Sandia
National
Laboratories

SIERRA Multimechanics Module: PMR User Manual – Version 5.18

Sierra Thermal Fluids Team

Contents

List of Figures

List of Tables

This page intentionally left blank.

1 Overview

PMR is a Multiple Program Multiple Data (MPMD) participating media radiation solver for calculating radiative transport in conjunction with a separate fluid solver (commonly Sierra/Fuego). Its primary use is for fire simulations, where resolving radiative fluxes is of high importance.
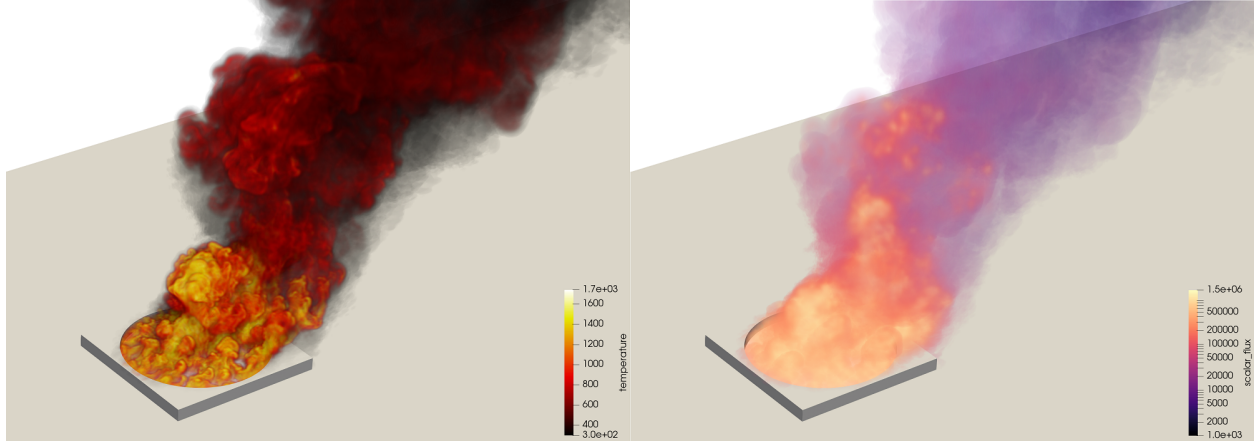
Fig. 1.1: Example Fuego/PMR fire simulation with temperature (left) and scalar flux (right)

2 Theory

PMR solves the radiative transport equation (RTE) using the method of discrete ordinates with user-selectable quadrature rules. The quadrature rule and order defines a set of discrete directions in which intensity is solved. Scattering is not currently supported, so this results in a decoupled set of linear PDEs in each ordinate direction.

For each ordinate direction ($\vec{s}_k$), a weight is provided (w_k) by the quadrature rule selected. For each ordinate direction, the ordinate intensity I_k is solved for and overall scalar flux (G) and radiative heat flux ($\vec{q}$) can then be defined by summing over ordinate directions as

$$G = \sum_k I_k w_k$$

and

$$\vec{q} = \sum_k I_k \vec{s}_k w_k$$

The continuous form of the PDE evaluated for intensity (excluding scattering terms) in a given ordinate direction is

$$\vec{s}_k \cdot \nabla I_k = S_I - \mu_a I_k$$

where the radiative source term (S_I) and absorption coefficient (μ_a) are calculated by the fluid code and transferred to PMR. The radiative source term is usually similar to the form below, but may include subgrid effects where relevant:

$$S_I \simeq \frac{\mu_a \sigma T^4}{\pi}$$

On the domain boundaries, the following condition is enforced for intensity

$$I_b = \frac{B_{src} + (1 - \epsilon - \tau) H}{\pi}$$

where H is the irradiation (sum of incoming intensities on the boundary) and the boundary source (B_{src}) is sent from the fluids code and usually has a form similar to the following equation:

$$B_{src} = \sigma \left(\epsilon T_{bc}^4 + \tau T_{env}^4 \right)$$

In the case where $\epsilon + \tau < 1$ the boundary is partially reflective and the problem becomes nonlinear and coupled across ordinate directions through H. When this is the case, multiple nonlinear iterations should be used to achieve a converged solution. The irradiation is calculated on all boundaries as

$$H = \sum_k \begin{array}{ll} w_k I_k \vec{n} \cdot \vec{s_k} & \text{if } \vec{n} \cdot \vec{s_k} > 0 \\ 0 & \text{if } \vec{n} \cdot \vec{s_k} \leq 0 \end{array}$$

where $\vec{n}$ is the outward facing normal vector.

