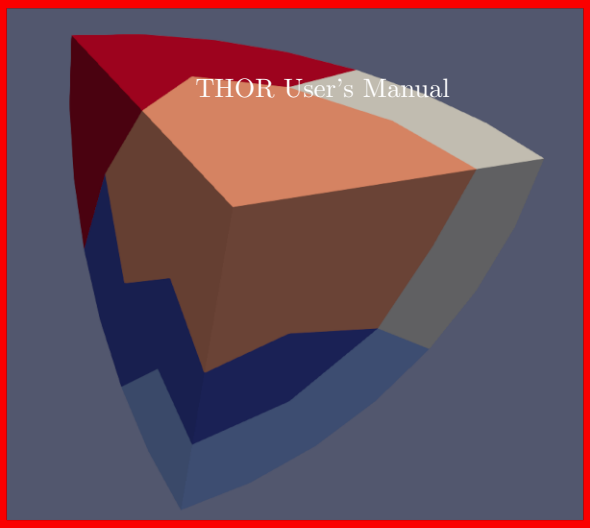
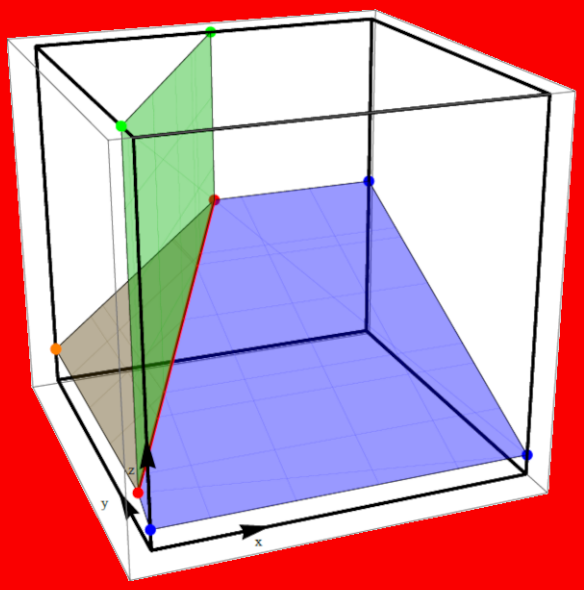
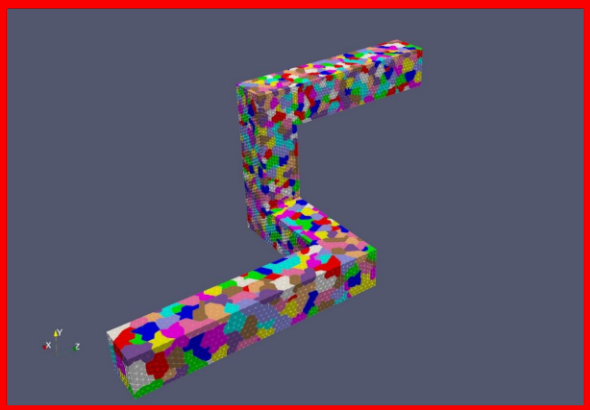
NC STATE UNIVERSITY



NC State University - Radiation Transport Group



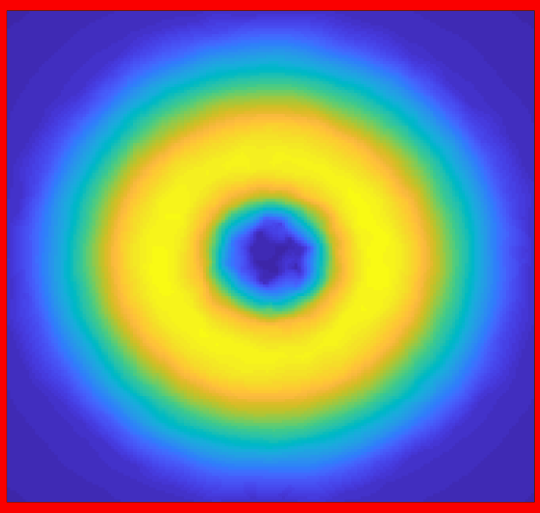
THOR User's Manual



Nicholas Herring[1], Yousry Azmy[1], Sebastian Schunert[2], Raffi Yessayan[3], and Rodolfo Ferrer[4]

[1]North Carolina State University
[2]Idaho National Laboratory
[3]Los Alamos National Laboratory
[4]Studsvik Scandpower

03/16/2022



DEPARTMENT OF NUCLEAR ENGINEERING
NC STATE UNIVERSITY

NC STATE

Revision Log

Revision	Date	Affected Pages	Revision Description
0	03/16/2022	All	Initial Release

Acronyms

Contents

NC STATE

The purpose of the User Manual is to provide the novice user with the necessary instructions to install, compile, and execute the Tetrahedral-grid High Order Radiation (THOR) transport code.

1. System Requirements

- UNIX-like operating system, recommended Ubuntu
- Some type of MPI, recommended `mpich`
- `make`
- Some fortran compiler (recommended `gfortran`)
- (Conditional on setup method) `git`

2. Getting Started

2.1 Obtaining THOR

This section describes how to obtain THOR. Please contact the code owner Yousry Azmy to be added to the project's membership.

Once a user has been added to the projects membership, they can navigate to their desired installation directory and clone THOR from the NCSU GitHub repository using the following command:

```
>> git clone https://github.ncsu.edu/NCSU-Rad-Transport/THOR.git
```

Alternatively, a user who will communicate frequently with THOR's github repository can link their computer's ssh keys with their GitHub account and clone THOR directly by issuing the follwoing command:

```
>> git clone git@github.ncsu.edu:NCSU-Rad-Transport/THOR.git
```

2.2 Obtaining lapack dependencies

THOR depends on certain lapack routines. These are provided with THOR as a submodule. The lapack submodule can be initialized by:

```
>> git submodule update --init
```

The lapack submodule is not expected to change at all. However, if it does, the THOR repository keeps track of the associated version of the lapack repository, so the user may run:

```
>> git submodule update
```

to obtain the latest lapack submodule. If as expected lapack hasn't changed an empty line will be displayed.

