M&C 2017 - International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering, Jeju, Korea, April 16-20, 2017, on USB (2017)

An Approach for Load Balancing Massively Parallel Transport Sweeps on Unstructured Grids[1]

Tarek H. Ghaddar, Jean C. Ragusa

Dept. of Nuclear Engineering, Texas A&M University, College Station, TX, 77843-3133
sethrj@umich.edu, honestabe@example.com

Abstract - *When running any massively parallel code, load balancing is a priority in order to achieve the highest possible parallel efficiency. A load-balanced problem has an equal number of degrees of freedom per processor. Load balancing is important in order to minimize idle time for all processors by equally distributing (as much as possible) the work for each processor. An unstructured meshing capability was implemented in PDT, Texas A&M University's massively parallel deterministic transport code, utilizing the Triangle mesh generator, hence allowing the user to define more realistic problem geometries and to define 3D problems through the extrusion of 2D meshes. However, unstructured grids are significantly harder to load balance than Cartesian rectangular meshes. A load balancing algorithm was implemented in PDT to minimize a load-imbalance metric. Three test cases were constructed and a series of 162 inputs were created for each case. A maximum improvement of 89.0% was seen in Test Case 1, 89.1% was seen in Test Case 2, and 55.2% was seen in Test Case 3.*

I. INTRODUCTION

When running any massively parallel code, attaining load balancing is a priority in order to achieve the best possible parallel efficiency. A load balanced problem has an equal number of degrees of freedom per processor. Load balancing is an important factor to minimize idle time for all processors and is attained by equally distributing (as much as possible) the work load among all processors.

To the best of our knowledge, transport sweeps is the only scalable solution technique on the current leadership class supercomputers. PDT, Texas A&M University's massively parallel deterministic transport code, has been shown to scale on logically Cartesian grids out to 750,000 cores [1]. Logically Cartesian grids are constructed with mesh cells that are identified using integer triplets ijk (i.e., cubic cells), but allow for vertex motion in order to conform to curved shapes. PDT solves radiation transport problems (neutron, gamma, coupled neutron-gamma, electron, coupled electron-photon, and radiative transfer). It uses discrete ordinates for angular discretization, multi-group and FEDS energy differencing, and discontinuous finite elements in space. add a ref for PWLD, you can take JCP Turcksin/Ragusa+JCP Teresa, add a ref for FEDS, no need for refs for Sn or MG

A new unstructured meshing capability was implemented in PDT in order to realistically represent certain geometries. Cut lines (cut planes for 3D cases) are used to partition such geometries into logically-Cartesian subdomains, which are then individually meshed in parallel using the Triangle Mesh Generator [2]. These subdomains are then "stitched" or "glued" together in order to create a continuous geometry. 2D meshes can be extruded in the z dimension for 3D problems.

However, unstructured meshes often create unbalanced problems due to the way localized features are meshed, so a load balancing algorithm was added into PDT.

[1]Notice: this manuscript is a work of fiction. Any resemblance to actual articles, living or dead, is purely coincidental.

II. THEORY

1. The Transport Equation

The steady-state neutron transport equation describes the behavior of neutrons in a medium and is given by Eq. (1):

$$\begin{aligned}\boldsymbol{\Omega}\cdot\boldsymbol{\nabla}\psi(\boldsymbol{r},E,\boldsymbol{\Omega})+\Sigma_t(\boldsymbol{r},E)\psi(\boldsymbol{r},E,\boldsymbol{\Omega})=\\ \int_0^\infty dE'\int_{4\pi}d\Omega'\Sigma_s(\boldsymbol{r},E'\rightarrow E,\Omega'\rightarrow\Omega)\psi(\boldsymbol{r},E',\boldsymbol{\Omega}')\\ +S_{ext}(\boldsymbol{r},E,\boldsymbol{\Omega}),\end{aligned}\quad(1)$$

where $\boldsymbol{\Omega}\cdot\boldsymbol{\nabla}\psi$ is the leakage term and $\Sigma_t\psi$ is the total collision term (absorption, outscatter, and within group scattering). The right hand side of Eq. (1) represents the gain terms, where S_{ext} is the external source of neutrons and $\int_0^\infty dE'\int_{4\pi}d\Omega'\Sigma_s(E'\rightarrow E,\Omega'\rightarrow\Omega)\psi(\boldsymbol{r},E',\boldsymbol{\Omega}')$ is the inscatter term, which represents all neutrons scattering from energy E' and direction $\boldsymbol{\Omega}'$ into energies about E and directions about $\boldsymbol{\Omega}$. We assumed a non-multiplying medium and will further assume isotropic scattering for simplicity and conciseness. We introduce the scalar flux as the integral of the angular flux:

$$\phi(\boldsymbol{r},E')=\int_{4\pi}d\Omega'\psi(\boldsymbol{r},E',\boldsymbol{\Omega}').\quad(2)$$

