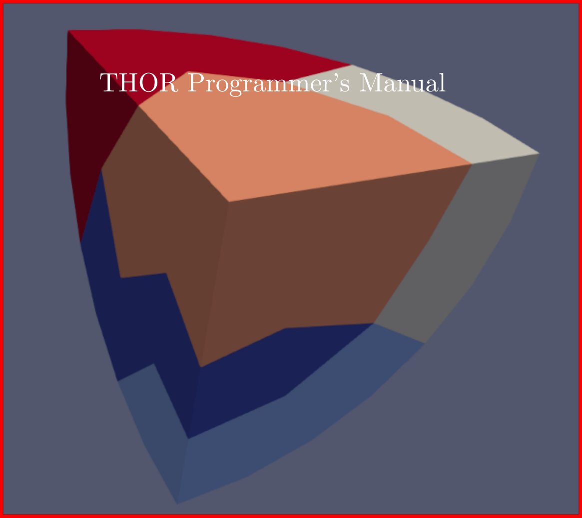
NC State University - Radiation Transport Group

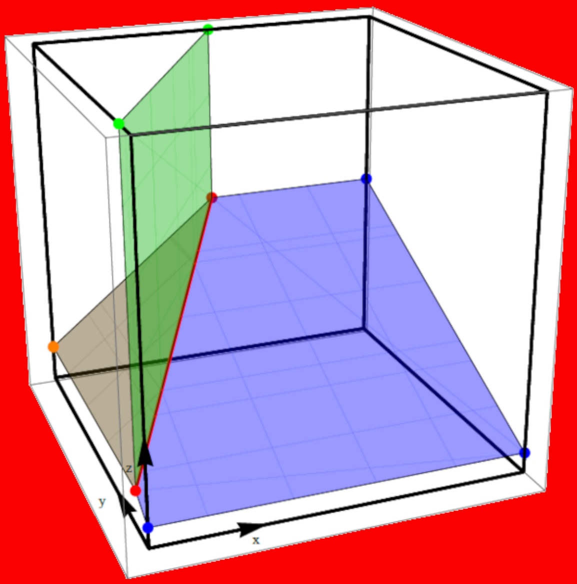
THOR Programmer's Manual

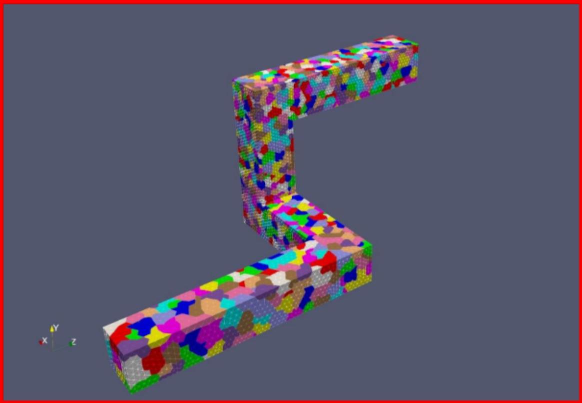
Nicholas Herring[1], Yousry Azmy[1], Sebastian Schunert[2], Raffi Yessayan[3], and Rodolfo Ferrer[4]

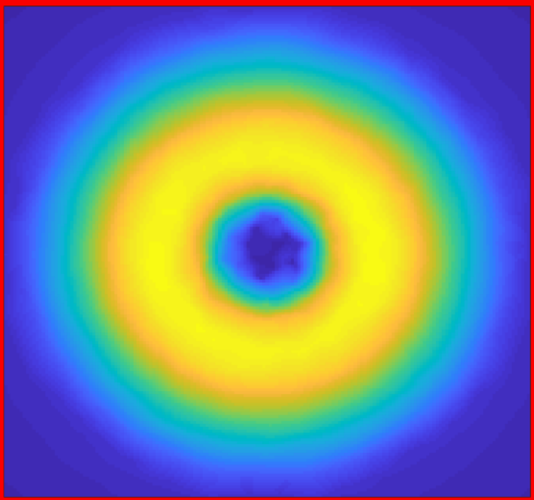
[1]North Carolina State University
[2]Idaho National Laboratory
[3]Los Alamos National Laboratory
[4]Studsvik Scandpower

03/16/2022









Revision Log

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

Acronyms

Contents

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.
- mpi
- GNU make
- (Conditional on setup method) Version control software git

The THOR team has good experiences with setting up your environment following the MOOSE setup instructions.

