SNAP: SN (Discrete Ordinates) Application Proxy, Version 1.10

User's Manual

Joseph Zerr (rzerr@lanl.gov) and Randal Baker (rsb@lanl.gov)
Los Alamos National Laboratory
Computational Physics and Methods, CCS-2
Updated: 6 April 2020

1. Introduction

SNAP is a proxy application to model the performance of a modern discrete ordinates neutral particle transport application. SNAP may be considered an update to Sweep3D [1], intended for hybrid computing architectures. 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. [2] 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. The equation it solves has been composed to use the same number of operations, use the same data layout, and load elements of the arrays in approximately the same order. Although the equation SNAP solves looks similar to the TE, it has no real world relevance.

The solution to the time-dependent TE is a "flux" function of seven independent variables: three spatial (3-D spatial mesh), two angular (set of discrete ordinates, directions in which particles travel), one energy (particle speeds binned into "groups"), and one temporal. PARTISN, and therefore SNAP, uses domain decomposition over these dimensions to coherently distribute the data and the tasks associated with solving the equation. The parallelization strategy is expected to be the most efficient compromise between computing resources and the iterative strategy necessary to converge the flux.

The iterative strategy is comprised of a set of two nested loops. These nested loops are performed for each step of a time-dependent calculation, wherein any particular time step requires information from the preceding one. No parallelization is performed over the temporal domain. However, for time-dependent calculations two copies of the unknown flux must be stored, each copy an array of the six remaining dimensions. The outer iterative loop involves solving for the flux over the energy domain with updated information about coupling among the energy groups. Typical calculations require tens to hundreds of groups, making the energy domain suitable for threading with the node's (or nodes') provided accelerator. [3] The inner loop involves sweeping across the entire spatial mesh along each discrete direction of the angular domain. The spatial mesh may be immensely large. Therefore, SNAP spatially decomposes the problem across nodes and communicates needed information according to the KBA method. [4] KBA is a transport-specific application of general parallel wavefront methods. Nested threads, spawned by energy group threads, are available to use in one of two ways. Per one approach, nested threads may be used to

further parallelize the work to sweep different energy groups assigned to a main-level thread. This option is still experimental and has only been implemented to work in the case of using a single MPI process. Alternatively, nested threads are used to perform "mini KBA" sweeps by concurrently operating on cells lying on the same diagonal of spatial sub-domains already decomposed across the distributed memory architecture (i.e., different MPI ranks). Lastly, although KBA efficiency is improved by pipelining operations according to the angle, current chipsets operate best with vectorized operations. During a mesh sweep, SNAP operations are vectorized over angles to take advantage of the modern hardware.

SNAP is written to the Fortran 90/95 standard primarily. The retrieval of command line arguments, which contain file names, is handled with a standard Fortran 2003 intrinsic subroutine. Modules are used to provide explicit interfacing among the different procedures. Distributed memory communications are performed using MPI commands, and threading is achieved with OpenMP directives.

2. Input

SNAP uses a single Fortran namelist for its input: `invar`. All variables are optional and default parameters are provided at compilation. An input check is performed to ensure logical consistency among the inputs and to ensure the variables are set to permissible values. Table 1 lists the available input parameters, a brief description of the variable, their range of acceptable values (absent consideration of logical consistency and other rules applied for simplifying SNAP coding), and the default value. A sample input also is provided.

Table 1. List of available SNAP inputs.

Variable	Description	Acceptable Values	Default
`npey`	number of *y*-processes	# ≥ 1	1
`npez`	number of *z*-processes	# ≥ 1	1
`ichunk`	number of *x*-planes for single work chunk	1 ≤ # ≤ nx	4
`nthreads`	number of parallel threads per MPI rank	# ≥ 1	1
`nnested`	number of nested "mini-KBA" threads	# ≥ 0	0
`ndimen`	number of spatial dimensions	1–3	1
`nx`	number of uniformly-sized cells in *x*-direction	# ≥ 4	4
`lx`	length of x-dimension	# > 0.0	1.0
`ny`	number of uniformly-sized cells in *y*-direction, `ny=1` if 1-D	# ≥ 4	1
`ly`	length of *y*-dimension, `ly=0.0` if 1-D	# > 0.0	0.0
`nz`	number of uniformly-sized cells in *z*-direction, `nz=1` if 1-D or 2-D	# ≥ 4	1
`lz`	length of *z*-dimension, `lz=0.0` if 1-D or 2-D	# > 0.0	0.0
`nmom`	order of the scattering expansion	1 ≤ # ≤ 4	1