Using the multigroup approximation yields:

$$\begin{aligned}&\boldsymbol{\Omega}\cdot\boldsymbol{\nabla}\psi_g(\boldsymbol{r},\boldsymbol{\Omega})+\Sigma_{t,g}(\boldsymbol{r})\psi_g(\boldsymbol{r},\boldsymbol{\Omega})\\ &=\frac{1}{4\pi}\sum_{g'}\Sigma_{s,g'\rightarrow g}(\boldsymbol{r})\phi_{g'}(\boldsymbol{r})+S_{ext,g}(\boldsymbol{r},\boldsymbol{\Omega}),\quad\text{for } 1\le g\le G\end{aligned}\quad(3)$$

where the multigroup transport equations now form a system of G coupled equations. Next, we discretize in angle using the discrete ordinates method, whereby an angular quadrature $(\boldsymbol{\Omega}_m,w_m)_{1\le m\le M}$ is used to solve the above equations along a given set of directions $\boldsymbol{\Omega}_m$:

$$\boldsymbol{\Omega}_m \cdot \boldsymbol{\nabla}\psi_{g,m}(\boldsymbol{r}) + \Sigma_{t,g}(\boldsymbol{r})\psi_{g,m}(\boldsymbol{r}) = \frac{1}{4\pi}\sum_{g'}\Sigma_{s,g'\to g}(\boldsymbol{r})\phi_{g'}(\boldsymbol{r}) + S_{ext,g,m}(\boldsymbol{r}), \quad (4)$$

where the subscript m is introduced to describe the angular flux in direction m. The scalar flux integral is numerically evaluated

$$\phi_g(\boldsymbol{r}) \approx \sum_{m=1}^{m=M} w_m \psi_{g,m}(\boldsymbol{r}). \quad (5)$$

From Equation (3), it is clear that we are solving a sequence of transport equations, one equation per group and per direction. Therefore, all transport equations are of the following form:

$$\boldsymbol{\Omega}_m \cdot \boldsymbol{\nabla}\psi_m(\boldsymbol{r}) + \Sigma_t(\boldsymbol{r})\psi_m(\boldsymbol{r}) = \frac{1}{4\pi}\Sigma_s(\boldsymbol{r})\phi(\boldsymbol{r}) + q_m^{ext+inscat}(\boldsymbol{r}) = q_m(\boldsymbol{r}), \quad (6)$$

where the group index notation is omitted for brevity. In order to obtain the solution for this discrete form of the transport equation, source iteration is introduced, for instance.

$$\boldsymbol{\Omega}_m \cdot \boldsymbol{\nabla}\psi_m^{(l+1)}(\boldsymbol{r}) + \Sigma_t\psi_m^{(l+1)}(\boldsymbol{r}) = q_m^{(l)}(\boldsymbol{r}), \quad (7)$$

where the right hand side terms of Eq. (4) have been combined into one general source term, q_m. The angular flux of iteration $(l+1)$ is calculated using the (l^{th}) value of the scalar flux.

After the angular and energy dependence have been accounted for, Eq. (7) must be discretized in space as well. We use a discontinuous Galerkin approximation in space, and the solution across a cell interface is connected based on an upwind approach, where face outflow radiation becomes face inflow radiation for the downwind cells. The solution is obtained by meshing the domain and solving the spatial problem one cell at a time for a given direction and a given group. Figure 1 shows the sweep ordering for a given direction on both a structured and unstructured mesh.

The number in each cell represents the order in which the cells are solved. All cells must receive the solution downwind from them before solving for their own solution. This dependency can be represented and stored as a directed task dependence graph, shown in Fig. 2.

The order in which radiation in a cell is solved is given by a task dependence graph. Transport sweep can be performed on parallel architectures in order to obtain the solution faster, as well as distribute the memory to many processors for memory intensive cases. Provably-optimal transport sweeping has been described in [3] and demonstrated in PDT using logically-Cartesian meshes. Our work utilizes that transport sweep machinery but adapts it to unstructured meshes. Performing a transport sweep on an unstructured mesh presents two challenges: (1) performing a transport sweep on a massively parallel scale in an efficient manner and (2) keeping non-concave sub-domains due to leverage fromt he provably-optimal transport sweep algorithms. a now we wait a long time for a description of the proposed work. not ideal. it feels the following is too detailed

A parallel sweep algorithm is defined by three properties [3] :

- partitioning: dividing the domain among available processors

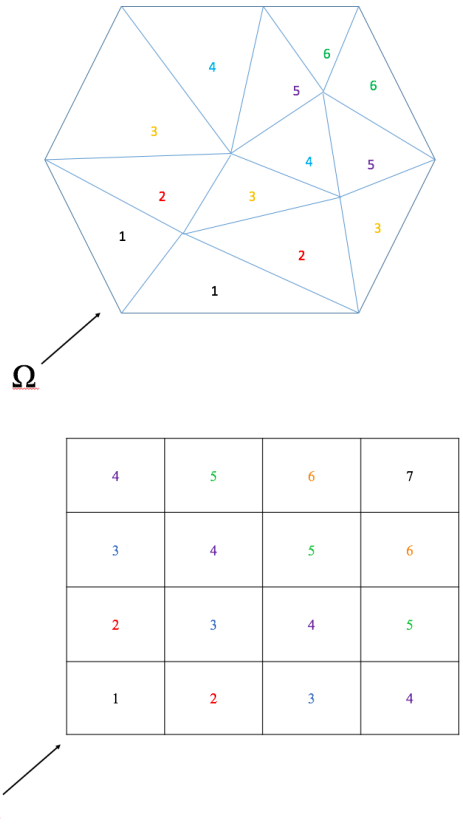

Fig. 1. A demonstration of a sweep on a structured and unstructured mesh.