2.3 Compiling THOR

This section describes how to compile THOR and its dependencies. The first step is to compile the lapack dependency. To this end, navigate to the installation scripts using (where `<thor_dir>` is the directory THOR was cloned into):

```
>> cd <thor_dir>/contrib/scripts
```

Edit the file `make.inc` to specify the MPI Fortran compiler available on the local machine (if `gfortran` and `mpich` are being used, there is no need to make any changes). Also, if necessary, enter command line that modify the environment to enable the compilation process to find the path to required executables; these typically have the form `>> load module pathname`, where `pathname` is a directory on the local computer where these necessary executables reside. Execute the `build_lapack.sh` script by (first command may not be necessary, it only ensures that `build_lapack.sh` is executable):

```
>> chmod +x build_lapack.sh
>> ./build_lapack.sh <n>
```

where `<n>` is the number of processors. For example, on Idaho National Laboratory's Sawtooth HPCthe compiler is set in `make.inc` via the statement `FORTRAN = mpif90`, and the environment is modified with the command line

```
>> module load mvapich2/2.3.3-gcc-8.4.0".
```

A successful lapack build will conclude the scrolled output on the screen with a table of the form:

```
                     -->   LAPACK TESTING SUMMARY  <--
               Processing LAPACK Testing output found in the TESTING directory
SUMMARY                  nb test run     numerical error         other error
================         ===========     =================       ================
REAL                     1291905         0       (0.000%)        0       (0.000%)
DOUBLE PRECISION         1292717         0       (0.000%)        0       (0.000%)
COMPLEX                  749868          0       (0.000%)        0       (0.000%)
COMPLEX16                749588          1       (0.000%)        1       (0.000%)

--> ALL PRECISIONS       4084078         1       (0.000%)        1       (0.000%)
```

The lapack build may conclude with:

```
make[2]: Leaving directory '<thor_dir>/contrib/lapack/TESTING/EIG'
NEP: Testing Nonsymmetric Eigenvalue Problem routines
./EIG/xeigtstz < nep.in > znep.out 2>&1
make[1]: *** [Makefile:464: znep.out] Error 139
make[1]: Leaving directory '<thor_dir>/contrib/lapack/TESTING'
make: *** [Makefile:43: lapack_testing] Error 2
```

These errors indicate that the system did not have enough memory allocated to lapack to complete the entirety of the testing suite. This is typically not a concern and if these are the sole errors the user is free to continue on to the next step. The correctness of THOR and the lapack linkage can later be verified with the regression tests if the user so desires.

Now, THOR can be compiled. Navigate to the cloned THOR folder, and then to the source folder within it:

```
>> cd \verb"<thor_dir>"/THOR/src
```

and, as before, edit the file `Makefile` to utilize the available MPI Fortran compiler and if necessary modify the environment to enable `make` to locate the compiler (again, for `gfortran` and `mpich`, no changes are necessary). Then type:

```
>> make
```

Successful compilation of THOR will conclude with the line:

```
mv ./thor-1.0.exe ../
```

The THOR executable (named in the above line) can be found here:

```
>> ls <thor_dir>/THOR/
```

that should produce:

```
doc  examples  hello_world  scripts  src   thor-1.0.exe  unit
```

2.4 Running THOR for the first time

Navigate to the `hello_world` directory:

```
>> cd <thor_dir>/THOR/hello_world
```

Check the content of this folder:

```
>> ls
```

It should show the following files:

```
>> ls
hello_world.in hello_world.o hello_world.thrm hello_world.xs
```

These files have the following significance:

- `hello_world.in` is a sample input file to THOR. This file is used to execute THOR.
- `hello_world.thrm` is the corresponding mesh file that is referenced within `hello_world.in`. At this point, it is only important that it is present and has the proper THOR mesh format. Creation of THOR mesh files is covered later in this manual.
- `hello_world.xs` is the corresponding cross section file, also referenced within `hello_world.in`, and again at this point, it is only important that it is present.
- `hello_world.o` is the corresponding output file created by redirecting THOR's standard output. This file can be used to compare THOR's printed output with what it should be upon correct termination of this run.

THOR is invoked with the executable name and the standard input file that is specified as the first and only command line argument passed to THOR.

```
>> ../thor-1.0.exe hello_world.in
```

For parallel execution type:

```
>> mpirun -np <n> ../thor-1.0.exe hello_world.in
```

where <n> is the number of processors. Several files should have been created:

- hello_world.flux
- hello_world.fluxeven
- hello_world.fluxodd
- hello_world.in.log
- hello_world.in_out.csv
- intermediate_output_even.dat
- intermediate_output_odd.dat

The significance of these files will be discussed later. THOR's standard output should start with a banner and conclude with:

```
------------------------------------------------------------
   Region averaged reaction rates
------------------------------------------------------------

-- Region --   0 Volume=   1.500000E+01
   Group          Flux       Fission    Absorption      Fiss Src
       1  9.515584E-01  1.284604E+00  8.564026E-01  1.284604E+00
   Total  9.515584E-01  1.284604E+00  8.564026E-01  1.284604E+00


------------------------------------------------------------
   Execution of THOR completed successfully
------------------------------------------------------------
```

2.4.1 Running THOR Regression Tests

If THOR appears to be running properly, it is recommended that the user run THOR's regression tests after making THOR. To do this, navigate to the regression tests directory:

```
>> cd <thor_dir>/THOR/examples/regression_tests
```

and run the script to run all regression tests

```
>> bash ./runall.bash <n>
```

Here <n> is the number of processors to use (default 1). These tests will take some time. For 24 threads, these tests take about 2 to 3 hours. As such, it is recommended that this testing be performed overnight. It should also be noted that two of the inputs, homogeneous_cube_keig_1G/keig_jfnk.inp and takeda-IV/takeda_jfnk.inp, use JFNK in problems with opposing reflective boundary conditions. THOR currently does not support the running of such problems in parallel. As such the user should manually rerun those problems after the testing script completes unless the user originally specified <n>=1.

2.5 Pre/post Processors

2.5.1 Mesh Converter

The mesh converter is the current recommended pre-processessing utility for THOR meshes. This converter takes a version 4 Gmsh file (tested with version 4.1) and converts it to the THOR mesh input file described in Section 4.3. There are plans to extend this converter to intake other versions of Gmsh and exodus and even perhaps add other output formats in addition to the current THOR mesh output. As of the publishing of this manual, Gmsh version 4.1 is the most recent release Gmsh mesh file format.

To compile the Mesh Converter, navigate to the source folder:

```
>> cd <thor_dir>/pre-processors/Mesh_Converter/src
```

and then make the converter by typing:

```
>> make
```

A successful compilation of the Mesh Converter will conclude with the line:

```
>> mv ./Mesh_Converter.exe ../
```

The Mesh Converter does not have any software requirements that are not also required by THOR.

To run the Mesh Converter, simply invoke the mesh converter binary and follow it immediately with the Gmsh input file (where <gmsh_file> is the name of the Gmsh file):

```
>> <path_to_Mesh_Converter>/Mesh_Converter.exe <gmsh_file>
```

Output will be titled <gmsh_file>_out.thrm. This output will set all boundary conditions to vacuum. If the user desires to set boundary conditions to reflective or incoming flux boundary conditions, then boundary conditions can be specified on the command line when invoking the Mesh Converter by using the -bc indicator. If the -bc indicator is called, then the next six entries will be assumed to be the boundary conditions (integer values) on each of the six primary directions. The order for the boundary conditions specified in this manner are as follows:

```
-x +x -y +y -z +z
```

0 is the integer value for vacuum boundary conditions, 1 is the integer value for reflective boundary conditions, and 2 is the integer value for incident flux boundary conditions. i.e. The following use of the Mesh Converter will convert the Gmsh file and assign reflective boundary conditions to the $-x$, $-y$, and $+y$ boundary faces, and all other boundary conditions will be set to vacuum.

```
>> <path_to_Mesh_Converter>/Mesh_Converter.exe <gmsh_file> -bc 1 0 1 1 0 0
```

It should be noted that if reflective boundary conditions are specified, then the reflective boundaries must all reside on flat boundary surfaces. If the user tries to assign reflective boundary conditions to a direction with a non-flat boundary, then the converter utility will throw an error and terminate. This check is ignored if all boundary conditions are reflective.