Variable	Description	Acceptable Values	Default
nang	number of discrete ordinates per octant	# ≥ 1	1
ng	number of energy groups	# ≥ 1	1
epsi	convergence criterion	0.0 < # < 1.0E-2	1.0E-04
iitm	number of inner iterations per energy group per outer	# ≥ 1	5
oitm	number of outer iterations per time step	# ≥ 1	100
timedep	no/yes time-dependent calculation	0/1	0
tf	total simulation time, 0.0 if timedep=0	# ≥ 0.0	0.0
nsteps	number of uniformly-sized time steps	# ≥ 1	1
mat_opt	material layout flag – homogeneous/center/corner	0/1/2	0
src_opt	source layout flag – everywhere/center/corner/MMS	0/1/2/3	0
scatp	no/yes print scattering matrix to file	0/1	0
it_det	no/yes print full iteration details	0/1	0
fluxp	print flux moments to file flag – no/scalar/all	0/1/2	0
fixup	no/yes perform negative flux fixup	0/1	0
soloutp	no/yes print single k-plane solution to output file	0/1	0
kplane	print specified k-plane to print with soloutp – 0 = default mid-plane, 1-nz = specified plane	0/1+	0
popout	no/final cycle only/all cycles print population data to output file	0/1/2	0
swp_typ	standard order/mini-KBA sweep per spatial work chunk	0/1	0
angcpy	store one or two copies of the time-edge angular flux; single copy requires additional operations	1/2	1
multiswp	no/yes concurrent octant mesh sweeps	0/1	1

2.1 Sample SNAP Input

```
! Input from namelist
&invar
 npey=2
 npez=2
 ichunk=2
 nthreads=2
 nnested=1
 ndimen=3
 nx=6
 lx=0.6
 ny=6
 ly=0.6
 nz=6
 lz=0.6
 nmom=1
 nang=10
 ng=4
 epsi=1.0E-4
 iitm=5
 oitm=30
 timedep=0
 tf=1.0
 nsteps=1
 mat_opt=0
 src_opt=0
 scatp=0
 it_det=0
 fluxp=0
 fixup=1
 soloutp=1
 kplane=0
 popout=0
 swp_typ=0
 angcpy=1
 multiswp=1
/
```

SNAP will process the given input through a series of checks that enforce logical consistency and other pre-determined rules. Some input errors may trigger program termination, while others will simply give a warning and reset the offending value. In either case, the SNAP output will inform the user of problems encountered during the input check.

2.2 Command Line Instruction for Running SNAP

For a standard implementation of MPI, SNAP can be run with the mpirun command in the following way:

```
mpirun [-cpus-per-proc nthreads] -np npey×npez path_to/executable infile outfile
```

For example, assuming the executable *snap* is placed in the working directory (with the input file *inp*), the above instruction would be specifically translated to run the code and produce the output file, *out*:

```
mpirun -cpus-per-proc 2 -np 4 ./snap inp out
```

3. Code Structure

SNAP has been developed with Fortran 90/95 modular programming techniques. Starting with the main subroutine, SNAP establishes the MPI environment. It then calls for input, setup, solution, and output in that order.

SNAP is comprised of Fortran modules, each containing data and/or subroutines built for specific tasks. The following is a list of the modules and standalone subroutines in logical order to their first use. Each module is briefly described by the data contained and the functionality of the subroutines also contained. A complete flowchart for SNAP is provided in Section 10.

- `snap_main`: main program; no data; controls overall flow of the program, including input, setup, solution, output
- `global_module`: global data and variable kind type declarations/definitions
- `utils_module`: input/output file control, error handling, program termination
- `plib_module`: parallel environment control data; wrapper containing subroutines for point-to-point and collective communications, MPI initialization/finalization, and communicator setup
- `version_module`: version data; subroutine for printing version information
- `time_module`: program execution timing data; subroutine for printing timing information
- `input_module`: subroutines for reading input, echoing input, checking input for logical consistency and preselected input bounds
- `geom_module`: spatial geometry data; runtime array allocation/deallocation; solution data setup
- `sn_module`: discrete ordinates, angular data; runtime array allocation/deallocation
- `data_module`: miscellaneous problem data; runtime array allocation/deallocation
- `control_module`: problem execution/iteration control data; runtime array allocation/deallocation
- `setup_module`: setup runtime data after allocation; establish material map, source distribution

- `mms_module`: method of manufactured solutions data; compute manufactured source; compare computed/manufactured solutions
- `translv` (subroutine): control the iterative solution process; call for solution array allocation, data setup; control outermost loop (temporal) of solution method
- `solvar_module`: runtime solution array data; allocation/deallocation of arrays
- `expxs_module`: expand data packed by material to a larger array sized by the spatial grid
- `outer_module`: control outer iterative loop; compute iteration-dependent sources; call for inner iterative loop; convergence checking
- `inner_module`: control inner iterative loop; compute iteration-dependent sources; call for spatial mesh sweep; convergence checking
- `sweep_module`: control spatial mesh sweep
- `thrd_comm_module`: establish task lists for threaded operations; handle MPI communications from spatial mesh sweep
- `octsweep_module`: call for appropriate sweep kernel depending on spatial dimensionality and time-dependence flag
- `dim1_sweep_module`: transport-like kernel for 1-D slab problems
- `dim3_sweep_module`: transport-like kernel for 2-D and 3-D Cartesian problems
- `mkba_sweep_module`: transport-like kernel for 2-D and 3-D Cartesian problems using mini-KBA sweeps over the spatial work chunks
- `output_module`: print information to output file and optional flux file; call for verification of computed source if manufactured solution option has been selected
- `dealloc_module`: call deallocation routines contained in other modules
- `analyze_module`: contains subroutines for additional data analysis/editing

4. Modules with Data of Non-Local Scope

The following modules are a list of those that contain data that takes on non-local scope. For each item, a brief description of their variable contents is also listed.