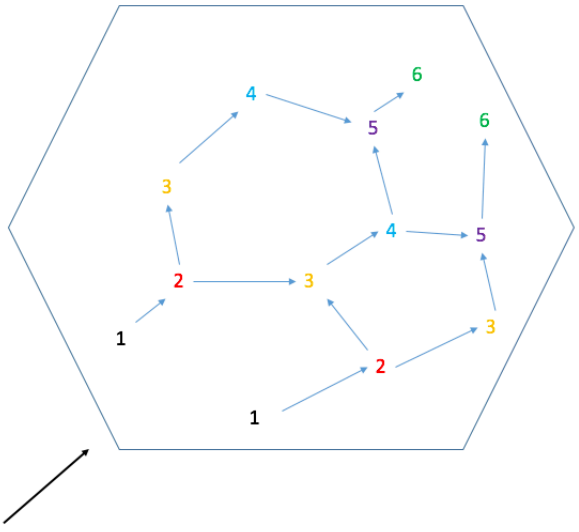

Fig. 2. A task dependence graph of the unstructured mesh example in Fig. 1.

- aggregation: grouping cells, directions, and energy groups into tasks
- scheduling: choosing which task to execute if more than one is available

The basic concepts of parallel transport sweeps, partitioning, aggregation, and scheduling, are most easily described in the context of a structured transport sweep. A structured transport sweep takes place on a Cartesian mesh. Furthermore, the work proposed utilizes aspects of the structured transport sweep.

If M is the number of angular directions per octant, G is the total number of energy groups, and N is the total number of cells, then the total fine grain work units is $8MGN$. The factor of 8 is present as M directions are swept for all 8 octants of the domain. The finest grain work unit is the calculation of a single direction and energy groups unknowns in a single cell, or $\psi_{m,g}$ for a single cell.

In a regular grid, we have the number of cells in each Cartesian direction: N_x, N_y, N_z. These cells are aggregated into "cellsets". If M is the total number of angular directions, G is the total number of energy groups, and N is the total number of cells, then the total fine grain work units is $8MGN$. The factor of 8 is present as M directions are swept for all 8 octants of the domain. The finest grain work unit is the calculation of a single direction and energy groups unknowns in a single cell, or $\psi_{m,g}$ for a single cell.

Fine grain work units are aggregated into coarser-grained units called *tasks*. A few terms are defined that describe how each variable is aggregated.

- $A_x = \frac{N_x}{P_x}$, where N_x is the number of cells in x and P_x is the number of processors in x
- $A_y = \frac{N_y}{P_y}$, where N_y is the number of cells in y and P_y is the number of processors in y
- $N_g = \frac{G}{A_g}$
- $N_m = \frac{M}{A_m}$
- $N_k = \frac{N_z}{P_z A_z}$
- $N_k A_x A_y A_z = \frac{N_x N_y N_z}{P_x P_y P_z}$

It follows that each process owns N_k cell-sets (each of which is A_z planes of $A_x A_y$ cells), $8N_m$ direction-sets, and N_g group-sets for a total of $8N_m N_g N_k$ tasks.

One task contains $A_x A_y A_z$ cells, A_m directions, and A_g groups. Equivalently, a task is the computation of one cellset, one groupset, and one angleset. One task takes a stage to complete. This is particularly important when comparing sweeps to the performance models.

Equation (8) approximately defines parallel sweep efficiency. This can be calculated for specific machinery and partitioning parameters by substituting in values calculated using Eqs. (12), (13), and (14).

$$\begin{aligned}\epsilon &= \frac{T_{\text{task}} N_{\text{tasks}}}{[N_{\text{stages}}][T_{\text{task}} + T_{\text{comm}}]} \\ &= \frac{1}{[1 + \frac{N_{\text{idle}}}{N_{\text{tasks}}}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]}\end{aligned} \tag{8}$$

Equations (9) and 10 show how T_{comm} and T_{task} are calculated:

$$T_{\text{comm}} = M_L T_{\text{latency}} + T_{\text{byte}} N_{\text{bytes}} \tag{9}$$

$$T_{\text{task}} = A_x A_y A_z A_m A_g T_{\text{grind}} \tag{10}$$

where T_{latency} is the message latency time, T_{byte} is the time required to send one byte of message, N_{bytes} is the total number of bytes of information that a processor must communicate to its downstream neighbors at each stage, and T_{grind} is the time it takes to compute a single cell, direction, and energy group. M_L is a latency parameter that is used to explore performance as a function of increased or decreased latency. If a high value of M_L is necessary for the performance model to match computational results, improvements should be made in code implementation.

2. KBA Partitioning for Structured Grids

Several parallel transport sweep codes use KBA partitioning in their sweeping, such as Denovo [4] and PARTISN [5]. The KBA partitioning scheme and algorithm was developed by Koch, Baker, and Alcouffe [5].

