NREL
NATIONAL RENEWABLE ENERGY LABORATORY

IHT: Tools for Computing Insolation Absorption by Particle Laden Flows

R. W. Grout

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC.

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Technical Report
NREL/TP-2C00-60467
October 2013

Contract No. DE-AC36-08GO28308

NREL
NATIONAL RENEWABLE ENERGY LABORATORY

IHT: Tools for Computing Insolation Absorption by Particle Laden Flows

R. W. Grout

Prepared under Task No. ER10.1000

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC.

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

Technical Report
NREL/TP-2C00-60467
October 2013

Contract No. DE-AC36-08GO28308

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Available electronically at http://www.osti.gov/bridge

Available for a processing fee to U.S. Department of Energy and its contractors, in paper, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
phone: 865.576.8401
fax: 865.576.5728
email: mailto:reports@adonis.osti.gov

Available for sale to the public, in paper, from:

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
phone: 800.553.6847
fax: 703.605.6900
email: orders@ntis.fedworld.gov
online ordering: http://www.ntis.gov/help/ordermethods.aspx

Cover Photos: (left to right) photo by Pat Corkery, NREL 16416, photo from SunEdison, NREL 17423, photo by Pat Corkery, NREL 16560, photo by Dennis Schroeder, NREL 17613, photo by Dean Armstrong, NREL 17436, photo by Pat Corkery, NREL 17721.

Printed on paper containing at least 50% wastepaper, including 10% post consumer waste.

Contents

List of Figures

List of Tables

1 Introduction

This report describes *IHT*, a toolkit for computing radiative heat exchange between particles. Well suited for insolation absorption computations, it is also has potential applications in combustion (sooting flames), biomass gasification processes and similar processes. The algorithm is based on the the "Photon Monte Carlo" approach described by Wang and Modest [6] and implemented in a library that can be interfaced with a variety of CFD codes to analyze radiative heat transfer in particle-laden flows. The implementation is in a library where MPI-based parallelism is used for performance on modern HPC resources. The implementation is tailored to particle-laden flows where the particles are distributed between parallel processes according to a Cartesian decomposition of the physical domain. It is initially coupled to the S3D code [2]. S3D solves the compressible Navier-Stokes equations coupled with detailed transport and chemical reaction mechanisms and is typically used for research on turbulent combustion. The emphasis in this report is on the data structures and organization of IHT for developers seeking to use the IHT toolkit to add Photon Monte Carlo capabilities to their own codes.

2 Implementation and Algorithmic Details

2.1 Formulation

The radiative transfer equation is a spatiotemporal integro-differential equation depending on space, time, wavenumber and direction [5]:

$$\frac{dI_\eta}{ds} = k_\eta I_{b\eta} - k_\eta I_\eta - \sigma_{s\eta} I_\eta + \frac{\sigma_{s\eta}}{4\pi} \int_{4\pi} I_\eta(\hat{s}_i)\Phi_\eta(\hat{s}_i,\hat{s})d\Omega, \tag{2.1}$$

where the terms represent, respectively, the change of spectral radiative intensity along a light of sight in the direction $\hat{s}$ due to: emission, absorption, scattering away from $\hat{s}$ and scattering into direction $\hat{s}$ from other directions. The subscript η is the wavenumber; in the most general case the emission/absorption (κ_η) coefficient is frequency dependent, and the scattering coefficient ($\sigma_{s\eta}$) is both direction and frequency dependent. The function $\Phi_\eta(\hat{s}_i,\hat{s})$ is the scattering phase function, supplying the probability that a ray coming from $\hat{s}_i$ is scattered into the direction $\hat{s}$. Wang and Modest [6] solve this equation with the aid of a discrete particle method where the medium (not necessarily comprised of physical particles) is represented by a collection of virtual particles. With a slight modification of the emissions function, we use a similar formulation to solve the radiative transfer problem for a physical particle field. Given the physical nature of the particles that have volume, we treat the photon bundles using line rays for intersection tests. Where the intra-particle medium is non-participating, the radiative transfer equation between the particles reduces to:

$$\frac{dI_\eta}{ds} = 0 \qquad \Rightarrow I_\eta(\hat{s}) = \text{constant}. \tag{2.2}$$

In the geometric limit with opaque particles, the method consists of emitting $N_{r,i}$ rays from the i^{th} particle and then finding the intersection of these rays with the nearest particle that the ray intersects to compute an interaction (absorption or reflection), and continuing until all rays have exited the domain or the domain reaches thermal equilibrium. In the current version, IHT neglects all Lorenz-Mie scattering, including neglecting scattering in the Rayleigh limit, although the framework provides a solid foundation to implement such capability.