- `global_module`: numeric kinds, file names and unit numbers, oft-used floating numbers
- `version_module`: version information, including number and date
- `plib_module`: parallel processing control variables
- `geom_module`: geometry input and solution variables
- `sn_module`: discrete ordinates input and solution variables
- `data_module`: pseudo cross section/nuclear data/group data input and solution variables
- `control_module`: time-dependence and convergence criterion input and solution variables
- `solvar_module`: variables necessary for solution, including flux and group source arrays
- `time_module`: variables for execution time measurements

- `mms_module`: variables storing the reference manufactured solution flux moments

5. Numerical Equation Solved

SNAP solves a numerical equation that mimics the TE. Specifically, SNAP begins with the analytic equation of the following 1-D form,

$$\begin{aligned}\frac{1}{v_g}\frac{\partial}{\partial t}f_{n,g}(\vec{r},t) + \widehat{\Omega}_n \cdot \vec{\nabla} f_{n,g} + \sigma_{t,g}(\vec{r}) f_{n,g} \\ = q_{n,g}(\vec{r},t) \\ + \sum_{g'=1}^{ng}\left[\sigma_{s,1,g'\to g}(\vec{r})\sum_{n'=1}^{N} w_{n'} f_{n',g'} + \sum_{l=2}^{nmom}\tilde{P}_l(\widehat{\Omega}_n)\sigma_{s,l,g'\to g}(\vec{r})\sum_{n'=1}^{N} w_{n'}\tilde{P}_l(\widehat{\Omega}_{n'}) f_{n',g'}\right].\end{aligned} \tag{1}$$

In Eq. (1), n is an angular index—i.e., Eq. (1) is solved for discrete directions that sum according to preset quadrature rules. A problem is solved along $N = nang \times noct$, where $noct = 2,\ 4,\ 8$ for 1-D, 2-D, 3-D problems, respectively. g is an energy group index, $1 \le g \le ng$, where the energy group is a range of energies over which particle populations are summed. $\vec{r}$ is the spatial location, and t refers to the time for a time-dependent problem.

Other variables in Eq. 1 are defined as follows:

- $f_{n,g}(\vec{r},t)$: pseudo-flux, unknown SNAP solves
- $\widehat{\Omega}_n = \mu_n\hat{\imath} + \eta_n\hat{\jmath} + \xi_n\hat{k}$: unit vector discrete direction, sum of directional cosines
- v_g: group g speed
- $\sigma_{t,g}(\vec{r})$: total particle-medium interaction probability
- $\sigma_{s,l,g'\to g}(\vec{r})$: particle-medium scattering probability for moment l, from g' to g
- $q_{n,g}(\vec{r},t)$: inhomogeneous source
- w_n: discrete directions' weights in the angular quadrature
- $\tilde{P}_l$: scattering expansion basis function

Note that the weights and the angles are defined by the quadrature, which is hardcoded in SNAP, depending on the number of angles per octant, selected by the user via `nang`.

The 1-D expansion function is

$$\tilde{P}_l(\widehat{\Omega}_n) = \begin{cases} 1, & l = 1 \\ (\mu_n)^{2l-3}, & l > 1. \end{cases} \tag{2}$$

The $l = 1$ case is the special case of no directional dependence, i.e., "isotropic," and therefore has an expansion function of unity.

To correctly simulate the number of operations, the multi-dimensional cases require modifications to the scattering expansion function. For the 2-D case, the scattering term—second right hand side (RHS) term of Eq. (1)—is rewritten as

$$\sum_{g'=1}^{ng}\left[\sigma_{s,1,g'\to g}(\vec{r})\sum_{n'=1}^{N}w_{n'}f_{n',g'}+\sum_{l=2}^{nmom}\sum_{m=1}^{l}\tilde{P}_{lm}(\widehat{\Omega}_n)\sigma_{s,l,g'\to g}(\vec{r})\sum_{n'=1}^{N}w_{n'}\tilde{P}_{lm}(\widehat{\Omega}_{n'})f_{n',g'}\right]. \quad (3)$$

Note that the scattering expansion basis function now has two indices and two sums are required. This is to account for the added dimensionality of the problem. However, the scattering cross section σ_s still varies with l only. The 2-D expansion function is

$$\tilde{P}_{lm}(\widehat{\Omega}_n)=\begin{cases}1, & l=1,m=1\\(\mu_n)^{2l-3}(\eta_n)^{m-1}, & l>1,1\le m\le l.\end{cases} \quad (4)$$

Lastly, for 3-D calculations, Eq. (3) is modified in the number of values for m:

$$\sum_{g'=1}^{ng}\left[\sigma_{s,1,g'\to g}(\vec{r})\sum_{n'=1}^{N}w_{n'}f_{n',g'}+\sum_{l=2}^{nmom}\sum_{m=1}^{2l-1}\tilde{P}_{lm}(\widehat{\Omega}_n)\sigma_{s,l,g'\to g}(\vec{r})\sum_{n'=1}^{N}w_{n'}\tilde{P}_{lm}(\widehat{\Omega}_{n'})f_{n',g'}\right]. \quad (5)$$

Finally, the 3-D expansion function is similar to that of Eq. (4), but includes the z-direction cosine and is valid for the larger m-range:

$$\tilde{P}_{lm}(\widehat{\Omega}_n)=\begin{cases}1, & l=1,m=1\\(\mu_n)^{2l-3}(\eta_n\xi_n)^{m-1}, & l>1,1\le m\le 2l-1.\end{cases} \quad (6)$$

The remainder of SNAP's documentation will rely on the 3-D case.