The KBA algorithm traditionally chooses $P_z = 1, A_m = 1, G = A_g = 1, A_x = N_x/P_x, A_y = N_y/P_y$, with A_z being the selectable number of z-planes to be aggregated into each task. With $N_k = N_z/A_z$, each processor performs $N_{\text{tasks}} = 8MN_k$ tasks. With the KBA algorithm, $2MN_k$ tasks are pipelined from a given corner of the 2D processor layout. The far corner processor remains idle for the first $P_x + P_y - 2$ stages, which means that an octant-pair (or quadrant) sweep completes in $2MN_k + P_x + P_y - 2$ stages. If an octant-pair sweep does not begin until the previous pair's finishes, the full sweep requires $8MN_k + 4(P_x + P_y - 2)$ stages, which means the KBA parallel efficiency is:

$$\varepsilon_{KBA} = \frac{1}{[1 + \frac{4(P_x+P_y-2)}{8MN_k}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]} \tag{11}$$

3. The Structured Transport Sweep in PDT

The minimum possible number of stages for given partitioning parameters P_i and A_j is $2N_{\text{fill}} + N_{\text{tasks}}$. N_{fill} is both the minimum number of stages before a sweepfront can reach the center-most processors and the number needed to finish a direction's sweep after the center-most processors have finished. Equations (12), (13), and (14) define N_{fill}, N_{idle}, and N_{tasks}:

$$N_{\text{fill}} = \frac{P_x + \delta_x}{2} - 1 + \frac{P_y + \delta_y}{2} - 1 + N_k(\frac{P_z + \delta_z}{2} - 1) \tag{12}$$

$$N_{\text{idle}} = 2N_{\text{fill}} \tag{13}$$

$$N_{\text{tasks}} = 8N_m N_g N_k \tag{14}$$

where δ_u is 1 for P_u odd, and 0 for P_u even.

When using KBA, P_z is fixed to 1, and with hybrid KBA, P_z is fixed to 2. Volumetric partitioning means that P_z is greater than two. Figure 3 shows three different partitioning schemes used in transport sweeps, KBA (which is defined in the previous section), volumetric non-overloaded, and volumetric overloaded. Volumetric non-overloaded requires that all

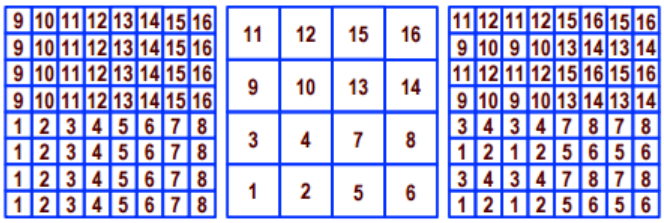

Fig. 3. Three different partitioning schemes in 2D, from left to right: KBA, volumetric non-overloaded, and volumetric overloaded. [1].

cells owned by a processor are contiguous, where as volumetric non-overloaded partitioning does not have this restriction.

The overloaded volumetric partitioning proceeds as follows:

1. In a 2D (3D) domain, cellsets are divided into 4 (8) spatial quadrants (octants), with an equal number of cellsets in each SQO (SQO is defined as a spatial quadrant or octant).
2. Assign 1/4 of the processors (1/8) in 3D to each SQO.
3. Choose the individual overload factors ω_x, ω_y, and ω_z and individual processor counts P_x, P_y, and P_z, such that $\omega_x\omega_y\omega_z = \omega_r$ and $P_xP_yP_z = P$, with all P_u even. ω_u is defined as the number of cellsets assigned to each P_u.
4. An array of $\omega_x \cdot \omega_y \cdot \omega_z$ "tiles" in each SQO. Each tile is an array of $1/2P_x \cdot 1/2P_y \cdot 1/2P_z$ cellsets. These cellsets are mapped one-to-one to the $1/2P_x \cdot 1/2P_y \cdot 1/2P_z$ processors assigned to the SQO, using the same mapping in each tile.

Each tile has a logically identical layout of cellsets, and each processor owns exactly one cellset in each tile in its SQO. This makes each processor responsible for ω_r cellsets.

In order to properly outline the optimal scheduling rules, the variables X, Y, and Z are defined as $P_u/2$ for each respective direction $u = x, y, z$. This splits up the processor layout into octants, where each processor has an index (i, j, k) determining where it is in the layout. Tiles are also indexed and referred to in the same way with the notation $T(i, j, k)$.

The optimal scheduling algorithm rules are as follows:

1. If $i \leq X$, then tasks with $\Omega_x > 0$ have priority, while for $i > X$, tasks with $\Omega_x < 0$ have priority.
2. If multiple ready tasks have the same sign on Ω_x, apply rule 1 to j, Y, Ω_y.
3. If multiple ready tasks have the same sign on Ω_x and Ω_y, apply rule 1 to k, Z, Ω_z.
4. If multiple tasks are ready in the same octant, then priority goes to the cellset for which the priority octant has greatest downstream depth.
5. If multiple ready tasks are in the same octant and have the same downstream depth of graph in x, then priority goes to the cellset for which the priority octant has greatest downstream depth of graph in y.
6. If multiple ready tasks are in the same octant and have the same downstream depth of graph in x and y, then priority goes to the cellset for which priority octant has greatest depth of graph in z.

This ensures that each SQO orders the octants: the one it can start right away (A), three that have one sign difference from A(B, C, and D), three that have two sign differences ($\bar{D}, \bar{C}, \bar{B}$), and one in opposition to its primary ($\bar{A}$). For example, if octant A is octant $(+x, +y, +z)$, then it's secondary octants (only one sign change at a time) would be octants $(-x, +y, +z)$, $(+x, -y, +z)$ and $(+x, +y, -z)$.

There are three constraints in order to achieve the optimal stage count. In these constraints, $M = \omega_g\omega_m/8$, which is the number of tasks per octant per cellset.