Photon bundles are characterized in IHT by an origin $\vec{O}_i$, direction $\vec{d}_i$, total energy Q_i and, optionally, spectral characteristics. Each particle emits photon bundles with energy based on its local temperature:

$$Q_i = \varepsilon_i \sigma T_i^4 A_i. \tag{2.3}$$

As suggested by Modest [5], if the energy Q_i emitted from a single particle significantly exceeds the average Q_{avg}it is divided equally amongst $N_{r,i}$ rays, where:

$$N_{r,i} = \text{floor}\left(\frac{Q_i}{Q_{\text{avg}}} + 0.5\right), \tag{2.4}$$

and, with total number of rays N_R emitted from N_p particles, the average is:

$$Q_{\text{avg}} = \frac{1}{N_R}\sum_{i=1}^{N_p} Q_i \tag{2.5}$$

and then the energy for the k^{th} ray from the i^{th} particle is:

$$Q_{i,k} = \frac{Q_i}{N_{r,i}}. \tag{2.6}$$

In the default mode, all of the particles are considered gray, and the spectral emissions/absorption functions are uniform. However, the spectral dependencies can be accounted for by assigning the ray wavelength by rejection sampling against the black body distribution, that is:

$$\xi_\lambda = \frac{1}{\varepsilon\sigma T^4}\int_0^\lambda \varepsilon_\lambda E_{b\lambda} d\lambda, \tag{2.7}$$

where ξ_λ is a uniformly distributed (pseudo-)random number and:

$$E_{b\lambda}(T,\lambda) = \frac{2\pi hc_0^2}{n^2\lambda^5\left[e^{hc_0/(n\lambda kT)}-1\right]} \tag{2.8}$$

is the blackbody emissive power spectrum [5].

The particles are assumed to fully absorb incident rays and emit rays isotropically. Isotropic emission consists of emission that is uniform with regard to the spherical angle Ω; the azimuthal ψ and polar angles ϕ are sampled according to:

$$\psi^* = 2\pi\xi_\psi \qquad \theta^* = \cos^{-1}(2\xi_\theta - 1)\,. \tag{2.9}$$

This leads to the directional cosines:

$$n_z^* = \sin(\theta^*)\cos(\psi^*) \qquad n_y^* = \sin(\theta^*)\sin(\psi^*) \qquad n_x = \cos(\theta^*). \tag{2.10}$$

The pseudo-random numbers necessary for spawning the rays are obtained using the Mersenne Twister algorithm [4]; currently the implementation of the MT algorithm that allows splitting the stream into several independents streams by Ishikawa [3] is used to facilitate parallelization.

2.2 Library structure

The library is divided into two major sections: the core ray tracing utilities (contained in the module *part_radht_m*), and a set of interface routines for interaction with the CFD program (contained in the module *part_radht_interface_m*). Core datastructures to hold the rays are contained in *ray_types_m* which is referenced by both of the above modules as well as the driver module. Additionally, the module *kd_tree_m* contains utilities for building a kd tree from unsorted particle data.

The library is arranged into the source files listed in Table 2.

File	Contents
accumulator_m.f90	Datastructures and utilities for round-off corrected accumulators.
geom_tools_m.f90	Tools for geometric operations (e.g., intersection testing)
kd_tree_m.f90	Tools and datastructures for kD trees
mts_m.f90	Wrapper for *mt_stream*
part_radht_driver_m.f90	Main driver routine and interface
part_radht_interface_m.f90	Module for interacting with CFD code
part_radht_m.f90	Main raytracing routines
ray_types_m.f90	Datastructures for storing lists of rays
timer_m.f90	Routines for performance timing

Table 2. Source file list

2.3 Data structures

Two primary data structures are used to store rays as an "array of structures". The *raylist_t* described in Datastructure 2 manages a list of rays described by the properties in Datastructure 1. In the version of the code current as of this report, these data structures are accessed directly, although the code is structured such that it would be trivial to write accessor functions to facilitate changing to a "structure of arrays".