2. Getting Started

2.1 Accessing THOR on github

THOR is hosted at North Carolina State University's github repository. Open a browser and navigate to:

```
github.ncsu.edu
```

Log in with your unity ID and password. For accessing THOR you have to be a member of the THOR project. Please contact the code owner Yousry Azmy to be added to the project's membership. Once you have obtained access to THOR, click on the THOR link on THOR's github page, then click fork, and then your username; below we refer to this username as `<git_usr>`. This creates your own personal THOR repository that is separate from the main repository. You have write access to this repository, while you most likely do not have write access to the main repository.

2.2 Cloning THOR repository from github

This subsection describes how to clone, i.e. copy a fresh version, of THOR from the github repository to your local computer. The first step is to set up ssh keys. This can be accomplished by following the directions provided here.

Now, the process of cloning is described. First open a terminal window on your local computer where you wish to clone THOR. In the following description terminal commands are indicated by `>>`; by `/home/<usr>` the home directory is indicated but it is understood that `<usr>` must be replaced by the actual user name on the local computer; this tutorial also assumes that `git` is installed on the local computer and is accessible to `<usr>`. In addition, we recognize that username on the local computer, `<usr>`, can be different from the same user's github username, `<git_usr>`, hence they are distinguished in the following instructions by different notation. It is a good idea to create a folder for all github projects, e.g. by

```
>>  cd /home/<usr> ; mkdir projects
```

Navigate to the `projects` directory.

```
>>  cd projects
```

Clone THOR by typing:

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

Alternatively, a user who will not communicate frequently with THOR's github repository can avoid establishing ssh keys and clone THOR directly by issuing the following line-command on the local computer:

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

Now, navigate into the THOR directory and check if things are properly set up.

```
>> cd THOR
```

First check the current branch:

```
>> git branch
```

It should return

```
>>  * devel
```

indicating that `devel` is the current branch. The `devel` branch (short for development) contains the most up to date version of THOR. The current branch can be changed by:

```
>> git checkout <branch>
```

where `<branch>` is the branch name to switch to. Next, the remotes are set up. The remotes are addresses to remote repositories and serve as shorthand when information is pulled from or pushed to one of the remotes. The convention is to call the remote of the master repository *upstream*, and to call the remote of the personal repository *origin*. To check the remotes, type:

```
>> git remote -v
```

This should show the following:

```
>> origin git@github.ncsu.edu:NCSU-Rad-Transport/THOR.git (fetch)
>> origin git@github.ncsu.edu:NCSU-Rad-Transport/THOR.git (push)
```

To set up the remote according to the THOR convention, type:

```
>> git remote rm origin
>> git remote add upstream git@github.ncsu.edu:NCSU-Rad-Transport/THOR.git
>> git remote add origin  git@github.ncsu.edu:<git_usr>/THOR.git
```

Checking the remote again should show:

```
>> upstream git@github.ncsu.edu:NCSU-Rad-Transport/THOR.git (fetch)
>> upstream git@github.ncsu.edu:NCSU-Rad-Transport/THOR.git (push)
>> origin git@github.ncsu.edu:<git_usr>/THOR.git (fetch)
>> origin git@github.ncsu.edu:<git_usr>/THOR.git (push)
```

2.3 Updating the devel branch

As THOR is developed, the local devel branch or any other user-created branches will become outdated. This section demonstrates how to obtain the most up-to-date version of THOR. It is a good idea to check the current status of the repository, by navigating to the THOR directory and typing:

```
>> git status
```

This command reveals if there are any modified or uncommitted files. Updating the current branch is prohibited if there are any modified files. Let us first assume that there is a modified file called `/path/to/modified_file.txt`. There are two options:

- You can stash the file by:

  ```
  >> git stash
  ```

  This removes the modification and stores it in the stash. To un-stash the modifications, do:

  ```
  >> git stash pop
  ```

- You can commit the file by:

  ```
  >> git add /path/to/modified_file.txt
  >> git commit -m "A message for this commit"
  ```

After either committing or stashing, the branch can be updated by:

```
>> git pull --rebase upstream devel
```

Updating can lead to merge conflicts when local changes conflict with changes in the upstream version of the devel branch. Conflict resolution is beyond the scope of this primer. Please consult git literature or google for guides on conflict resolution.