2.5.2 THOR_MESH_Generator

THOR MESH generator is an older pre-processor that converts *exodus* and *gmsh* (the legacy version 2) mesh formats to THOR's native mesh format. It is currently undergoing maintenance and will likely have important capabilities simply added to the Mesh Converter after which it may be removed. As such, use of the THOR MESH generator is not currently recommended.

3. Tutorials

THOR currently includes two tutorials to guide new users through the process of creating a mesh with Gmsh, using the Mesh Converter to make it a THOR mesh, and running THOR using that mesh. The first tutorial is the Godiva tutorial, creating a model of the bare Godiva critical experiment sphere, meshing it, and running it with THOR. This tutorial demonstrates basic THOR problem creation and running concepts.

The second tutorial is the BeRP tutorial, creating a model of the BeRP ball with 3 inches of polyethylene reflector surrounding the BeRP ball. This tutorial demonstrates a more advanced problem involving multiple materials and regions.

Since both systems are physically equivalent to one-dimensional spherical problems, the symmetry is taken advantage of and the problems are modeled using a one/eighth model with reflective boundary conditions to be equivalent to the full spheres for the tutorial. In the tutorial folders, `<thor_dir>/THOR/examples/Godiva_tutorial` and `<thor_dir>/THOR/examples/BeRP_tutorial`, versions of these tutorials with full spheres, half spheres, and quarter spheres are also included along with reference results.

3.1 Godiva Tutorial

Godiva is an un-shielded, pulsed, nuclear burst reactor. It is essentially a homogeneous sphere of highly enriched uranium with a diameter of 30 cm, that was operated by inserting a piston of fissile material [2]. In this tutorial the critical benchmark configuration described in Ref. [1] is considered. The geometry that is modeled by THOR is a homogeneous sphere of radius 8.7407 cm discretized by tetrahedra similar to Fig. 3.1. The energy domain is discretized with six energy groups, and cross sections are provided by [1].

This tutorial first explains how a tetrahedral mesh is created for the Godiva problem, then the cross sections data input is discussed, and finally the standard input to THOR is covered. The input files discussed below for the Godiva tutorial are located in:

```
>> <thor_dir>/THOR/examples/Godiva_tutorial
```

3.1.1 Godiva Mesh

The workflow described here is suitable if the user has access to a compatible version of Gmsh. Any version 4 Gmsh should work, but the example specifically performed here was done using Gmsh version 4.10.1.

Begin by navigating to the location of the Godiva Gmsh geometry files, which are found in:

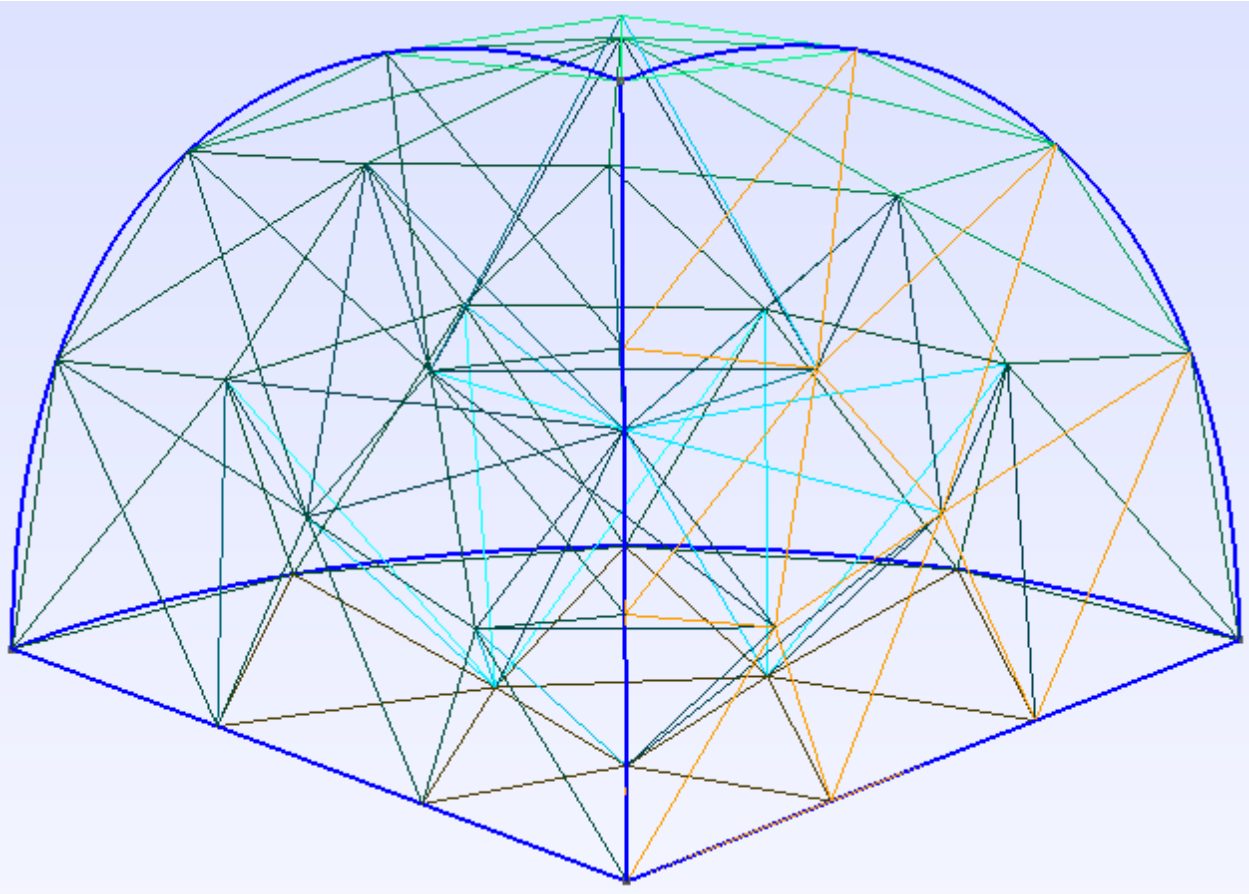


Figure 3.1: Coarse mesh for Godiva problem

```
<thor_dir>/THOR/examples/Godiva_tutorial/mesh_create/
```

Opening the file geometry file `godiva_octant.geo` in a text editor, it can be observed that the model is created by removing the negative portions of each direction from a sphere centered at the origin. For more details on creating original Gmsh inputs, see the Gmsh reference manual.

Open `godiva_octant.geo` in Gmsh and run the "3D" command from the "Mesh" dropdown menu under "Modules". The mesh should be generated and now become visible in the GUI. Now, select the "Save" command from the same "Mesh" dropdown to save the generated mesh to `godiva_octant.msh`. This mesh may be compared to the provided `godiva_octant_msh.ref`, however they may differ slightly if the versions differ or if optimization of the mesh is employed.