Datastructure 1 TYPE :: rayprop_t

real*8	origin(3)	*Ray global origin*
real*8	dir(3)	*Ray direction origin*
real*8	Tsrc	*Ray source temperature*
real*8	e	*Energy carried by ray*
integer	incident	*Flag for incident insolation ray*
real*8	l_start(3)	*Origin/entry on local rank*
real*8	l_stop(3)	*Termination/exit on local rank*
integer	nextdest	*Rank of next process ray should be passed to*

Datastructure 2 TYPE :: raylist_t

rayprop_t	prop(:)	*Array of ray property structures*
integer	fill	*Index of highest occupied position in* prop
integer	alloc_size	*Allocated size of* prop
integer	nempty	*Quick reference for number of empty positions in* prop
integer	initialized = 0	*Set to 1 if* prop *is allocated*
integer	rstatus(:)	*Array of status for corresponding to* rprop *(ACTIVE, ABSORBED, EMPTY)*

2.4 Parallel ray tracing

The ray tracing portion of the library identifies the closest interacting particle to the ray origin. The objective is as depicted in Figure 1. Given a ray, R_j, described by its direction $\vec{d}$ and point of origin $\vec{O}$, determine which of the n_p particles at points p_i that $\vec{R}_j$ interacts with. An interaction is defined by the closest intersection between R_j and a sphere of radius r_{eff} centered at p_i. We adopt a combined ray-tracing parallelism approach for the intersection tests. Each ray R_j is initially assigned to the process where it originates; the ray depicted in the figure is initially assigned to Rank 4. This rank checks the ray for the closest intersecting particle within the portion of the domain owned by Rank 4, which is an entirely local operation. If no intersection is found, the ray is passed to the next process along the ray (Rank 5 here) for testing against that process's local particles, and so on until either an intersection is found or the ray exits the domain (at E in this example). To aid the local queries (i.e., restrict the local particles tested for interaction to those in the bounding box defined by the particle entry and exit points for each rank), the particle locations are organized into a kD tree. The cost of building this data structure is amortized over the many rays that pass through each rank. From a global perspective the particles are organized into a two-tiered data structure, the top level being equal sized spatial bins that are distributed and the lower level being a well-balanced kD tree constructed from the particles within each top level bin. A further optimization is that the algorithm steps along the ray within each process, reducing the particles that need to be tested for intersection further. This is indicated on Rank 5 (the same number of subdivisions is used on all ranks in practice, but shown only once for clarity), where the query 'Q2' is split into 'Q2a'–'Q2c', which encompass a smaller total volume as they follow the ray more closely than 'Q2' does.

This approach reduces the complexity of the algorithm significantly from a brute force approach (that would involve $n_{rays} \times n_{particles}$ global intersection tests followed by a depth test for the candidate interacting particles) and also enables a scalable parallel solution.

The loop described in Algorithm 1 is implemented in the module *part_radht_driver_m* with the aid of the routines in *part_radht_m* to implement the raytracing. The major public routines in the latter module include:

initialize_part_radht Initialization routine

finalize_part_radht Cleanup routine

propagate_rays Loop over local ray tracing and exchange until all rays have been absorbed or left the domain