2.4 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 after updating as described in Sect. 2.3, 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.

2.5 Compiling THOR

This section describes how to compile THOR and its dependencies. If THOR is not set up from github, then this is your entry point for the *Getting started* tutorial. Simply unzip the THOR directory where you want it to reside; this tutorial assumes that THOR is unzipped in the /home/<usr>/projects directory.

The first step is to compile the lapack dependency. To this end, navigate to:

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

Edit the file make.inc to specify the MPI Fortran compiler available on the local machine. 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%)
```

Now, THOR can be compiled. Navigate to the THOR source folder:

```
>> cd /home/<usr>/projects/THOR/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. 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 /home/<usr>/projects/THOR/THOR/
```

that should produce:

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

2.6 Running THOR for the first time

Navigate to the `hello_thor` directory:

```
>> cd /home/<usr>/projects/THOR/THOR/hello_world
```

Check the content of this folder:

```
>> ls
```

It should show the following files:

```
>> ls -l
total 696
-rwxrwxr-x 1 azmyyy azmyyy    603 Jun 26 19:56 hello_world.in
-rw-rw-r-- 1 azmyyy azmyyy  42700 Jun 26 19:56 hello_world.o
-rwxrwxr-x 1 azmyyy azmyyy 150040 Jun 26 19:56 hello_world.thrm
-rwxrwxr-x 1 azmyyy azmyyy     23 Jun 26 19:56 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 at a minimum with the executable name and the standard input file that is specified after the `-i` modifier.

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

For parallel execution type:

```
>> mpiexec -n <n> ../thor-1.0.exe -i 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_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
--------------------------------------------------------
```

2.7 Pre/post Processors

2.7.1 Setting up THOR_MESH_Generator

THOR mesh generator converts *exodus* and *gmsh* mesh formats to THOR's native mesh format. It also permits uniform refinement of meshes provided in exodus files. Conversion from *exodus* format and uniform refinement uses the *libmesh* [5] *meshtool*. Therefore, *libmesh* has to be set up first. To this end, navigate to the `scripts` directory:

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

and execute `build_libmesh.sh`:

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

where the first command makes **build_libmesh.sh** executable (if it is not already) and <n> is the number of processors. It must be provided even if it is simply 1. Executing this script may take a long time to complete installing *libmesh*, however, it will show progress on the screen. If the git-clone command in the build_libmesh.sh script does not work, replace it with the command:

```
git clone https://github.com/libMesh/libmesh.git
```

Finally, the THOR_LIBMESH_DIRECTORY environment variable has to be set. This environment variable must point to the directory that the meshtool-opt executable is located. For the standard installation, one should execute:

```
>> export THOR_LIBMESH_DIRECTORY=/home/<usr>/projects/THOR/contrib/libmesh/build
```

The next step is to make the THOR_MESH_GENERATOR application. Navigate to its source folder:

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

and type:

```
>> make
```

The exectutable

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

should have been created.

```
**************************************************************
The tests as described below did not execute as described.
Instead, I did the following and still execution of the tests
did not work properly:

1. In ~/PROJECTS/THOR/pre-processors/THOR_Mesh_Generator:
ln -s Thor_Mesh_Generator\_MP.exe Thor_Mesh_Generator.exe

2. In ~/PROJECTS/THOR/pre-processors/THOR\_Mesh_Generator/scripts:
chmode u+x test\_all.sh

3. ./test\_all.sh
This ran but did not give the output below and reported execution
errors. It is not clear if these reported errors are part of the
testing since some of the cases are labeled Bad, or the error
indicates erroneous installation of libmesh.
**************************************************************
```

To ensure that the THOR_MESH_GENERATOR application compiled correctly, execute the regression tests. Change directory to:

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

and execute:

```
>> python run_thor_tests.py
```

You should see screen output similar to this:

```
--------------------------------------------------------------------------------
Test  1 tests/bad_gmesh_non_tet_element:bad_gmesh_non_tet_element success
Test  2 tests/bad_gmesh_no_elements_block:bad_gmesh_no_elements_block success
Test  3 tests/homogeneous_domain:homogeneous success
Test  4 tests/homogeneous_domain:homogeneous_from_exodus success
Test  5 tests/homogeneous_domain:homogeneous_r1 success
Test  6 tests/homogeneous_domain:homogeneous_from_exodus_r1 success
Test  7 tests/bad_gmesh_no_nodes_block:bad_gmesh_no_nodes_block success
Test  8 tests/bad_gmesh_no_format_block:bad_gmesh_no_format_block success
Test  9 tests/bad_gmesh_non_tri_face:bad_gmesh_non_tri_face success
Test  10 tests/convert_old_to_new_THOR:convert_old_to_new_THOR success
Test  11 tests/unv_sphere_in_shell_in_box:unv_sphere_in_shell_in_box success
Test  12 tests/Basic_Cube_Mesh_test:basic_cube_mesh success
Test  13 tests/split_hex_and_prism:split_hex success
Test  14 tests/split_hex_and_prism:split_prism success
Test  15 tests/bad_unv_sphere_in_shell_in_box_no_2411:bad_unv_sphere_in_shell_in_box_no_2411 success
--------------------------------------------------------------------------------
Successes:  15            Failures:  0
```

All or at least the vast majority of tests should pass, so `Failures` should be close to zero.

3. 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 8.71 cm discretized by tetrahedra similar to Fig. 3.1 (with the exception that Fig. 3.1 shows on-eighth of the domain). The energy domain is discretized with six energy groups, and cross sections are provided by [1].

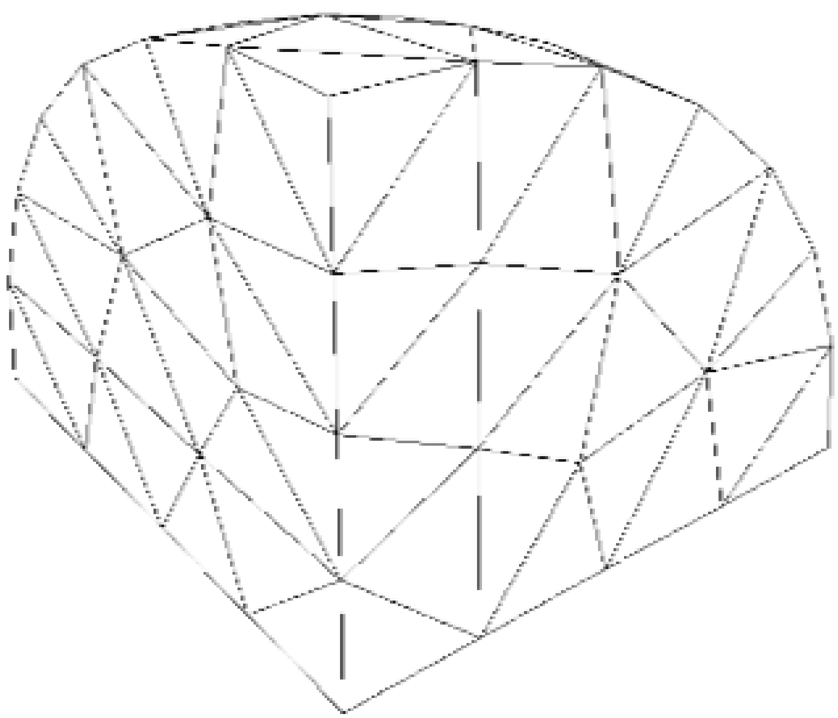


Figure 3.1: Coarse mesh for Godiva problem, picture courtesy [3]

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:

```
>> /home/<usr>/projects/THOR/THOR/examples/Godiva_tutorial
```

3.0.1 Godiva Mesh

The workflow described here is suitable if the user has access to the Cubit mesh generator. A Cubit journal file is provided in directory:

```
>> /home/<usr>/projects/THOR/THOR/examples/Godiva_tutorial
```

It creates the exodus file `godiva.e`. To verify whether Cubit is available to the user on the target computer execute the command line:

```
>> which cubit
```

