxy application has been developed to model the performance of a modern discrete ordinates neutral particle transport application. It is modeled off the Los Alamos National Laboratory code PARTISN. PARTISN solves the linear Boltzmann transport equation (TE), a governing equation for determining the number of neutral particles (e.g., neutrons and gamma rays) in a multi-dimensional phase space. SNAP itself is not a particle transport application; SNAP incorporates no actual physics in its available data, nor does it use numerical operators specifically designed for particle transport. Rather, SNAP mimics the computational workload, memory requirements, and communication patterns of PARTISN. It features spatial decomposition via MPI, threading over energy groups via OpenMP, and vectorization over angles.