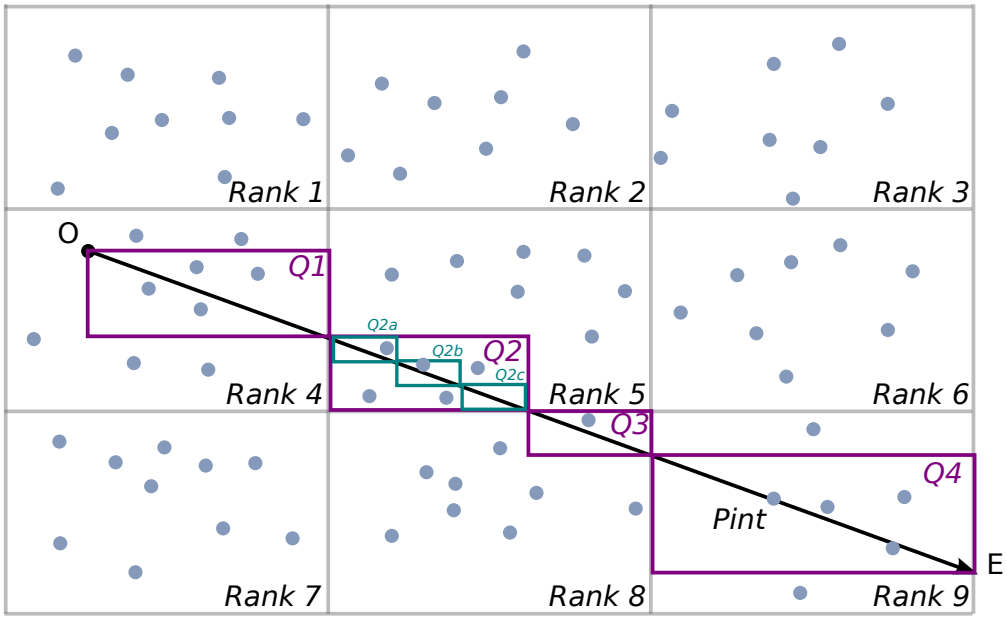

Figure 1. Intersection testing by ray tracing through domain

locally_absorb Loop over all rays and the portion of the rays between the entry/exit from the portion of the domain local to the current rank for intersections and queue for communication if no intersection

check_local_particles Check local particles for intersection with a specific ray

check_kd_intersect Check for intersection of a ray with particles inside a supplied bounding box

reloc_incident_origin Relocate rays marked as incident to the rank where their origin is local.

2.4.1 kD tree construction

As an optimization over a brute-force query within the bounding box for each step along the ray, the particles local to each MPI rank are organized into a kD-tree [1] at the start of the radiative heat transfer driver routine. While the cost of construction is significant, a list of particles within a specified bounding box can be obtained efficiently, reducing the number of applications of the intersection test. The cost of construction is amortized over all of the rays considered by a given rank so overall gains in performance are realized.

Utility routines to construct the kD tree from unsorted arrays of particle data from the simulation based on physical location are provided in the module contained in the file *kd_tree_m.f90*. The tree is stored in three data structures: an array of structures describing the tree nodes (*treelist_t*), the node structure (*treenode_t*) and a structure of arrays holding the leaf node data (*treedata_t*). At the end of the build process, the former two structures contain all of the data necessary to query the tree for a list of nodes based on a spatial bounding box, and the latter data structure contains the mapping between those nodes and the indices of the particles in the source data used to build the tree.

The major public routines are as follows:

kd_tree_init Initialization routine, currently no action other than to cache the local MPI rank id.

tracer_to_kd Main interface routine, used to convert simulation tracer data

kd_tree_build Recursive function used by *tracer_to_kd* to build the tree.

sort_treedata Utility to sort the *treedata_t* structure by kD tree node to optimize access based on a node id.

Algorithm 1 Ray absorption loop

```
1:  while active rays > 0 do
2:      for ir ← 1 to rlist%fill do                    ▷ Loop over local raylist; in locally_absorb
3:          GET_LOCAL_ORIGIN                           ▷ Implemented in get_exit_direction_new
4:          GET_LOCAL_EXIT
5:          GET_NEXT_PROCESS(nextdest)
6:          for i ← 1 to nstep_local do
7:              CHECK FOR INTERSECTION ON THIS STEP(mindist_idx)   ▷ Uses check_kd_intersect
8:              if mindist_idx > 0 then
9:                  rlist%status(ir) ← ABSORBED
10:                 ABSORB_RAY(temperature(mindist_idx),rlist%prop(ir))
11:             if rlist%status(ir) == ACTIVE then
12:                 if nextdest > 0 then
13:                     QUEUE_RAY_COMM(rlist%prop(ir))
14:                 else
15:                     terminate ray
16:     DO_RAY_EXCHANGE()
```

Datastructure 3 TYPE :: treedata_t

real	xloc(max_sz, ndim)	*Particle position*
integer(kind=8)	src_id(max_sz)	*Index in unsorted source array*
integer(kind=8)	nd_id(max_sz)	*Index of associated node after construction of kD tree*
integer	fill	*Index of end of treedata array usage*
integer	node_range(2,max_sz)	*Indices for processing node i stored in node_range(2,i)*