The gmsh file `godiva_octant.msh` is converted to THOR's native mesh format by executing the Mesh Converter with the command line:

```
>> <thor_dir>/THOR/pre-processors/Mesh_Converter/Mesh_Converter.exe convert_godiva.msh
   -bc 1 0 1 0 1 0
```

Note that since we are modeling the fully positive octant of the sphere we are setting all of the flat negative faces to be reflective (see Section 2.5.1 for more details). After successful completion of the conversion, the following printout should appear:

```
---------------------- Reading in gmsh:
Progress:**************************************************************************
---------------------- Calculating Adjacencies:
Progress:**************************************************************************
---------------------- Outputting thrm file:
Progress:**************************************************************************
---------------------- Calculating volumes:
Progress:**************************************************************************
Region 5 volume:   3.3099604624050352E+02
Region 5 equivalent radius:   4.2911933629822876E+00
Total system volume:   3.3099604624050352E+02
Equivalent radius:   4.2911933629822876E+00
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------ THOR mesh converter successful ----------------------
---------------------- Output written to godiva_octant.thrm
```

The file `godiva_octant.thrm` should result from this execution for use by THOR. This mesh may be compared to the provided `godiva_octant_thrm.ref`, which it should match if `godiva_octant.msh` matches `godiva_octant_msh.ref`. Notice that the given volume for Region 5 (the Godiva eighth of a sphere as seen in `godiva_octant.geo`) is 330.996 cm^3, but the actual octant volume for the Godiva sphere is 349.653 cm^3. The ratio of the actual volume to the meshed volume is then 1.056366, which will come in handy later. This concludes the mesh generation step for this tutorial.

3.1.2 Cross section data

The user should now move `godiva_octant.thrm` to the input file location

```
<thor_dir>/THOR/examples/Godiva_tutorial/run_files/
```

and navigate there to continue the tutorial.

The THOR cross section file for the Godiva benchmark is provided by `godiva.xs`. THOR uses a custom cross section format that is explained in detail in Sec. 4.4.

At the end of Section 3.1.1, it was observed that there was a discrepancy in the volume of the Godiva mesh compared to the original problem. To preserve material mass, the cross sections must be altered by increasing them by a factor of 1.056366. In THOR, the user need not alter the cross sections themselves to make this adjustment. Instead, THOR will automatically adjust reaction and material attenuation calculations by a given density factor for each region. By default, this factor is 1.0, which will lead to use of the original cross sections unaltered. However, the user may specify density factors in a density factor file, described in Section **??**. For this tutorial, this density factor adjustment is provided by `godiva_octant.dens`

3.1.3 THOR input file and executing THOR

The THOR input file is `godiva_octant.inp`. THOR uses a keyword-based input that is listed in Sect. 4.1. The Godiva tutorial input file is verbose and some parameters are ignored as they are not relevant to the problem. Upon running THOR, a verbose form of the input will always be echoed, and ignored parameters will be highlighted as such.

```
type                  keig
keigsolver            pi
lambda                0
inflow                no
piacc                 errmode
page_sweep            no
page_refl             save
page_iflw             all
kconv                 1e-8
innerconv             1e-12
outerconv             1e-7
maxinner              4
maxouter              5000
jfnk_krsze            25
jfnk_maxkr            250
jfnk_method           flat
initial_guess         no
restart_out           no
ipiter                0
print_conv            yes
density_factor        fromfile godiva_octant.dens
execution             yes
mesh                  ./godiva_octant.thrm
source                source.dat
flux_out              no
xs                    ./godiva.xs
vtk_flux_out          yes
vtk_mat_out           yes
vtk_reg_out           no
vtk_src_out           no
cartesian_map_out     no
print_xs              no
ngroups               1
pnorder               0
pnread                0
upscattering          yes
multiplying           yes
scatt_mult_included   yes
qdtype                levelsym
qdorder               4
cartesian_map         no
point_value_locations no
region_map            5 1
```

The Godiva tutorial is solved with THOR via the command line:

```
>> <thor_dir>/THOR/thor-1.0.exe godiva_octant.inp
```

Completion of execution of the Godiva tutorial is indicated by the printout:

```
-----------------------------------------------------
   Execution of THOR completed successfully
-----------------------------------------------------
```

THOR provides the following output that is discussed in this tutorial:

- The final estimate of the multiplication factor is printed under "Execution Summary", "Final eigenvalue". In this case the value is 0.935. This is not close to critical because the mesh that is created is very coarse.
- A summary of group-wise, region-averaged reaction rates is provided for each region identifier separately under "Region averaged reaction rates". The volume of each region, and group-wise fluxes, fission, absorption, and fission source rates are listed.
- Two vtk formatted files, `godiva_octant_flux.vtk` contains spatial flux maps, and `godiva_octant_mat.vtk` contains the material map. These files can be opened with the paraview post-processing tool that is available here.

A plot of the fast flux using ParaView 5.10.0 for this run is shown in Figure 3.2.

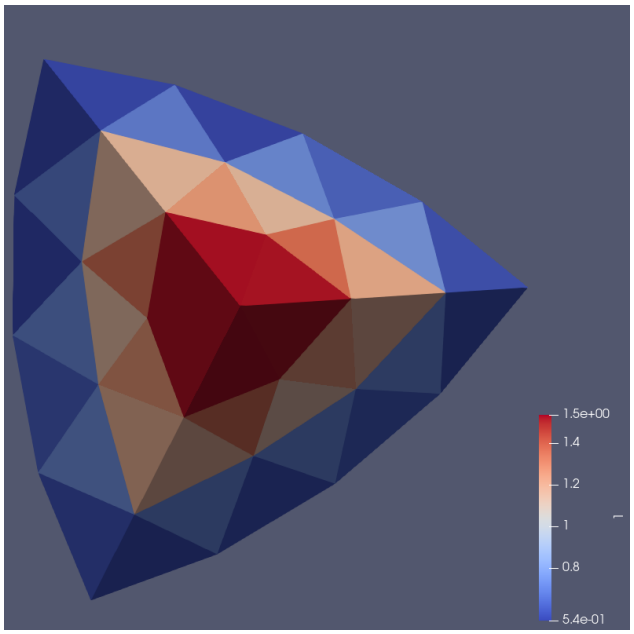


Figure 3.2: Fast flux for Godiva tutorial.

The reaction rate summary is given by:

```
------------------------------------------------------------
   Region averaged reaction rates
------------------------------------------------------------

-- Region --    5 Volume=   3.309960E+02
   Group          Flux       Fission    Absorption      Fiss Src
       1  8.317355E-01  1.381271E-01  4.883317E-02  1.381271E-01
       2  1.549185E+00  2.298278E-01  9.364997E-02  2.298278E-01
       3  9.627054E-01  1.353285E-01  5.854598E-02  1.353285E-01
       4  1.585196E+00  2.152633E-01  9.782396E-02  2.152633E-01
       5  1.137791E+00  1.819135E-01  8.630310E-02  1.819135E-01
       6  1.682607E-01  4.366399E-02  2.265428E-02  4.366399E-02
   Total  6.234875E+00  9.441242E-01  4.078105E-01  9.441242E-01
```

The results can be improved by increasing the refinement of the mesh. This can be achieved by reducing the mesh size parameter in the `godiva_octant.geo` file, that parameter is `MeshSize{ PointsOf{ Volume{:}; } };` which can be seen is set to 4.