1. $M \geq 2(Z - 1)$
2. $\omega_z M \geq 2(Y - 1)$
3. If $\omega_x > 1$, then $\omega_y\omega_z M \geq X$

Constraint 1 ensures that there is no idle time between a processor finishing an octant's work in one tile and beginning that octant's work on the next tile in the same tile-column; processor $P(1, Y, 1)$ finishing its tile $T(1, \omega_y, 1)$ octant C work and beginning its octant B work; processor $P(X, 1, 1)$ finishing its tile $T(\omega_x, 1, 1)$ octant D work and beginning its octant B work. Constraint 2 ensures that there is no idle time time between a processor finishing an octant's work for one z column of tiles and beginning that octant's work on the next column; processor $P(X, 1, 1)$ finishing its tile $T(\omega_x, 1, 1)$ octant D work available to it and beginning its octant C work. Constraint 3 ensures that there is no idle time between a processor finishing an octant's work for one yz plane of tiles and beginning that octant's work in the next plane.

As a result of these constraints, there is no idle time for a variety of situtations. At large processor counts, the product $\omega_m\omega_g$ must be large, which requires N_mN_g be large. This means that a weak scaling series refined only in space, but only coarsely refined in angle and energy, will eventually fail the constraints.

The optimal efficiency formula changes slightly from the KBA and hybrid KBA partitioning method in order to account for the overload factors. The only change is in the $\frac{N_{idle}}{N_{tasks}}$ term, as shown in Eq. (15).

$$\varepsilon_{opt} = \frac{1}{[1 + \frac{P_x+P_y+P_z-6}{\omega_g\omega_m\omega_r}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]} \tag{15}$$

4. The Unstructured Transport Sweep

In an unstructured mesh, the number of cells cannot be described in the same way as an unstructured mesh. In PDT, the 2D geometry is first subdivided into subsets, which are just rectangular subdomains. Within each subset, an unstructured mesh is generated (using Triangle) and then extruded in 3D. These subsets become the N_x, N_y, N_z equivalent for an structured mesh. The spatial aggregation in a PDT unstructured mesh is done by aggregating subsets into cellsets.

While the structured PDT transport sweep has scaled well out to 750,000 cores, similar levels of parallel scaling have not been achieved using unstructured sweeps yet. Pautz proposed a new list scheduling algorithm has been constructed for modest levels of parallelism (up to 126 processors)[6]. what's the point of this paragraph? You are way way past the literature review. at this point on the paper, it is only about your work

There are three requirements for a sweep scheduling algorithm to have. First, the algorithm should have low complexity, since millions of individual tasks are swept over in a typical problem. Second, the algorithm should schedule on a set of processors that is small in comparison to the number of tasks in the sweep graph. Last, the algorithm should distribute work in the spatial dimension only, so that there is no need to communicate during the calculation of the scattering source.

Here is the pseudocode[6] for the algorithm:

```
Assign priorities to every cell-angle pair
Place all initially ready tasks in priority queue
While (uncompleted tasks)
    For i=1,maxCellsPerStep
       Perform task at top of priority queue
       Place new on-processor tasks in queue
    Send new partition boundary data
    Receive new partition boundary data
    Place new tasks in queue
```

An important part of the algorithm above is the assigning priorities to tasks. Specialized prioritization heuristics generate partition boundary data as rapidly as possible in order to minimize the processor idle time.

Nearly linear speedups were obtained on up to 126 processors[6]. Further work is being done for scaling to thousands of processors. how does this relate to our load-balancing?

A. Cycle Detection

no need for this here. at most, one sentence somewhere because we never have cycles with how we do it so far: we have orthogonal subsets, there are no cycles in 2D triangular grids and extrusion preserves this. period A cycle is a loop in a directed graph and they can occur commonly in unstructured meshes. However, they do not exist in 2D triangular extruded problems and, because our domain partitioning is convex, arbitrary degenerate polygons appearing on subdomain boundaries will not produce cycles. Even though they are not applicable to this application of unstructured transport sweeps, they are discussed here for completeness.

Cycles can cause hang time in the problem, as a processor will wait for a message that might will never come. This means that the computation for one or more elements will never be completed. The solution to this is to "break" any cycles that exist by removing an edge of the task dependence graph (TDG). Old flux information is used on a particular element face in the domain. Most of the time, the edge removed is oriented obliquely with respect to the radiation direction.

Algorithms for finding cycles are called *cycle detection* algorithms. This must be done efficiently in parallel, both because the task dependence graph is distributed and because the finite element grid may be deforming every timestep and changing the associated TDG.

Cycle detection utilizes two operations: trim and mark. Trimming identifies and discards elements which are not in cycles. At the beginning of cycle detection, graphs are trimmed in the downwind direction, then the remaining graphs are trimmed in the upwind direction. A pivot vertex is then selected in each graph. Graph vertices are then marked as upwind, downwind, or unmarked. Then, if any vertices are both upwind and downwind, the cycle is these vertices plus the pivot vertex. An edge is removed between 2 cycle vertices, and 4 new graphs are created: a new cycle, the upwind vertices without the cycle, the downwind vertices without the cycle, and a set of unmarked vertices. This recursively continues until all cycles are eliminated.