The primary numerical choices when solving this equation are what type of stabilization to use for the advection-like term (upwind or SUCV) and whether to use an edge-based or element-based stencil.

2.1 Quadrature Rules

Thurgood

The Thurgood [1] quadrature rule partitions space into $8N^2$ ordinate directions, where N is the specified quadrature order. The order must be at least 2 for this quadrature rule.

Level-Symmetric

The Level-Symmetric [2] quadrature rule partitions space into $N(N + 2)$ ordinate directions, where N is the specified quadrature order. The order must be one of: 2, 4, 6, 8, 12, or 16. No other orders are valid for this quadrature rule.

PNTN

The Legendre-Chebyshev (PN-TN) [3] quadrature rule partitions space into $N(N+2)$ ordinate directions, where N is the specified quadrature order. The order must be even for this quadrature rule.

3 Input File

PMR supports two input file formats (both YAML) - the legacy Nalu input files and the new more compact PMR syntax. We recommend using the PMR syntax for any new or modified cases, so this format is the only one described below.

> **ⓘ** **Note:** PMR uses a YAML input file, which means that commands are case-sensitive and indentation matters. Use spaces for indentation, not tabs.

3.1 Linear Solver

The linear solver will default to gmres with and sgs preconditioner if not specified. To change any of the arguments, add a `linear_solver` block (shown below) with the relevant line commands. Any line commands not included will use the default values.

```
linear_solver:
  method: gmres
  preconditioner: sgs
  num_sweeps: 3
  damping_factor: 0.7
  tolerance: 1e-5
  max_iterations: 100
  kspace: 100
  output_level: 0
```

Supported solver options include (but are not limited to) the values in the table below:

Table 3.1: Solver options

Option	Valid Parameters
method	`gmres bicgstab`
preconditioner	`jacobi sgs sgs2`
num_sweeps	Integer number of preconditioner sweeps
tolerance	Linear solver tolerance
damping_factor	Preconditioner damping factor (scalar from 0 to 1)
max_iterations	Maximum number of linear solver iterations
kspace	K-space (restart) iterations for solver
output_level	Integer for output verbosity (higher = more verbose)

Note: Nalu did not report solver failures, so they were silently ignored. The prior default solver used in Nalu used 1 sweep and no damping, which can struggle to converge for some PMR problems. Increasing to 3 sweeps and setting a damping factor of 0.7 can improve the solver robustness. The two-stage SGS preconditioner (sgs2) is an alternative to SGS meant for better performance on GPU architectures. SGS does not work with batching and does not perform well on GPU systems.

3.2 Simulation Settings

The simulation settings block is the only required block in the PMR input file. This is where you specify the name of the mesh to use (required), decomposition method (optional), the advection operator method (required), the quadrature type (required), and nonlinear settings (optional).

```
simulation_settings:
  mesh: pmr.g
  decomposition_method: rcb
  method: edge_upwind

  nonlinear_settings:
    max_iterations: 1
    convergence_tolerance: 1e-6

  quadrature:
    type: thurgood
    order: 4
```

Solution Method

The following solution methods are supported for the `method:` argument:

- `edge_upwind` (or `edge upwind` or `edge_upw`) - Use an edge-based discretization with upwind stabilization (faster, first order).

- `edge_sucv` (or `edge sucv`) - Use an edge-based discretization with SUCV stabilization (second order).

- `elem_sucv` (or `elem sucv`) - Use an element-based discretization with SUCV stabilization (second order). Using this option allows linear system rotation so the linear system does not need to be re-assembled for each ordinate direction.

Quadrature Type

The supported quadrature types are listed below. All types except for user-defined require an `order` parameter as well as the type.