Datastructure 4 TYPE :: treenode_t

integer	inuse	*Status of this entry*
integer	parent	*Index for parent*
integer	has_children	*false if there's no left/right*
integer	left, right	*Index for left/right child*
real*8	split	*Split value*
integer	splitdim	*Index of dimension corresponding to* split
integer	depth	*Number of splits from root*

Datastructure 5 TYPE, public :: treelist_t

treenode_t	nodes(max_sz)	*List of nodes comprising tree*
integer	fill	*Highest node index*

copy_treedata Utility to duplicate a *treedata_t* structure

The second routine (*tracer_to_kd*) is the interface to the simulation and requires either accessor functions to be provided or access to the native particle data structure. The current implementation is based on the expectation that the particle data is stored in several arrays where the index into that array is sufficient to identify the particle locally, as indicated in Datastructure 6.

Datastructure 6 Simulation data structure for particles

integer	fill	*Particle array fill level*
integer	state(ARRAY_SIZE)	*enum list of particle state for each index (UNOCCUPIED, HEALTHY, etc . . .)*
integer, parameter	HEALTHY	*enum value for active occupied state*
real	loc(ARRAY_SIZE,3)	*Particle position*

2.4.2 Intersection testing

As pointed out by Wang and Modest [6], line rays and point particles can not intersect in practice. The formulation used here is that line rays are checked for intersection against spheres of radius r_{eff}, where the effective radius is a problem-specific parameter.

2.5 Interface with CFD code

The intention is that a set of interface routines must be written that can work with both the ray tracing and CFD particle data structures to add incident rays, spawn new rays from particles, compute ray-particle interaction (currently absorption) when an intersection is detected, and compute geometry / topology relationships. These routines are contained in the module in the file *part_radht_interface_m.f90*. The key routines that must be defined are specified in Routine 1–5 below.

Routine 1 emit_rays_x0(rlist)

Inputs: local raylist (raylist_t)
Outputs: local raylist (raylist_t)
Effect: Add rays corresponding to incident insolation

Routine 2 emit_from_particles(rlist)

Inputs: local raylist (raylist_t)
Outputs: local raylist (raylist_t) augmented with new rays
Effect: Add rays corresponding to emission from particles

Routine 3 emitted_energy(temp,e)

Inputs: Particle temperature (temp)
Outputs: Emitted energy (e)
Effect: Physics routine to compute energy emitted by a particle at a given temperature, used in Routine 2.

Routine 4 absorb_ray(idx, rprop)

Inputs: Ray descriptor (rayprop_t), and simulation index of particle to update (idx)
Effect: Update particle state based on intersecting ray properties

Routine 5 get_exit_direction_new(org, dir, strt, stp, nextdest)

Inputs: Ray global origin (org), direction (dir)
Outputs: Ray local origin (strt), termination (stp), rank of next process on trajectory (nextdest)
Effect: Use routines for ray/plane intersection tests to determine the vector that needs to be checked for intersection with local particles, and which rank to pass the ray to next if it is not absorbed on this rank.

Routine 6 get_rank_coords(pos, rank, ierr)

Inputs: Position (pos)
Outputs: MPI Rank (rank), Error value (ierr)
Effect: Determine the MPI rank where the supplied position is local

Routine 7 get_timestep

Inputs:
Outputs: Simulation timestep for particle integration
Effect: Extract the simulation timestep for converting heat transfer rates into energy units and cache it in the module

3 Verification and Performance

For the purposes of verification we turn to the example problem presented by Modest in Example 21.3. The situation is depicted in Figure 2. A semi-infinite slab of 1m height of an isotropically scattering medium is irradiated by incident photons from above in a circle of radius R sampled according to:

$$r^* = R\sqrt{\xi_R} \qquad \phi^* = 2\pi\xi_\phi; \tag{3.1}$$

$$x^* = r^*\cos(\phi^*) \qquad y = r^*\sin(\phi^*). \tag{3.2}$$

These rays are then scattered by the medium, and the energy flux falling on to a small region at the bottom of the slab extending from $(0.2, -0.01)$ to $(0.22, 0.01)$ is computed conditional on the angle of incidence. To perform this calculation, particles are distributed so that the interaction length for the scattering medium is matched by the typical interaction distance based on the particle density:

$$\sigma_s = \pi r_p^2 n_p, \tag{3.3}$$

where r_p is the particle effective radius and n_p is the number density of the particles. Additionally, the particle emission functions need to be modified for consistency with the photon Monte Carlo method described by Modest. In the PMC method, a ray is initiated and then extended a distance:

$$l_\sigma = \frac{1}{\sigma_s}\ln\frac{1}{\xi_\sigma}, \tag{3.4}$$

where ξ_σ is drawn from a uniform distribution, then extended in a new direction based on isotropic scattering and so on until the ray exits the domain. When rays extend until they interact with particles that are randomly distributed in space, we need to ensure that the particles have no effective thermal mass. This is done by using the same absorption function, i.e.:

$$T_{\text{new}} = T + \frac{e_r}{mC_p}, \tag{3.5}$$

and then modifying the emission function so that:

$$e_{\text{em}} = mC_p(T - T_0). \tag{3.6}$$

This ensures that each particle emits all of the energy received during the last absorption. For every introduction of incident rays, the ray tracing, absorption, particle emission processes are looped over until no active rays remain, ensuring that all incident rays are effectively traced until they exit the domain. The simulated flux observations by the sensor as a function of acceptance angle are given in Figure 3.

3.1 Performance Breakdown

The relative cost of the major operations is given in Table 4 for two scenarios: the benchmark problem described above, where the number of incident rays is orders of magnitude larger than the number of particles, and only the hot particles emit rays, and, secondly, a problem where the number of incident rays is a small fraction of the number of particles and all of the particles emit rays. For the first case, the bulk of the computational effort is in performing ray-sphere intersection tests, even with the aid of the kD datastructure (observed to lead to a >50% reduction in the time to process the leaves compared checking exhaustively). For the uniform case, with far fewer rays active, the relative importance of the overhead for introducing incident rays increases, the overhead for constructing the kD tree becomes noticeable, and the imbalance decreases significantly. The high imbalance in the benchmark problem is because at the start of the iteration, *all* of the active rays are introduced on a small number of MPI ranks (those associated with the portion of the domain where the inlet is located). Even though the number of incident rays is small in the uniform case, the cost of relocating incident rays is significant, but this routine is not optimized and makes several passes over every active ray to identify those that are incident and eligible for relocation.