Note that even if Cubit is installed on the target computer it might not be available to the user unless its path is defined in the user's search paths. To execute the journal file, the last line must be modified by replacing `<path>` with:

```
/home/<usr>/projects/THOR/THOR/examples/Godiva_tutorial ..
/create_mesh/from_cubit
```

then execute Cubit with the command line:

```
>> cubit <godiva_mesh_CUBIT.jou
```

For users that do not have access to Cubit, the `godiva.e` file is provided in the `from_cubit` directory. The exodus file `godiva.e` is converted to THOR's native mesh format by executing the THOR mesh generator using the standard input file `convert_godiva.in` that is included in the `from_cubit` directory with the command line:

```
>> /home/<usr>/projects/THOR/pre-processors/THOR_Mesh_Generator/Thor_Mesh_Generator_MP.exe -i conve
```

In this case this input file contains the following lines:

```
./godiva_1_c.e
./godiva_1_c.thrm
```

specifying the input exodus file and the THOR mesh formatted output file. The mesh conversion infers the file format of the input by the filename extension; currently `.e` and `.gmsh` for exodus and gmesh formats, respectively, are supported. The conversion is performed by typing:

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

After successful completion of the conversion, the following printout should appear:

```
============================================================

   TTTTTTT  HH     HH  OOOOO  RRRRRR
     TTT    HH     HH OOOOOOO RRRRRRR
     TTT    HH     HH OO   OO RR   RR
     TTT    HHHHHHHHH OO   OO RRRRRR
     TTT    HHHHHHHHH OO   OO RRRR
     TTT    HH     HH OO   OO RR RR
     TTT    HH     HH OOOOOOO RR  RR
     TTT    HH     HH  OOOOO  RR   RR

   Tetrahedral High Order Radiation Transport Code

   By North Carolina State University

   Version 1.0 - Update 2020
============================================================

============================================================
Executing -->THOR_MESH_GENERATOR<-- application
============================================================

------------------------------------ Input ------------------------------------
Infile mesh file name: ./godiva_1_c.msh
Output mesh file name: ./godiva_1_c.thrm
Mesh refinement level:    0

No region ID edits are provided
No source ID edits are provided
No boundary ID edits are provided


============================================================
Application -->THOR_MESH_GENERATOR<-- terminated successfully
============================================================
```

The file `godiva_1_c.thrm` should result from this execution for use by THOR. This concludes the mesh generation step for this tutorial.

3.0.2 Cross section data

The THOR cross section file for the Godiva benchmark is provided by:

`/home/<usr>/projects/THOR/THOR/examples/Godiva_tutorial/cross_sections/godiva.xs`

THOR uses a custom cross section format that is explained in detail in Sec. 4.0.1.

3.0.3 THOR input file and executing THOR

The THOR input file is

```
/home/<usr>/projects/THOR/THOR/examples/Godiva_tutorial/...
THOR/godiva.i
```

THOR uses a keyword-based input that is listed in Sect. 4.0.1. The Godiva tutorial input file is now explained in detail; note that this listing includes comments separated by # at the end of each line that are not present in the original input file.

```
Godiva 6 group example               # title line

  start problem                      # beginning of problem block
    execution = yes                  # problem is solved
    lambda = 0                       # constant approximation (SC)
    type = keig                      # solve an eigenvalue problem
    keigsolver = pi ; piacc = none   # Power iterations w/o
                                     # acceleration, note separation
                                     # by ; to have same-line synatx
    sweep = precomp                  # sweep path precomputed
    page_refl = save                 # reflected bnd fluxes stored
    kconv = 1E-8                     # stopping tolerance on k-eff
    innerconv = 1E-12               # stopping tolerance inner it
    outerconv = 1E-7                 # stopping tolerance outer it
    maxinner = 4 ; maxouter = 5000   # max # inner/outer it
  end problem                        # terminate the problem block

  start inout                        # start input/output file block
    mesh_file =  ../create_mesh/from_CUBIT/godiva_1_c.thrm  # mesh file
    xs_file =  ../cross_sections/godiva.xs  # XS file
    flux_file = godiva.flux          # plain flux output file
    vtk = flux mat                   # print flux and mat ids to vtk
    print_xs = yes                   # XS are echoed
  end inout.                         # terminate this block

  start cross_sections
     ngroups = 6                     # the number of energy groups
     upscattering = no               # upscattering is not present
                                     # scat. XS is lower triangular
  end cross_sections

  start quadrature
    qdtype = levelsym                # level-symmetric quadrature
    qdorder = 4                      # order 4
  end quadrature

  start regionmap                    # the region map maps block
     1                               # ids to materials; in this
                                     # case the only block is
  end regionmap                      # assigned material 1 that is
                                     # present in XS file

end file                             # the input file is terminated
                                     # by "end file"
```