- `thurgood` - Uses the Thurgood triangulation-based quadrature with $8N^2$ ordinate directions

- `levelsymmetric` (or `level_symmetric`) - Uses the level symmetric quadrature with $N(N + 2)$ ordinate directions. The order must be one of 2, 4, 6, 8, 12, or 16.

- `pntn` - Uses the Legendre-Chebyshev (PN-TN) quadrature rule with $N(N + 2)$ ordinate directions. The quadrature order must be even.

- `userdefined` (or `user_defined`) - Reads a CSV file of user-defined quadrature directions and weights. The CSV file should list the number of ordinate directions on the first line, and all subsequent lines must be `nx`, `ny`, `nz`, `weight`. The weights must sum to 4π. An example file is shown below. The user defined quadrature takes a file name instead of an order argument like the other types. If omitted, this defaults to "user_quadrature.csv".

```
4
1, 0, 0, 3.141
0, 1, 0, 3.141
0, 0, 1, 3.141
-1, 0, 0, 3.141
```

```
quadrature:
  type: user_defined
  file: my_quadrature.csv
```

3.3 Post-Processing

PMR supports doing integrals and averages of scalar fields on sidesets (internal or external). Each post-processor is defined with either an `integral` or `average` block, and must have a `field`, `target_name`, and `output_name` specified. The `target_name` argument can be a single sideset or a list of sidesets. The calculated quantities are sent to the fluid code, which registers them as global variables available for output.

> **(i)** **Note:** Post-processed global variables cannot be output directly by PMR, they are only sent to the calling fluid code.

```
post_process:
- integral:
    field: irradiation
    target_name: surface_1
    output_name: int_irr1
- average:
    field: irradiation_opp
    target_name: [surface_1, surface_2]
    output_name: int_irr1_opp
```

When doing a spectral solve, the fluid code determines the naming scheme for the per-band quantities. For example, when coupling with Fuego, the above example would produce `int_irr1_B0`, `int_irr1_B1`, etc as well as a band sum saved as the original name (`int_irr1`).

The field specified must be an existing field, with one exception. Since irradiation is a directionally dependent quantity, on internal sidesets you may want to calculate irradiation from either side. Post-processing `irradiation_opp` will post-process irradiation with the opposite normal convention on the requested sideset(s).

3.4 Output Settings

Typically the fields of interest are transferred to the fluid code and output there, but an optional output settings block can be included in the PMR input file to output fields from PMR directly. Only nodal fields can be output from PMR.

```
results_output:
  output_filename: pmr_out.e
  output_frequency: 2
  output_variables:
```

```
    - absorption_coefficient
    - scalar_flux
    - radiative_heat_flux
    - radiation_source
    - div_radiative_heat_flux
    - irradiation
    - emissivity
    - transmissivity
```

4 Running and Troubleshooting

4.1 Launching an MPMD Job

The command to run in MPMD mode is different from what is used to run typical MPI programs. To run a typical MPI program you simply run a parallel job using something like

```
$ mpirun -np 10 some_program
```

which launches 10 ranks of *some_program* to run on 10 CPU cores. With MPMD runs you are launching two separate MPI jobs with two different codes that can communicate. An example MPMD launch command to run PMR and Fuego together would look like

```
$ mpirun -np 10 fuego -i fuego.i : -np 10 pmr -i pmr.i
```

For some applications, the launch command should include an MPI color, used for the MPMD communication. For example, to launch PMR with Aria, you would use

```
$ mpirun -np 10 aria -i aria.i --mpmd_master --mpi_color 9999 : -np 10␣
↪pmr -i pmr.i
```

The `mpmd_master` flag indicates that Aria will be controlling the execution (the "leader") of the simulation while the other application "follows" Aria.

Note that the order of the two apps *does not* matter in an MPMD call. The `mpi_color` argument for Aria specifies an id to label the cores executing Aria with to distinguish them from the cores executing the coupled application. This can be any integer, and must be unique across the coupled apps.

By default, Fuego is always the "leader" in MPMD execution mode and uses a hard-coded color so these flags are not required when coupling to Fuego.