3.2 Polyethylene Shielded BeRP Ball Tutorial

TODO: Fix up this beginning

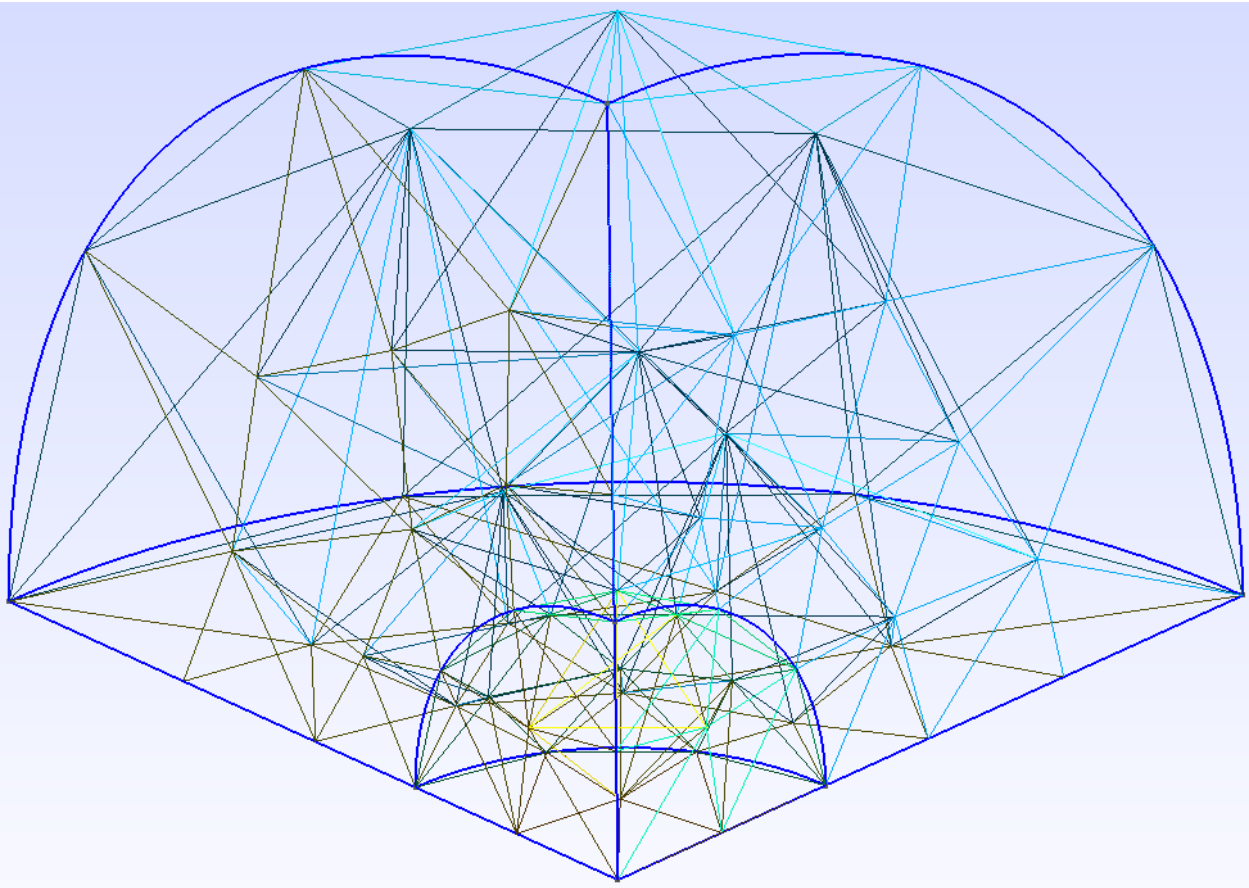


Figure 3.3: Coarse mesh for BeRP Ball with Poly Shield

This tutorial first explains how a tetrahedral mesh is created for the BeRP ball surrounded by a poly shield, then the cross sections data input is discussed, and finally the standard input to THOR is covered. The input files discussed below for the BeRP tutorial are located in:

```
>> <thor_dir>/THOR/examples/BeRP_tutorial
```

3.2.1 BeRP Ball Mesh

The workflow described here is suitable if the user has access to a compatible version of Gmsh. Any version 4 Gmsh should work, but the example specifically performed here was done using Gmsh version 4.10.1.

Begin by navigating to the location of the BeRP Gmsh geometry files, which are found in:

```
<thor_dir>/THOR/examples/Godiva_tutorial/mesh_create/
```

Opening the file geometry file `berp_octant.geo` in a text editor, it can be observed that the model is created by removing the negative portions of each direction from a sphere centered at the origin surrounded by another sphere centered at the origin. For more details on creating original Gmsh inputs, see the Gmsh reference manual.

Open `berp_octant.geo` in Gmsh and run the "3D" command from the "Mesh" dropdown menu under "Modules". The mesh should be generated and now become visible in the GUI. Now, select the "Save" command from the same "Mesh" dropdown to save the generated mesh to `berp_octant.msh`. This mesh may be compared to the provided `berp_octant_msh.ref`, however they may differ slightly if the versions differ or if optimization of the mesh is employed.

The gmsh file `berp_octant.msh` is converted to THOR's native mesh format by executing the Mesh Converter with the command line:

```
>> <thor_dir>/THOR/pre-processors/Mesh_Converter/Mesh_Converter.exe berp_octant.msh
   -bc 1 0 1 0 1 0
```

Note that since we are modeling the fully positive octant of the sphere we are setting all of the flat negative faces to be reflective (see Section 2.5.1 for more details). After successful completion of the conversion, the following printout should appear:

```
---------------------- Reading in gmsh:
Progress:**************************************************************************
---------------------- Calculating Adjacencies:
Progress:**************************************************************************
---------------------- Outputting thrm file:
Progress:**************************************************************************
---------------------- Calculating volumes:
Progress:**************************************************************************
Region 6 volume:   2.6511347836966994E+01
Region 6 equivalent radius:   1.8497558414045954E+00
Region 7 volume:   6.6374040817476271E+02
Region 7 equivalent radius:   5.4113202652319341E+00
Total system volume:   6.9025175601172975E+02
Equivalent radius:   5.4824287000562011E+00
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
---------------------- THOR mesh converter successful ----------------------
---------------------- Output written to berp_octant.thrm
```

The file `berp_octant.thrm` should result from this execution for use by THOR. This mesh may be compared to the provided `berp_octant_thrm.ref`, which it should match if `berp_octant.msh` matches `berp_octant_msh.ref`. Notice that the given volume for Region 6 (the BeRP eighth of a sphere as seen in `berp_octant.geo`) is 26.511 cm^3, but the actual octant volume for the BeRP ball is 28.591 cm^3. Similarly, the given volume for Region 7 (the Poly shield eighth of a sphere as seen in `berp_octant.geo`) is 663.740 cm^3, but the actual octant volume for the Poly shield is 749.965 cm^3. The ratio of the actual volume to the meshed volume for these two regions is then 1.078425 and 1.129907 respectively, which will come in handy later. This concludes the mesh generation step for this tutorial.