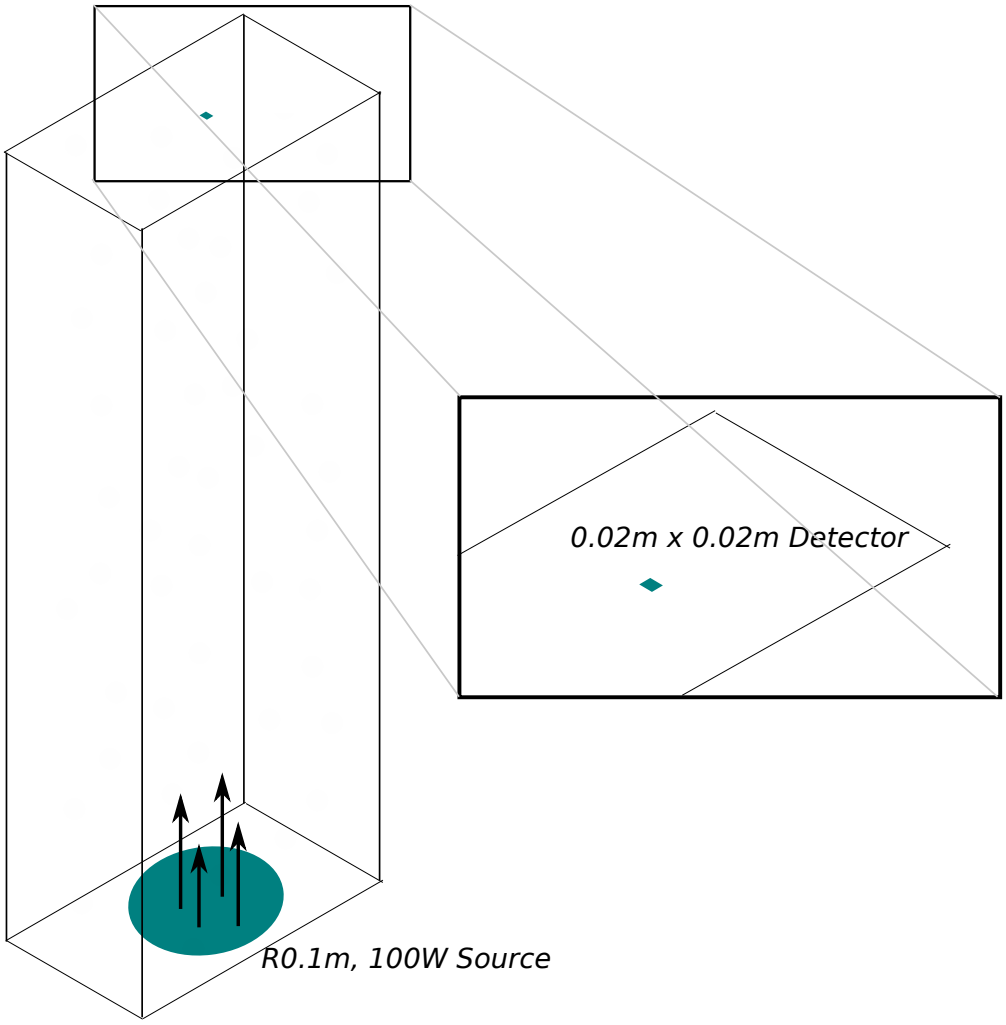

Figure 2. Configuration for test problem

Operation	Benchmark Case	All Particles Emit Case
Emit incident rays	1%	0%
Relocating incident rays	3%	29%
Construct particle kD tree	1%	19%
Emit rays from particles	1%	16%
Traverse kD tree	5 %	4%
Process tree leaves	50%	5%
Absorb rays	0%	0%
Balance of intersection detection	39%	40%
Maximum Imbalance[1]	89%	20%

Table 4. Division of labor for scattering test case

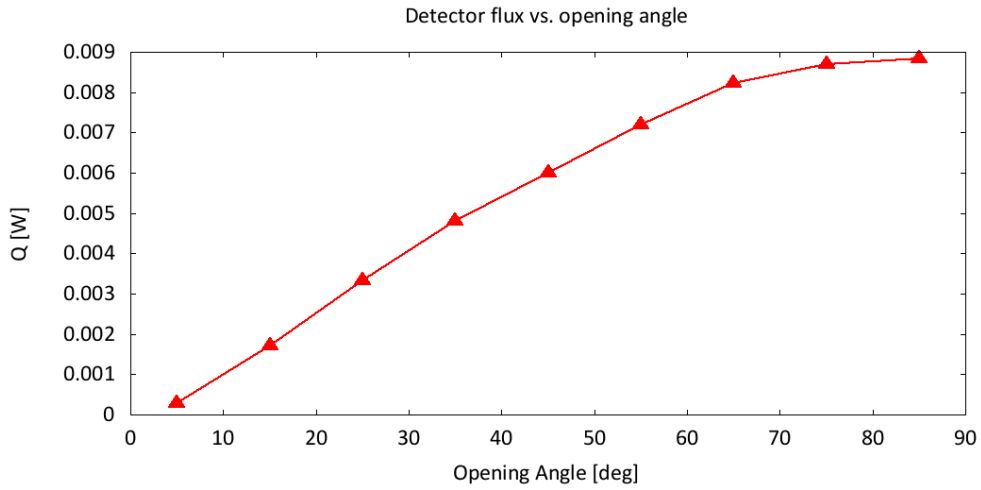

Figure 3. Benchmark problem solution

4 Future directions

The code described herein, the IHT toolkit, is under active development. The current motivation is to maintain a simplified proxy application for the radiative heat transfer in turbulent sooting flames that can be used to study performance of these algorithms on future architectures. Questions regarding IHT can be directed to the author at *ray.grout@nrel.gov*.

Bibliography

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[2] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorski, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using S3D. *Comp. Sci. Disc.*, 2:1–31, 2009.

[3] Ken-Ichi Ishikawa. Multiple stream Mersenne Twister PRNG. `http://theo.phys.sci.hiroshima-u.ac.jp/~ishikawa/PRNG/mt_stream_en.html`.

[4] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998.

[5] M. F. Modest. *Radiative Heat Transfer*. Elsevier Academic Press, 3rd edition, 2013.

[6] A. Wang and M.F. Modest. Photon monte carlo simulation for radiative transfer in gaseous media represented by discrete particle fields. *J. Heat Trans.*, 128:1041–1049, 2006.