The expansion function is used to determine angular moments of the function $f_{m,g}$,

$$F_{g,l,m}=\sum_{n=1}^{N}w_n\tilde{P}_{lm}(\widehat{\Omega}_n)\,f_{n,g}. \quad (7)$$

In multi-dimensional systems, the $l=1,m=1$ case is isotropic. All other cases are considered anisotropic—the angular moment has some dependence on the direction of particle travel.

The analytic expression of Eq. (1) and its multi-dimensional analogs undergo numerical treatment. Namely, the time-derivative is approximated with Crank-Nicholson differencing, and the spatial relationship is treated by spatially integrating Eq. (1) (or the analogs) and then applying a closure relation. Eq. (1) is modified for the 3-D (*x*-*y*-*z*) case by substituting Eqs. (5) and (6) into the second RHS term (i.e., the scattering source). The numerical equation then solved by SNAP is given by

$$
\begin{aligned}
&\frac{1}{v_g \Delta t^h}\left(f^{h+1/2}_{n,i,j,k,g} - f^{h-1/2}_{n,i,j,k,g}\right) + \frac{\mu_n}{\Delta x_i}\left(f^{h}_{n,i+1/2,j,k,g} - f^{h}_{n,i-1/2,j,k,g}\right) \\
&\quad + \frac{\eta_n}{\Delta y_j}\left(f^{h}_{n,i,j+1/2,k,g} - f^{h}_{n,i,j-1/2,k,g}\right) + \frac{\xi_n}{\Delta z_k}\left(f^{h}_{n,i,j,k+1/2,g} - f^{h}_{n,i,j,k-1/2,g}\right) \\
&\quad + \sigma_{t,i,j,k,g} f^{h}_{n,i,j,k,g} \\
&\quad = q^{h}_{n,i,j,k,g} \\
&\quad + \sum_{g'=1}^{ng} \left[\sigma_{s,1,i,j,k,g' \to g} \sum_{n'=1}^{N} w_{n'} f^{h}_{n',i,j,k,g'} \right. \\
&\quad \left. + \sum_{l=2}^{nmom} \sum_{m=1}^{2l-1} (\mu_n)^{2l-1} (\eta_n \xi_n)^{m-1} \sigma_{s,l,i,j,k,g' \to g} \sum_{n'=1}^{N} w_{n'} (\mu_{n'})^{2l-1} (\eta_{n'} \xi_{n'})^{m-1} f^{h}_{n',i,j,k,g'} \right],
\end{aligned}
\tag{8}
$$

with the closure relations,

$$
\begin{aligned}
2f^{h}_{n,i,j,k,g} &= \left(f^{h+1/2}_{n,i,j,k,g} + f^{h-1/2}_{n,i,j,k,g}\right) = \left(f^{h}_{n,i+1/2,j,k,g} + f^{h}_{n,i-1/2,j,k,g}\right) \\
&= \left(f^{h}_{n,i,j+1/2,k,g} + f^{h}_{n,i,j-1/2,k,g}\right) = \left(f^{h}_{n,i,j+1/2,k,g} + f^{h}_{n,i,j-1/2,k,g}\right).
\end{aligned}
\tag{9}
$$

Equation (9) closes Eq. (8) by assuming the flux shape in a space-time cell is linear between any opposing edges. Eq. (9) is substituted into Eq. (8) and solved for each discrete direction via iterative mesh sweeps previously mentioned in Section 1.

5.1 Negative Flux Fixup

SNAP spatial and temporal discretizations can be shown to be second-order accurate. A consequence of these discretizations is the possibility that outgoing edge fluxes are computed to be negative values according to Eq. (9). Negative flux values are unphysical and often avoided to maintain solution accuracy.

SNAP is equipped with a costly, non-linear fixup routine that is applied at each spatial cell (for a given time step and group) across all angles of the octant being swept. The fixup algorithm involves checking for a negative edge flux first. If one exists, the outgoing edge flux is set to zero. Then Eq. (8) is resolved with the known (zero) value and without the appropriate closure relation according to Eq. (9). The fixup algorithm is multi-pass; after the center flux value is recomputed any non-fixed-up outgoing edges are again checked for negativities. This loop continues until no more fixup operations are necessary for each angle being swept.

6. Hardwired Data Descriptions

Within the `setup_module` of SNAP, several subroutines are called to assign values to the factors in Eq. (8). The user has flexibility in setting some problem parameters, as described in Section 2. With that input, SNAP has been programmed with preset rules to assign data values. These rules are not derived from any physical properties; rather they are simple constructs that provide SNAP with proper array sizes to mimic PARTISN operations.

Note that Eq. (8) is independent of units. SNAP data is unitless, and the values presented below are done so absent any relation to physical measures.