3.2.2 Cross section data

The user should now move `berp_octant.thrm` to the input file location

```
<thor_dir>/THOR/examples/BeRP_tutorial/run_files/
```

and navigate there to continue the tutorial.

The THOR cross section file for the BeRP benchmark is provided by `berp.xs`. THOR uses a custom cross section format that is explained in detail in Sec. 4.4.

At the end of Section 3.2.1, it was observed that there was a discrepancy in the volume of the BeRP mesh compared to the original problem. To preserve material mass, the cross sections must be altered by increasing them by a factor of 1.078425 in the BeRP ball and 1.129907 in the polyethylene. In THOR, the user need not alter the cross sections themselves to make this adjustment. Instead, THOR will automatically adjust reaction and material attenuation calculations by a given density factor for each region. By default, this factor is 1.0, which will lead to use of the original cross sections unaltered. However, the user may specify density factors in a density factor file, described in Section **??**. For this tutorial, this density factor adjustment is provided by `berp_octant.dens`

3.2.3 THOR input file and executing THOR

The THOR input file is `berp_octant.inp`. THOR uses a keyword-based input that is listed in Sect. 4.1. The BeRP tutorial input file is not verbose and all parameters given are used, though not all are necessary since many are the same as the default values. Upon running THOR, a verbose form of the input will always be echoed, and ignored parameters will be highlighted as such.

```
print_conv yes
lambda 0
type keig ; keigsolver pi ; piacc errmode
page_refl save
innerconv 1E-8 ; outerconv 1E-6
maxinner 5 ; maxouter 5000

mesh ./berp_octant.thrm
xs ./berp.xs
density_factor fromfile berp_octant.dens
vtk_flux_out yes
vtk_mat_out yes

qdtype levelsym ; qdorder 4

region_map
6 1
7 2
```

The BeRP tutorial is solved with THOR via the command line:

```
>> <thor_dir>/THOR/thor-1.0.exe berp_octant.inp
```

Completion of execution of the BeRP tutorial is indicated by the printout:

```
-----------------------------------------------------------
   Execution of THOR completed successfully
-----------------------------------------------------------
```

THOR provides the following output that is discussed in this tutorial:

- The final estimate of the multiplication factor is printed under "Execution Summary", "Final eigenvalue". In this case the value is 1.08.
- A summary of group-wise, region-averaged reaction rates is provided for each region identifier separately under "Region averaged reaction rates". The volume of each region, and group-wise fluxes, fission, absorption, and fission source rates are listed.
- Two vtk formatted files, `berp_octant_flux.vtk` contains spatial flux maps, and `berp_octant_mat.vtk` contains the material map. These files can be opened with the paraview post-processing tool that is available here.

A plot of the thermal flux using ParaView 5.10.0 for this run is shown in Figure 3.4.

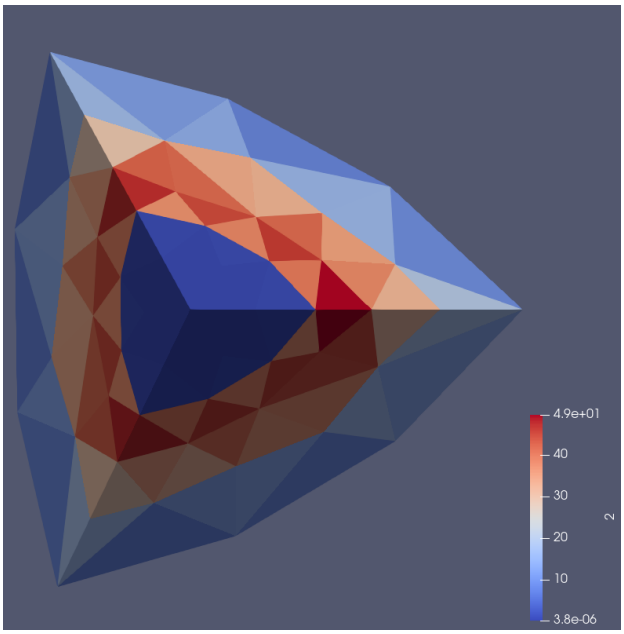


Figure 3.4: Thermal flux for BeRP tutorial.

The reaction rate summary is given by:

```
-----------------------------------------------------------
   Region averaged reaction rates
-----------------------------------------------------------

-- Region --   6 Volume=   2.651135E+01
   Group          Flux       Fission    Absorption      Fiss Src
       1  4.796767E+00  4.145841E-01  4.913242E-01  1.322154E+00
       2  1.094419E-02  3.641023E-03  1.236176E-01  1.054076E-02
   Total  4.807711E+00  4.182251E-01  6.149418E-01  1.332695E+00
```

```
-- Region --   7 Volume=   6.637404E+02
   Group          Flux       Fission    Absorption      Fiss Src
       1  3.984566E-01  0.000000E+00  1.461140E-04  0.000000E+00
       2  3.051344E-01  0.000000E+00  6.844744E-03  0.000000E+00
   Total  7.035909E-01  0.000000E+00  6.990858E-03  0.000000E+00
```

The results can be improved by increasing the refinement of the mesh. This can be achieved by reducing the mesh size parameter in the `berp_octant.geo` file, that parameter is
`MeshSize{ PointsOf{ Volume{:}; } };` which can be seen is set to 6 for the poly and 2 for the BeRP.

4. Input Format

4.1 THOR Inputs

The THOR transport solver has distinct user input formats for the following separate input files:

- Standard Input (Sec. 4.2) - The primary input file to be run by THOR. All other input files will either be listed in this file, or assumed to be the default filenames as described in Section 4.2. This is the only input file given to THOR by way of the command line.
- Mesh File (Sec. 4.3) - File containing the physical 1st order tet mesh for the problem.
- Cross Section File (Sec. 4.4) - File containing cross sections for the problem.
- Density Factor File (Sec. **??**) - File containing the density factors for each adjustment of cross sections in each region.
- Initial Guess File (Sec. **??**) - File containing the initial guess for the problem.
- Source File (Sec. **??**) - File containing the source for a fixed source problem.
- Inflow File (Sec. **??**) - File containing the boundary condition inflow data for a fixed source problem.

This chapter describes the input formats of the THOR transport solver.