III. MOTIVATION AND METHODS

The capability for PDT to generate and run on an unstructured mesh is important because it allows us to run problems without having to conform our mesh to the problem as much. The idea is to have a logically Cartesian grid (creating orthogonal "subsets") with an unstructured mesh inside each subset. These logically Cartesian subdomains are obtained using cut planes in 3D and cut lines in 2D. Figure 4 demonstrates this functionality. It is decomposed into 3 subsets in x and 3 in y, with the first two subsets meshed using the Triangle Mesh Generator[2], a 2D mesh generator.

This orthogonal grid is superimposed and each subset is meshed in parallel. Subsets are now the base structured unit when calculating our parallel efficiency. Discontinuities along the boundary are fixed by "stitching" hanging nodes, creating degenerate polygons along subset boundaries. Because PDT's spatial discretization employs Piece-Wise Linear Discontinuous (PWLD) finite element basis functions, there is no problem solving on degenerate polygons.

When using the unstructured meshing capability in PDT, the input geometry is described by a Planar Straight Line Graph (PSLG)[2]. After superimposing the orthogonal grid, a PSLG is created for each subset, and meshed. Because the input's and each subset's PSLG must be described and meshed in 2D, the mesh can be extruded in the z dimension in order to give us the capability to run on 3D problems. Obviously, this is not as good as general an unstructured grid as a tetrahedral grid, but for many problems (e.g., reactor problems), it is a useful capability to have and it can be employed to assess load-balancing algorithm.

When discussing the parallel scaling of transport sweeps, a load balanced problem is of great importance. A load balanced problem has an equal number of degrees of freedom per processor. Load balancing is important in order to minimize idle time for all processes by equally distributing (as much as possible) the work on each process. For the purposes of unstructured meshes in PDT, we are looking to "balance" the number of cells. Ideally, each process will be responsible for an equal number of cells.

If the number of cells in each subset can be reasonably balanced, then the problem is effectively load balanced. The

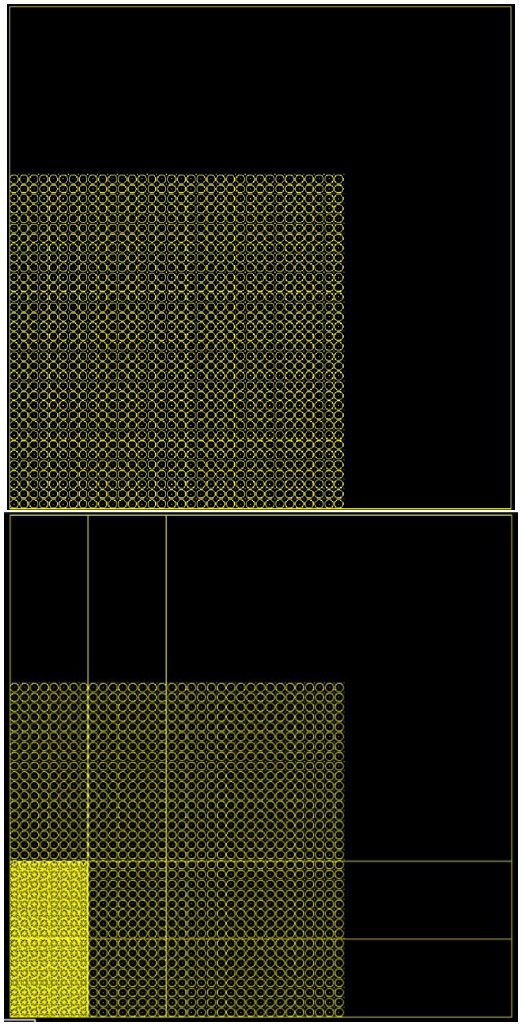

Fig. 4. A PSLG describing a fuel lattice, and with an orthogonal “subset" grid imposed on the PSLG.

Load Balance algorithm described below details how the subsets will be load balanced. In summary, the procedure of the algorithm involves moving the initially user specified x and y cut planes, re-meshing, and iterating until a reasonably load balanced problem is obtained. Equation 16 shows the equation for calculating the load balancing metric, which dictates how balanced or unbalanced the problem is.

$$f = \frac{\max\limits_{ij}(N_{ij})}{\frac{N_{tot}}{I \cdot J}}, \tag{16}$$

where f is the load balance metric, N_{ij} is the number of cells in subset i, j, N_{tot} is the global number of cells in the problem, and I and J are the total number of in the x and y direction, respectively. The metric is a measure of the maximum number of cells per subset divided by the average number of cells per subset.

The load balancing algorithm moves cut planes based on two sub-metrics, f_I and f_J. Equation (17) defines these two parameters:

$$f_I = \max_i[\sum_j N_{ij}]/\frac{N_{tot}}{I}$$
$$f_J = \max_j[\sum_i N_{ij}]/\frac{N_{tot}}{J}. \tag{17}$$

f_I is calculated by taking the maximum number of cells per column and dividing it by the average number of cells per column. f_j is calculated by taking the maximum number of cells per row and dividing it by the average number of cells per row. If these two numbers are greater than predefined tolerances, the cut lines in the respective directions are redistributed. Once redistribution and remeshing occur, a new metric is calculated. This iterative process occurs until a maximum number of iterations is reached, or until f converges within the user defined tolerance. The Load Balance algorithm behaves as follows:

```
//I,J subsets specified by user
//Check if all subsets meet the tolerance
while (f > tol_subset)
{
  //Mesh all subsets
  if (f_I > tol_column)
  {
    Redistribute(X);
  }
  if (f_J > tol_row)
  {
    Redistribute(Y);
  }
}
//Remesh to get the final mesh.
```

Redistribute: A function that moves cut lines in either X or Y.