6.1 Phase-space cell sizes

The phase space of the analytic domain is discretized to yield the numerical equation given by Eq. (8). In Eq. (8), as many as four phase-space cell sizes are required, three spatial cell sizes and one time step size: $\Delta x_i, \Delta y_j, \Delta z_k, \Delta t^h$. In SNAP, the mesh spacing is uniform for all dimensions. These factors are computed with the input variables `lx`, `nx`, `ly`, `ny`, `lz`, `nz`, `tf`, `nsteps`. For example,

$$\Delta x_i = \Delta x \equiv lx/nx. \tag{10}$$

Δy and Δz are analogously defined. Moreover,

$$\Delta t^h = \Delta t \equiv tf/nsteps. \tag{11}$$

6.2 Group speeds

The group speeds v_g are stored in an array of `ng` elements. The values are set according to the value of `ng`, simply using reverse ordering of the group indices as the speed. For group g,

$$v_g = ng - g + 1. \tag{12}$$

For a simple one-group problem, $v_1 = 1$. For a ten-group problem $v_1 = 10$, $v_2 = 9$, and so on, up to $v_{10} = 1$.

6.3 Angular quadrature

The angular quadrature is defined as a set of discrete ordinates (or "angles") and corresponding weights. The information is used to determine streaming terms, scattering sources, and leakages. With the user inputs `nang` and `ndimen`, SNAP creates an angular quadrature. `nang` informs SNAP how many ordinates are to be used per "octant." The number of "octants" is determined by the spatial dimensionality of a problem: 2 (left-right half-slabs) for `ndimen=1` (1-D), 4 (quadrants) for `ndimen=2`, 8 (actual octants) for `ndimen=3` (3-D). The quadrature is set only for the all positive, principal "octant," and angles for all other "octants" are known by enforced full symmetry of the set.

The discrete ordinates are unit vectors. Each ordinate is given by three directional cosines, μ_n, η_n, ξ_n:

$$\widehat{\Omega}_n = \mu_n \hat{\imath} + \eta_n \hat{\jmath} + \xi_n \hat{k}. \tag{13}$$

As the name implies, a directional cosine is the cosine of the geometric angle between the ordinate and the positive axis of any of the spatial dimensions. Because the discrete ordinate is a unit vector, the directional cosines must satisfy

$$\mu_n^2 + \eta_n^2 + \xi_n^2 = 1. \tag{14}$$

These directional cosines are stored as the required angular quadrature information. Each directional cosine is stored in an array of size `nang`. To create the angular quadrature, SNAP first determines some angular spacing between the μ_n values. Using `nang`, SNAP sets

$$\Delta n \equiv 1/nang. \tag{15}$$

With this, SNAP can compute the μ_n values according to

$$\mu_n = \frac{1}{2}\Delta n + (n-1)\Delta n, \;\; 1 \le n \le nang. \tag{16}$$

The *y* directional cosines are similarly spaced, but in reverse order,

$$\eta_n = 1 - \frac{1}{2}\Delta n - (n-1)\Delta n, \;\; 1 \le n \le nang. \tag{17}$$

Lastly, the *z* directional cosines are used by applying Eq. (14). Explicitly,

$$\xi_n = \sqrt{1 - \mu_n^2 - \eta_n^2}. \tag{18}$$

This algorithm for setting μ and η values can be easily shown to produce only real (i.e., non-complex) corresponding ξ values.

To complete the quadrature set, SNAP creates an array of angular weights, length `nang`. Although the weights could vary, for simplicity SNAP sets each weight as

$$w_n = w \equiv \frac{1}{nang \times noct}, \tag{19}$$

Where again $noct$ is determined by the level of spatial dimensionality: 2, 4, or 8 for 1-D, 2-D, or 3-D, respectively.

6.4 Material Layout

The material layout in SNAP is determined by the input variable `mat_opt`. Setting `mat_opt=0` instructs SNAP to use a single material (Material 1) everywhere (i.e., homogeneous problem). When `mat_opt=1`, SNAP creates two materials and places Material 2 in the middle of the problem. For every spatial dimension, SNAP determines the range of cells that will be assigned Material 2. Mathematically, this range is a closed set,

$$\left[\left\lfloor\frac{n}{4}\right\rfloor + 1, \left\lfloor\frac{3n}{4}\right\rfloor\right], \tag{20}$$

where n is the number of cells in the x, y, or z direction, and $\lfloor x \rfloor$ is the floor function, which is equivalent to integer division.

When `mat_opt=2`, a similar procedure is applied, by now the slab, square, cube region of Material 2 is positioned in the left half, bottom-left corner, or front-bottom-left corner of the 1-D, 2-D, 3-D domain, respectively. That is, the range analogous to Eq. (20) for `mat_opt=2` is

$$\left[1, \left\lfloor\frac{n}{2}\right\rfloor\right]. \tag{21}$$

SNAP output will inform the user of the range of cells that are assigned Material 2. All other cells are Material 1.