4.2 Standard Input

THOR input is organized in blocks. The blocks are

- The first line must contain a user selected name for the problem.
- `problem`: general parameters to define the problem to be solved
- `inout`: Names of inputs and outputs files and toggles for specific output.
- `cross_sections`: parameters pertaining to the cross section data.
- `quadrature`: parameters pertaining to the angular quadrature.
- `postprocess`: parameters pertaining to postprocessing outputs. `cartesian_map` sets up an overlayed Cartesian mesh that fluxes and reactions rates are averaged over. The Cartesian mesh is defined by the minimum and maximum coordinates for each direction (x, y, z) and number of subdivisions between. `point_value_locations` allows extraction of flux values at user provided points.

- `regionmap`: mapping from region id to cross section id. Region ids are an integer assigned to to each tetrahedral element that are used to group elements into regions or blocks (see Sect. **??**). Cross section ids are indices that identify sets of cross sections provided in the cross section input file (see Sect. **??**).

Blocks are delineated with `start` and `end` keywords like this:

```
start <block_name>
   key1 = value1 ; key2 = value2
   key3 = value3
end <block_name>
```

Each blocks contains several keyword-value pairs. Multiply assignments can be placed on the same line if they are separated by `;`. The keyword-value pairs can be provided in an arbitrary order. All keywords are listed in Table **??**.

The `regionmap` block does not contain keyword-value pairs. Instead, it maps region ids to cross section ids. We denote by `min_reg` and `max_reg` the smallest and largest region ids in the mesh file. The number of entries in the `regionmap` field must then be `num_entries` = `max_reg` - `min_reg` + 1. The assignment is best illustrated for an example. Let us assume that `min_reg` = -1 and `max_reg` = 2 and we want to assign

```
-1 -> 12
 0 -> 1
 1 -> 1
 2 -> 3
```

Then the `regionmap` block is given by:

```
start regionmap
 12 1 1 3
end regionmap
```

Unused region ids can be accommodated by padding the entries in the regionmap field.

4.3 THOR Mesh Format

Line 1: number of vertices

Line 2: number of elements

Line 3: unused enter 1

Line 4: unused enter 1

Block 1: vertex coordinates, number of lines = number of vertices; each line is as follows: vertex_id (integer) x-coordinate (Real) y-coordinate (Real) z-coordinate (Real)

Block 2: region and source id assignments, number of lines = number of elements; each line is as follows: element_id region_id source_id (all integers). For setting up Monte Carlo on the tet mesh, this block can be ignored.

NC STATE

Table 1: Keywords of THOR Transport Solver Application

Keyword	Type	Options	Explanation
Block: problem			
type	string	keig/fsrc	Problem type. Either eigenvalue (keig) or fixed source (fsrc)
keigsolver	string	pi/jfnk	Solve type for keig. Either power iteration (pi) or Jacobian-Free Newtor
lambda	integer	-	Expansion order, negative number indicates reduced set
inflow	string	yes/no	If fixed inflow boundary conditions are provided for fsrc problems
piacc	string	errmode/none	Type of power iteration acceleration: none or error mode extrapolation
sweep	string	precomp	Must be set to precomp at this point. (Redundant keyword)
page_sweep	string	yes/no	If the sweep path is saved or is paged to scratch file when not needed
page_refl	string	page/save/inner	If significant angular fluxes are paged to/from scratch file (page), stored
page_iflw	string	bygroup/all	If inflow information is loaded to memory completely (all) or for each gr
kconv	Real	-	Stopping criterion for eigenvalue
innerconv	Real	-	Stopping criterion for group flux during inner iteration
outerconv	Real	-	Stopping criterion for group flux during outer/power iteration
maxinner	integer	-	Maximum number of inner iterations
maxouter	integer	-	Maximum number of outer/power iterations
jfnk_krsze	integer	-	Maximum size of Krylov subspace during jfnk
jfnk_maxkr	integer	-	Maximum number of Krylov iterations
jfnk_method	string	outer/flat/flat_wds	Type of jfnk formulation, see []
initial_guess	string	yes/no	If an initial guess file should be read
save_restart	string	yes/no	If a restart file should be written
ipiter	integer	-	Number of initial power iterations for jfnk
print_conv	string	yes/no	If convergence monitor is written to file `thor.convergence`
density_factor	string	no/byvolume/fromfile	Density factor options: use no density factors (no), use density factors a
execution	string	yes/no	If yes problem is executed, if no then input is only read and checked.

Table 2: Keywords of THOR Transport Solver Application

Keyword	Type	Options	Explanation
Block: inout			
`mesh_file`	string	-	Name of the mesh file
`inflow_file`	string	-	Name of the boundary inflow informatic
`source_file`	string	-	Name of the volumetric source file
`flux_file`	string	-	Name of the THOR formatted output fl
`xs_file`	string	-	Name of the cross section file
`density_factor_file`	string	-	Name of the density factor file
`quad_file`	string	-	Name of the angular quadrature file
`vtk`	string (multiple)	flux/mat/reg/src	Which information is written to vtk forr
`vtk_flux_file`	string	-	Name of the vtk flux file
`vtk_mat_file`	string	-	Name of the vtk material file
`vtk_reg_file`	string	-	Name of the vtk region file
`vtk_src_file`	string	-	Name of the vtk volumetric source file
`restart_file`	string	-	Name of the restart file written when sa
`inguess_file`	string	-	Name of the initial guess file if initial_gu
`cartesian_map_file`			Name of the file that the cartesian map
`print_xs`	string	yes/no	If cross sections are echoed to standard
Block: cross_sections			
`ngroups`	integer	-	Number of energy groups
`pnorder`	integer	-	Spherical harmonics order used for scatt
`pnread`	integer	-	Spherical harmonics expansion provided
`upscattering`	string	yes/no	Read upscattering data from cross sectio
`multiplying`	string	yes/no	If the cross section file contains fission in
`scatt_mult_included`	string	yes/no	If the scattering data includes the $2l+1$
Block: quadrature			
`qdtype`	string	levelsym/legcheb/fromfile	Quadrature type: level-symmetric, Lege
`qdorder`	integer	-	Order of the angular quadrature
Block: postprocess			
`cartesian_map`	Real/integer (9 entries)	-	xmin, xmax, nx, ymin, ymax, ny, zmin,
`point_value_locations`	Real (3 N)	-	N is the number of points, (x,y,z) coordi

Block 3: element descriptions, the vertex_ids that form each element. Number of lines = number of elements; each line is as follows: element_id vertex_id1 vertex_id2 vertex_id3 vertex_id4 (all integers).

Next line: number of boundary face edits

Block 4: boundary face descriptions. All exterior faces associated with their boundary condition id, number if lines = number of boundary face edits; each line is as follows: element_id local_tetrahedron_face_id boundary_condition_id.

Explanation: local_tetrahedron_face_id: natural local id of tetrahedrons face which is the id of the vertex opposite to this face. Note: indexed 0-3. boundary_condition_id: value = 0: vacuum BC value = 1: reflective BC value = 2: fixed inflow

Next line: number of adjacency list entries

Block 5: adjacency list, number of lines = number of adjacency list entries; each line is as follows: element_id face_id neighbor_id neighbor_face_id. Explanation: The element_id is the current element. The neighbor across the face indexed by face_id has the element id neighbor_id and the its own local index for the said common face is neighbor_face_id.