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

```
>> /home/<usr>/projects/THOR/THOR/thor-1.0.exe -i godiva.i
```

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 0.9611. This is not close to critical because the mesh that is created does not conserve the volume of the critical sphere. When creating a tetrahedral mesh, Cubit places the nodes on the sphere’s surface, but that necessarily leads to the volume of the discretized geometry to be smaller than the original volume. In this case, the computational volume is 2742.4 cm^3, while the actual volume is 2767.8 cm^3.
- 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, `flux.vtk` contains spatial flux maps, and `mat.vtk` contains the material map. These files can be opened with the paraview post-processing tool that is available here.

The reaction rate summary is given by:

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

-- Region --    1 Volume=   2.742401E+03

   Group           Flux        Fission     Absorption      Fiss Src
       1  8.424948E-01  1.399140E-01  4.946487E-02  1.399140E-01
       2  1.573081E+00  2.333728E-01  9.509448E-02  2.333728E-01
       3  9.814645E-01  1.379655E-01  5.968679E-02  1.379655E-01
       4  1.631588E+00  2.215631E-01  1.006868E-01  2.215631E-01
       5  1.198341E+00  1.915944E-01  9.089592E-02  1.915944E-01
       6  1.792724E-01  4.652155E-02  2.413688E-02  4.652155E-02
Total     6.406241E+00  9.709313E-01  4.199657E-01  9.709313E-01
```

The results can be improved greatly by enforcing volume conservation on the Cubit mesh. This is accomplished manually in this tutorial by changing the radius in the Cubit journal file to .

4. Input Format

4.0.1 THOR Transport Solver

add a description of input format for

- standard input
- XS
- mesh
- source

This section describes the input format of the THOR transport solver.

4.0.2 Compilation and Invocation

Navigate to:

```
>> cd /home/<usr>/projects/THOR/THOR/src
```

and make the application:

```
>> make
```

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

```
>> /home/<usr>/projects/THOR/THOR/THOR_TRANSPORT_SOLVER.exe
```

The THOR transport solver is invoked by

```
>> mpiexec -n <n> THOR_TRANSPORT_SOLVER.exe -i standard_input -t
```

where `-i` precedes the name of the standard input file `standard_input` and `-t` is an optional parameter for additional timing. **Remark:** `-i standard_input` must currently follow the executable name `THOR_TRANSPORT_SOLVER.exe`. This will be fixed in future versions.

4.0.3 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. 4.0.4). Cross section ids are indices that identify sets of cross sections provided in the cross section input file (see Sect. 4.0.5).

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.

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 Newton
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 informatio
`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 ir
`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, Leger
`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

4.0.4 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.

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.

4.0.5 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.

4.0.6 List of all Inputs and Outputs of THOR transport solver

4.0.7 THOR Mesh Generator

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

- Convert exodus [6] formatted meshes to THOR format.
- Convert gmsh [4] 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.

4.0.7.1 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 [5]. You must compile libMesh and set the environment variable THOR_LIBMESH_DIRECTORY.

4.0.7.2 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. 4.0.7.3.
- source_id_file: name of file that contains instructions to reassign source ids. Formatting instructions for this file is provided in Sect. 4.0.7.3.
- boundary_id_file: name of file that contains instructions to reassign boundary ids. Formatting instructions for this file is provided in Sect. 4.0.7.3.

4.0.7.3 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.

4.0.8 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] R. M. Ferrer. *An Arbitrarily High Order Transport Method of the Characteristic Type for Unstructured Tetrahedral Grids*. PhD thesis, The Pennsylvania State University, Department of Mechanical and Nuclear Engineering, 2010.

[4] 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.

[5] 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.

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