Input: CutLines (X or Y vector that stores cut lines).
Input: num_tri_row or num_tri_col, # of tri in each row/col
Input: The total number of triangles in the domain, N_{tot}
stapl::array_view num_tri_view, over num_tri_row/column
stapl::array_vew offset_view
stapl::partial_sum(num_tri_view) {Perform prefix sum}
{We now have a cumulative distribution stored in offset_view}
for i = 1 :CutLines.size()-1 **do**
vector <double> pt1 = [CutLines(i-1), offset_view(i-1)]
vector <double> pt2 = [CutLines(i), offset_view(i)]
ideal_value = $i \cdot \frac{N_{tot}}{\text{CutLines.size()}-1}$
{Calculate X-intersect of the line formed by pt1 and pt2}
{and the line y = ideal_value.}
X-intersect(pt1,pt2,ideal_value)
CutLines(i) = X-intersect
end for

IV. RESULTS AND ANALYSIS

The following sections will showcase the metric behavior and convergence for three test cases, solution verification for pure absorber and pure scatterer 2D slab problems, and the new unstructured meshing capability both in 2D and 3D.

1. Test Cases for Metric Behavior and Convergence

In order to showcase the behavior of the load balancing metric, calculated by Eq. 16 three test cases are presented. Figure 5 shows the first test case, a 20 cm by 20 cm domain with two pins in opposite corners of the domain. Figure 6 shows the same size domain but with the pins on the same side.These are two theoretically very unbalanced cases, as geometrically there are two features located distantly from each other with an empty geometry throughout the rest of the domain. Figure 7 shows a lattice and reflector, which due to it's denser and repeated geometry, theoretically is a more balanced problem.

A series of 162 inputs was constructed for each case. These inputs are constructed by varying the maximum triangle area from the coarsest possible (refers to Triangle utilizing the fewest possible triangles to mesh the geometry) to 0.01 cm^2 and the number of subsets, N from 2×2 to 10×10.

2. Metric Behavior and Convergence

For each test case, the 162 input inputs are run twice, once with no load balancing iterations, and once with ten load balancing iterations. The best metric is reported and recorded. Three figures for each test cases are presented below: the first figure will show the metric behavior for no iterations, the second figure will show the metric behavior for each input run with ten load balancing iterations, and the third figure will show a ratio of the ten iteration runs over the no iteration runs.

Figure 8 shows the metric behavior for Fig. 5. The maximum metric value is 24.7650, and occurs when Fig. 5 is run with 8x8 subsets and a maximum triangle area of 1.6 cm^2. The minimum metric value is 1.0016 and occurs when Fig. 5 is run

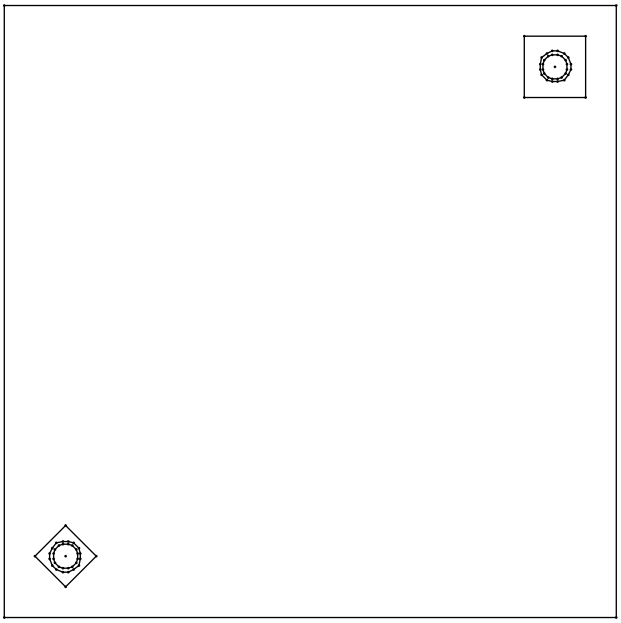

Fig. 5. The first test case used in order to test effectiveness and convergence of the load balancing metric.

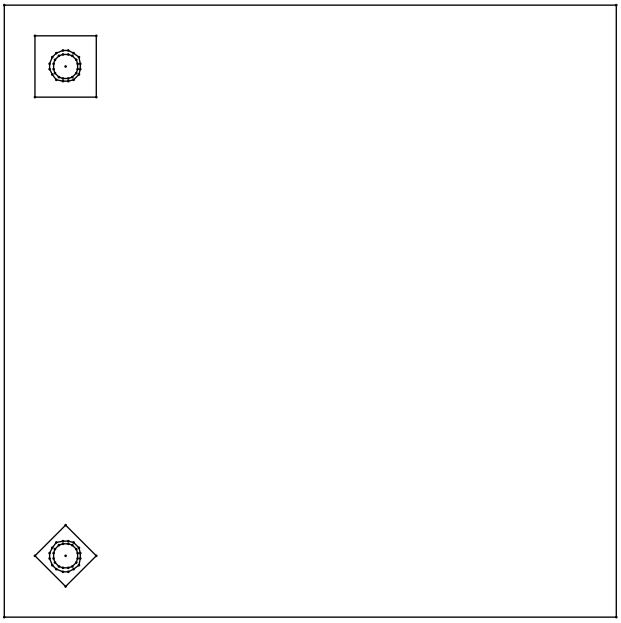

Fig. 6. The second test case used in order to test effectiveness and convergence of the load balancing metric.