4.4 THOR Cross Section Format

```
Line 1: number of materials

Block 1: each entry in this block contains cross sections for a single material. Each entry contains  L

Entry line 1: material_id
Entry line 2: fission_spectrum_1 fission_spectrum_2  fission_spectrum_G
Entry line 3: energy_group_boundary_1 energy_group_boundary_3  energy_group_boundary_G
Entry line 4: fission_xs_1 fission_xs_2 fission_xs_3  fission_xs_G
Entry line 5: nu_bar_1 nu_bar_2  nu_bar_G
Entry line 6: total_xs_1 total_xs_2  total_xs_G
Entry line 7: sig_scat_{0, 1->1} sig_scat_{0, 2->1}  sig_scat_{0, G->1}
Entry line 8: sig_scat_{0, 1->2} sig_scat_{0, 2->2}  sig_scat_{0, G->2}

        :
Entry line G + 6: sig_scat_{0, 1->G} sig_scat_{0, 2->G}  sig_scat_{0, G->G}
Entry line G + 7: sig_scat_{1, 1->1} sig_scat_{1, 2->1}  sig_scat_{1, G->1}
Entry line G + 8: sig_scat_{1, 1->2} sig_scat_{1, 2->2}  sig_scat_{1, G->2}

        :
Entry line 2 * G + 6: sig_scat_{1, 1->G} sig_scat_{1, 2->G}  sig_scat_{1, G->G}
        :
```

- G = total number of groups.
- L = scattering expansion.
- fission_spectrum_g: fraction of neutrons born in fission that appear in energy group g.

- energy_group_boundary_g: currently unused, can be filled with 0s. Upper bound of energy group g.
- fission_xs_g: fission cross section (NOTE: not nu_bar * fission_xs) in group g.
- nu_bar_g: average number of neutrons released by fission caused by a neutron in energy group g.
- total_xs_g: total cross section in energy group g.
- sig_scat_l, g-¿g: l-th Legendre polynomial moment of the scattering cross section from group g to g. The (2 * l + 1) factor may be included in the value of the cross section or not, THOR can handle both cases. It needs to be specified separately every time.

4.5 List of all Inputs and Outputs of THOR transport solver

4.6 THOR Mesh Generator

This section discusses the input of THOR's mesh generator tool. The mesh generator tool provides the following capabilties:

- Convert exodus [5] formatted meshes to THOR format.
- Convert gmsh [3] formatted meshes to THOR format.
- Convert universal file format meshes to THOR format.
- Split the elements of hexahedra and wedge (triangular prisms) meshes into tetrahedra before comverting to THOR mesh format.
- Conversion of legacy THOR mesh format to new format.

4.6.1 Compilation and Invocation

Navigate to:

```
>> cd /home/<usr>/projects/THOR/pre-processors/THOR_Mesh_Generator/src
```

and make the application:

```
>> make
```

The executable `Thor_Mesh_Generator.exe` should have been created here:

```
>> /home/<usr>/projects/THOR/pre-processors/THOR_Mesh_Generator/Thor_Mesh_Generator.exe
```

The application is invoked with:

```
>> /path/to/Thor_Mesh_Generator.exe -i standard_input
```

where `standard_input` is the standard input file.

Remark: Conversion of exodus files to gmsh files relies on using libMesh's `mesh-tool` [4]. You must compile libMesh and set the environment variable `THOR_LIBMESH_DIRECTORY`.

4.6.2 Format of the THOR's mesh generator standard input

The standard input file of the THOR mesh generator contains the following lines:

```
input_mesh_file
output_mesh_file
region_id_file
source_id_file
boundary_id_file
```

where `input_mesh_file` and `output_mesh_file` are required parameters, while the remaining parameters are optional. If a parameter is omitted, the line should just be left blank (that means that e.g. `source_id_file` will always be provided on line 4 regardless of whether `region_id_file` was provided).

The purpose of these files is as follows:

- `input_mesh_file`: name of the input mesh file. Note that the file ending matters: exodus is `.e` and gmsh is `.msh`, because the mesh format is inferred from it.
- `output_mesh_file`: name of the THOR mesh format files. Use file ending `.thrm`.
- `region_id_file`: name of file that contains instructions to reassign region (sometimes called block) ids. Formatting instructions for this file is provided in Sect. 4.6.3.
- `source_id_file`: name of file that contains instructions to reassign source ids. Formatting instructions for this file is provided in Sect. 4.6.3.
- `boundary_id_file`: name of file that contains instructions to reassign boundary ids. Formatting instructions for this file is provided in Sect. 4.6.3.

4.6.3 Formatting instructions for regions, source, and boundary id reassignment files

IDs are integers that are assigned to each tetrahedral element to group it into a region, a source region or assigned to boundary faces to group it into a set of faces for boundary assignment. The file format for id reassignment files is:

```
n
old_id_<1> new_id_<1>
  :
  :
old_id_<n> new_id_<n>
```

The meaning is as follows:

- `n`: the number of instructions in the file (i.e. the number of lines following this line).
- `old_id_<j>`: the j-th old id that will be replaced by `new_id_<j>`.
- `new_id_<j>`: the j-th new id that will replace `old_id_<j>`

Remarks:

- Note, each instruction needs to be on a different line.
- The boundary id characterizes the boundary condition. It must be 0, 1, or 2, where 0 is a vacuum, 1 is a reflective, and 2 is a fixed inflow boundary.

4.7 Testing

THOR provides a convenient test harness that can be executed by the user by navigating to the `scripts` directory that exists as subdirectory for all applications; for example, for the THOR transport solver the test file is located in:

```
>> /home/<usr>/projects/THOR/THOR/scripts
```

The tests are executed via the `run_thor_tests.py` python script. The execution of the python script requires *python3*. Tests are executed by:

```
>> cd /home/<usr>/projects/THOR/THOR/scripts
>> python run_thor_tests.py
```

The output of the test script should look like this:

```
-------------------------------------
Test  1 name success
  :
Test  N name success
-------------------------------------
Successes:  N            Failures:  0
```

Bibliography

[1] Takumi ASAOKA, Norio ASANO, Hisashi NAKAMURA, Hiroshi MIZUTA, Hiroshi CHICHIWA, Tadahiro OHNISHI, Shun ichi MIYASAKA, Atsushi ZUKERAN, Tsuneo TSUTSUI, Toichiro FUJIMURA, and Satoru KATSURAGI. Benchmark tests of radiation transport computer codes for reactor core and shield calculations. *Journal of Nuclear Science and Technology*, 15(1):56–71, 1978.

[2] M.J. Engelke, E.A. Bemis Jr., and J.A. Sayeg. Neutron tissue dose rate survey for the godiva ii critical assembly. Technical report, Los Alamos National Laboratory, 1961.

[3] Christophe Geuzaine and Jean-Franois Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[4] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3–4):237–254, 2006. https://doi.org/10.1007/s00366-006-0049-3.

[5] L A Schoof and V R Yarberry. Exodus ii: A finite element data model. 9 1994.