6.5 Material Data

The material data needed by Eq. (8)—$\sigma_{t,i,j,k,g}$ and $\sigma_{s,l,i,j,k,g'\to g}$—is set by SNAP with some flexibility by the user. The previous section discussed how the user can choose to use one or two materials and set their layout on the $I \times J \times K$ ($nx \times ny \times nz$) grid. Moreover, the values stored by the σ arrays will vary with the user inputs `ng` and `nmom`. Otherwise, SNAP controls the setting of material data.

First, the Material 1 total interaction probabilities for each group are set,

$$\sigma_{t,1,g} = 1 + (g-1)0.01, \;\; 1 \le g \le ng. \tag{22}$$

The spatial index has been suppressed for brevity, and a material index has been added. The set of cells whose $\sigma_{t,1,g}$ are defined by Eq. (22) is determined by the rules documented in Section 5.4. The data is stored by SNAP in an array sized number of materials times $nx \times ny \times nz$. If Material 2 is present, the total interaction array is set as

$$\sigma_{t,2,g} = 2 + (g-1)0.01, \;\; 1 \le g \le ng. \tag{23}$$

The total interaction probability is divided into two components, an *a*bsorption/loss probability and a *s*cattering probability. The two components must sum to the total:

$$\sigma_t = \sigma_a + \sigma_{s,l=1}. \tag{24}$$

SNAP automatically chooses how total interaction is divided between these components. For Material 1,

$$\sigma_{a,1,g} = 0.5 + (g-1)0.005, \;\; 1 \le g \le ng, \tag{25}$$

and

$$\sigma_{s,1,1,g} = 0.5 + (g-1)0.005, \;\; 1 \le g \le ng. \tag{26}$$

For Material 2,

$$\sigma_{a,2,g} = 0.8 + (g-1)0.005, \;\; 1 \le g \le ng, \tag{27}$$

and

$$\sigma_{s,1,2,g} = 1.2 + (g-1)0.005, \;\; 1 \le g \le ng, \tag{28}$$

Equations (26) and (28) are used to determine total scattering from group g to any other group. However, Eq. (8) requires known coupling from each group to all other groups. If `ng=1`, then the above equations are sufficient. However, for multigroup problems SNAP automatically sets group-to-group scattering probabilities, $\sigma_{s,l,mat,g'\to g}$.

For Material 1, scattering into the same group, g' to g', is 20% of all scattering. All scattering $g' \to g, \;\; 1 \le g \le g'-1$, is 10% of all scattering [Eq. (26)], divided evenly among those $g'-1$ groups. If $g'=1$, this 10% of scattering is included in the scattering $g' \to g'$. Lastly, scattering $g' \to g$, $g'+1 \le g \le ng$, is 70% of all scattering, divided evenly among the $ng-g'$ groups. If $g'=ng$, this 70% of scattering is added to the $g' \to g'$ scattering.

A similar prescription is applied to Material 2. Scattering into the same group is 50% of total. Scattering $g' \to g, \;\; 1 \le g \le g'-1$ is 10%. Scattering $g' \to g, \;\; g'+1 \le g \le ng$ is 40%. The same rules apply when g' is 1 or ng.

The scattering probabilities also vary with the angular moments l, $0 \le l \le nmom-1$. After the data has been set for the group-to-group couplings, SNAP modifies all values for all groups for the varying l. The above data-setting algorithm is applied initially to the $l=1$ moment. `nmom` may be set as high as 4 in the input file. For Material 1, SNAP sets the other moments data as

$$\sigma_{s,l,1,g'\to g} = \begin{cases} \sigma_{s,1,1,g'\to g} \times 0.100, & l=2 \\ \sigma_{s,1,1,g'\to g} \times 0.050, & l=3 \\ \sigma_{s,1,1,g'\to g} \times 0.025, & l=4. \end{cases} \tag{29}$$

Likewise, for Material 2, the angular moment scattering data is set as

$$\sigma_{s,l,2,g'\to g} = \begin{cases} \sigma_{s,1,2,g'\to g} \times 0.800, & l=2 \\ \sigma_{s,1,2,g'\to g} \times 0.480, & l=3 \\ \sigma_{s,1,2,g'\to g} \times 0.288, & l=4. \end{cases} \tag{30}$$

6.6 Source Layout

The source layout ($q_{n,i,j,k,g}$) for SNAP is selected by the user with the `src_opt` input variable. Setting `src_opt=1` instructs SNAP to supply all spatial cells and all energy groups with a volumetric isotropic source of unity. When `src_opt=1`, the same flat source is applied to all groups for the same range of cells given by Eq. (20). The remaining cells have zero source. Likewise, when `src_opt=2`, the flat source for all groups is applied to the range of cells determined by Eq. (21), and the remaining cells have zero source.

A fourth option, `src_opt=3`, is available to the user. `src_opt=3` instructs SNAP to construct the source that corresponds to a preset manufactured solution. SNAP's method of manufactured solutions implementation is described in Section 7.

7. Method of Manufactured Solutions Implementation

Because SNAP is a proxy application, it has been developed in a manner that should allow fast implementation of varying parallel programming models. SNAP uses the method of manufactured solutions (MMS) to ensure that any modification to SNAP's solution algorithms does not affect the resulting solutions. The benefit of MMS is that a manufactured solution can be defined for an arbitrarily sized problem. This is important, because it is impossible to predict the type and size of problems users will be interested in running to test SNAP.