There is no requirement that the two codes use the same number of cores, so depending on the mesh and computational costs you may choose a different allocation per code. For example, if

your PMR solve is very expensive, you may allocate more cores to it than the fluid code:

```
$ mpirun -np 10 fuego -i fuego.i : -np 100 pmr -i pmr.i
```

HPC Execution

Special care must be taken when submitting MPMD jobs on the HPCs or any queued environment. By default, the two MPMD codes cannot share cores so to launch a case where each code is given 100 cores on an HPC you would need to request an allocation of 200 cores. This is unnecessarily wasteful though since PMR would not be using its 100 cores while the fluid code runs, and the fluid code would not be using its 100 cores while PMR runs. To get around this, you must enable oversubscription to allow the two codes to share resources. To get an allocation of 100 cores and use all the cores for both codes you must add additional mpi flags:

```
$ mpiexec --oversubscribe \
    --bind-to core:overload-allowed -np 100 fuego -i fuego.i : \
    --bind-to core:overload-allowed -np 100 pmr -i pmr.i
```

Keep in mind that the specific command to use can be platform dependent. A more complete example submission script on an HPC may look like

```
#!/bin/bash

#SBATCH --nodes=10
#SBATCH --time=48:00:00
#SBATCH --account=PUT_YOUR_WCID_HERE
#SBATCH --job-name=pmr
#SBATCH --partition=batch

nodes=$SLURM_JOB_NUM_NODES
cores=36

module load sierra
export OMPI_MCA_rmaps_base_oversubscribe=1
mpiexec --oversubscribe                                        \
  --bind-to core:overload-allowed                              \
  --npernode $cores --n $(($cores*$nodes)) fuego -i fuego.i :  \
  --bind-to core:overload-allowed                              \
  --npernode $cores --n $(($cores*$nodes)) pmr -i pmr.i
```

Contact sierra-help@sandia.gov if you need more help or encounter issues running MPMD jobs.

4.2 Mesh Requirements

The PMR mesh will apply a single radiative boundary condition to all exposed faces in the mesh automatically, and solve the RTE equations on all elements. This means that you do not need to provide sidesets in the mesh (if present, they will be ignored). Internal sidesets can be defined for post-processing and will not affect the RTE solver domain or transferred boundaries.

4.3 Troubleshooting

For help using PMR, visit the TF Teams channel, or submit a help request at the CompSim Help Portal or by emailing sierra-help@sandia.gov. Be sure to include relevant information like log files, input files, the version you are running, and what commands you used.

5 Symbols

Table 5.1: List Of Symbols

Symbol	Description	First Appearance
B_{src}	Boundary intensity source	*Theory*
G	Scalar Flux	*Theory*
H	Boundary irradiation	*Theory*
I	Intensity	*Theory*
I_b	Boundary intensity	*Theory*
I_k	Intensity for ordinate direction k	*Theory*
S_I	Radiation source term	*Theory*
ϵ	Surface emissivity	*Theory*
μ_a	Absorption coefficient	*Theory*
σ	Stefan Boltzmann constant	*Theory*
τ	Surface transmissivity	*Theory*
$\vec{s}_k$	Ordinate direction vector k	*Theory*
w_k	Weight for ordinate direction k	*Theory*

This is the web-based PMR user manual (Version 5.18). Use the navigation on the left to explore different topics, or use the search to quickly find content. The structure of the manual is as follows:

Overview

An overview of the PMR solver and common applications

Theory

Details of the theory behind the RTE solve in PMR and the different quadrature rules and solution methods available

Input File

Details about how to set up an input file for PMR

Running and Troubleshooting

Details about how to launch an MPMD simulation and troubleshooting steps.

Note that some sections may be incomplete since this is work in progress. If you have any feedback, please contact the TF team on Teams or send an email to sierra-help@sandia.gov.

References

[1] C. P. Thurgood, A. Pollard, and H. A. Becker. The tn quadrature set for the discrete ordinates method. *ASME Journal of Heat Transfer*, 117(4):1068–1070, 1995.

[2] E. E. Lewis and W. F. Miller. Computational methods of neutron aq8 608 transport. *American Nuclear Society*, 1993.

[3] G. Longoni. *Advanced Quadrature Sets, Acceleration and Preconditioning Techniques for the Discrete Ordinates Method in Parallel Computing Environments*. PhD thesis, The University of Florida, Gainesville, FL, 2004.