The basic premise of MMS is to choose an analytic solution, insert it into the analytic equation of interest, and determine the corresponding analytic inhomogeneous source. This source must then undergo the same approximations/discretizations applied to attain the numerical equation actually solved by the program. While MMS is typically used in a full verification study, in SNAP the implementation is only designed to help the user/future developer determine if code modifications have affected adversely the solution algorithms.

For MMS in SNAP, several properties of the manufactured solution are chosen for simplifying the implementation. First, SNAP has only vacuum boundary conditions and has not been equipped to handle any boundary sources. Therefore, the manufactured solution should already obey these properties. Moreover, a solution symmetric in space has been found to be a useful aid to detect code bugs. The manufactured solution must also vary with the number of groups and time to fully exercise the SNAP solution algorithms. As in the case of no boundary conditions, the manufactured solution features a zero initial condition. For simplicity, angular dependence in the manufactured solution is not sought.

Considering these points, SNAP's manufactured solution is

$$f_{n,g}(x,y,z,t) = f_g(x,y,z,t) = tg\sin(ax)\sin(by)\sin(cz), \tag{31}$$

where

$$a = \frac{\pi}{lx}, \qquad b = \frac{\pi}{ly}, \qquad c = \frac{\pi}{lz}. \tag{32}$$

If `ng=1`, then $g = 1$ in Eq. (31). Likewise, if the calculation is static, then $t = 1$. For 1-D and 2-D problems the sine terms are dropped to match the level of dimensionality.

When Eq. (31) is substituted into the 3-D analog of Eq. (1), the inhomogeneous source is determined. After much simplification of the scattering source term, the source is found to be

$$\begin{aligned} & q_{n,g}(x,y,z,t) \\ & \quad = \frac{g}{v_g}\sin(ax)\sin(by)\sin(cz) + tg\mu_n a\cos(ax)\sin(by)\sin(cz) \\ & \quad + tg\eta_n b\sin(ax)\cos(by)\sin(cz) + tg\xi_n c\sin(ax)\sin(by)\cos(cz) \\ & \quad + \sigma_{t,g}^{ng}(x,y,z)tg\sin(ax)\sin(by)\sin(cz) \\ & \quad - \sum_{g'=1}^{} \sigma_{s,1,g'\to g}(x,y,z)tg'\sin(ax)\sin(by)\sin(cz). \end{aligned} \tag{33}$$

Due to the definition of the scattering expansions and the angular independence of the manufactured solution, the fixed source from the scattering term contains only an isotropic moment contribution. Note that the first term on the right hand side (RHS) of Eq. (33) is dropped if the problem is time-independent. Time-dependent source terms are amplified with each new time step. Therefore they are stored separately in SNAP for simplicity and summed with other sources as needed. The third and fourth RHS terms of Eq. (29) are dropped if the spatial dimensionality does not require them.

Equation (33) shows the analytic manufactured source. This source must be modified according the same spatial and time discretizations used to determine the numerical equation solved computationally, Eq. (8). Namely, this source will be applied at time centers, t^h, and spatially integrated over the mesh cells. The linear relationship of the source to time makes the former step straightforward. The spatial integration includes sine and cosine terms that are independent each dimension, simplifying the volume triple integral into individual sine and cosine integrations. For example, the *x*-dimension terms' integrations are

$$\frac{1}{\Delta x_i}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}}\sin(ax)\,dx = \frac{1}{\Delta x_i a}\left[\cos\left(ax_{i-\frac{1}{2}}\right) - \cos\left(ax_{i+\frac{1}{2}}\right)\right] \tag{34}$$

and

$$\frac{1}{\Delta x_i}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}}\cos(ax)\,dx = \frac{1}{\Delta x_i a}\left[\sin\left(ax_{i+\frac{1}{2}}\right) - \sin\left(ax_{i-\frac{1}{2}}\right)\right]. \tag{35}$$

Analogous relations are made for the *y*- and *z*-dimensions.

SNAP computes the manufactured solution and stores it as a reference. It computes the source and uses it in the solution algorithm. At the end of a simulation, SNAP reports the difference between

the reference, manufactured solution and the computed one, allowing the user/developer to see how code modifications may have affected the solution scheme.

8. References

[1] "LANL: CCS-3: Performance and Architecture: Software," Available at http://www.ccs3.lanl.gov/PAL/software.shtml," Last accessed 11/30/2012 (2006).

[2] E. E. Lewis and W. F. Miller, Jr., *Computational Methods of Neutron Transport*, American Nuclear Society, La Grange Park, IL, USA (1993).

[3] R. S. Baker et al., "Solution of the First-Order Form of the Multi-Dimensional Discrete Ordinates Equations on a Two-level Heterogeneous Processing System," *Transactions of the ANS*, **105**, 510 (2011).

[4] R.S. Baker and K.R. Koch, "An S_n Algorithm for the Massively Parallel CM-200 Computer," *Nuclear Science and Engineering*, **128**, 312 (1998).

9. Simple Procedure Flowchart (excluding timing calls, collective communications, MPI subroutine calls, error messages, and execution termination)

snap_main (main file)

1. pinit (plib_module)
2. cmdarg (utils_module)
3. open_file (utils_module)
4. open_file (utils_module)
5. version_print (utils_module)
6. input_read (input_module)
    a. input_echo (input_module)
    b. input_check (input_module)
7. close_file (utils_module)
8. pinit_omp (plib_module)
    a. plock_omp (plib_module)
9. pcomm_set (plib_module)
10. setup (setup_module)
    a. setup_alloc (setup_module)
        i. sn_allocate (sn_module)
        ii. data_allocate (data_module)
        iii. control_allocate (control_module)
    b. setup_delta (setup_module)
    c. setup_vel (setup_module)
    d. setup_angle (setup_module)
    e. setup_mat (setup_module)
    f. setup_data (setup_module)
    g. expcoeff (sn_module)
    h. setup_src (setup_module)
        i. mms_setup (mms_module)
        ii. mms_allocate (mms_module)
        iii. mms_cells (mms_module)
        iv. mms_flux_1 (mms_module)
            1. mms_trigint (mms_module)
        v. mms_src_1 (mms_module)
            1. mms_trigint (mms_module)
        vi. mms_flux_1_2 (mms_module)
    i. setup_echo (setup_module)
    j. setup_scatp (setup_module)
        i. open_file (utils_module)
        ii. close_file (utils_module)
11. translv (subroutine)
    a. geom_allocate (geom_module)
    b. solvar_allocate (solvar_module)
    c. assign_thrd_set (thrd_comm_module)
    d. expxs_reg (expxs_module)
    e. expxs_reg (expxs_module)
    f. geom_param_calc (geom_module)

g. expxs_reg (expxs_module)
h. outer (outer_module)
    i. assign_thrd_set (thrd_comm_module)
    ii. outer_src (outer_module)
        1. outer_src_calc (outer_module)
            a. expxs_reg (expxs_module)
            b. expxs_slgg (expxs_module)
    iii. inner (inner_module)
        1. assign_thrd_set (thrd_comm_module)
        2. inner_src (inner_module)
            a. inner_src_calc (inner_module)
        3. sweep (sweep_module)
            a. assign_thrd_set (thrd_comm_module)
            b. waitinit (plib_module)
            c. multiswp_recv_bdry (thrd_comm_module)
                i. plock_omp (plib_module)
                ii. irecv (plib_module)
                iii. irecv (plib_module)
                iv. plock_omp (plib_module)
            d. multiswp_test_pick (thrd_comm_module)
                i. testsome (plib_module)
                ii. plock_omp (plib_module)
                iii. tests (plib_module)
                iv. tests (plib_module)
            e. octsweep (octsweep_module)
                i. no_op_lock_control (thrd_comm_module)
                    1. lock_control (thrd_comm_module)
                        a. plock_omp (plib_module)
                        b. plock_omp (plib_module)
                ii. dim1_sweep (dim1_sweep_module)
                iii. dim3_sweep (dim3_sweep_module)
                    1. sweep_recv_bdry (thrd_comm_module)
                        a. lock_control (thrd_comm_module)
                            i. plock_omp (plib_module)
                            ii. plock_omp (plib_module)
                        b. precv (plib_module)
                        c. precv (plib_module)
                        d. waitall (plib_module)
                        e. lock_control (thrd_comm_module)
                            i. plock_omp (plib_module)
                            ii. plock_omp (plib_module)

2. sweep_send_bdry (thrd_comm_module)
    a. lock_control (thrd_comm_module)
    b. isend (plib_module)
    c. isend (plib_module)
    d. lock_control (thrd_comm_module)

iv. mkba_sweep (mkba_sweep_module)

1. sweep_recv_bdry (thrd_comm_module)
2. sweep_send_bdry (thrd_comm_module)

f. multiswp_send_bdry (thrd_comm_module)

i. plock_omp (plib_module)
ii. isend (plib_module)
iii. isend (plib_module)
iv. plock_omp (plib_module)

4. assign_thrd_set (thrd_comm_module)
5. inner_df_calc (inner_module)

iv. assign_thrd_set (thrd_comm_module)
v. outer_df_calc (outer_module)

i. analyze_pop_calc (analyze_module)
j. analyze_pop_calc (analyze_module)

12. output (output_module)
    a. output_send (output_module)
        i. psend (plib_module)
    b. output_recv (output_module)
        i. precv (plib_module)
    c. output_flux_file (output_module)
        i. open_file (utils_module)
        ii. output_send (output_module)
            1. psend (plib_module)
        iii. output_recv (output_module)
            1. precv (plib_module)
        iv. close_file (utils_module)
    d. mms_verify_1 (mms_module)
13. time_summ (time_module)
14. dealloc_input (dealloc_module)
    a. sn_deallocate (sn_module)
    b. data_deallocate (data_module)
    c. control_deallocate (control_module)
    d. mms_deallocate (mms_module)
15. dealloc_solve (dealloc_module)
    a. geom_dealloc (geom_module)
    b. solvar_dealloc (solvar_module)
16. close_file (utils_module)
17. stop_run (utils_module)

a. `plock_omp (plib_module)`
b. `pend (plib_module)`
c. `EXIT`