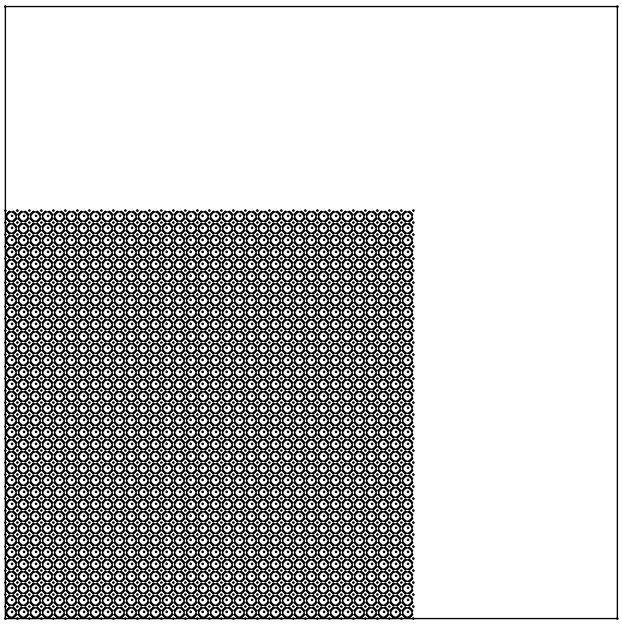

Fig. 7. The third test case used in order to test effectiveness and convergence of the load balancing metric.

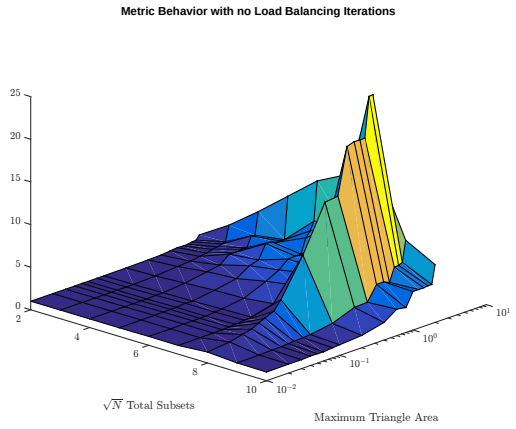

Fig. 8. The metric behavior of the first test case run with no load balancing iterations.

with 4x4 subsets and a maximum triangle area of 0.04 cm^2. The z axis in all figures is the value of the metric.

Figure 9 shows the metric behavior for Fig. 5 after 10 load balancing iterations. The maximum metric value is 5.0538 and occurs when Fig. 5 is run with 10x10 subsets and a maximum triangle area of 1.2 cm^2. The minimum metric value is 1.0017 and occurs when Fig. 5 is run with 4x4 subsets and a maximum triangle area of 0.04 cm^2.

Figure 10 shows the difference in metric behavior for Fig. 5. This difference is calculated by dividing the metric with no iterations by the metric with 10 iterations. The maximum improvement has a value of 0.1097 and occurs for Fig. 5 is

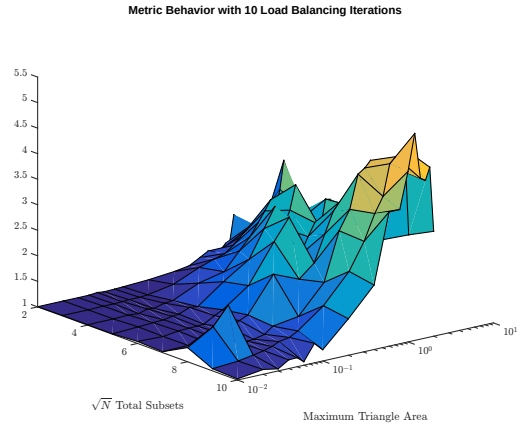

Fig. 9. The metric behavior of the first test case run with 10 load balancing iterations.

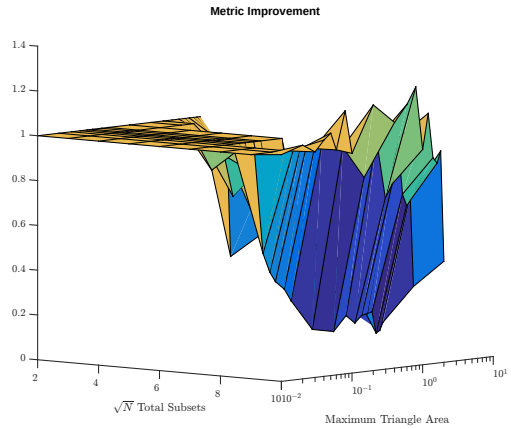

Fig. 10. The difference in metric behavior between no iteration and 10 iterations. The closer the z-value to zero, the better the improvement.

run with 8x8 subsets with a maximum triangle area of 1.6 cm^2. The minimum improvement has a value of very close to 1.0 and occurs for many of the inputs.

Figure 11 shows the metric behavior for Fig. 6. The maximum metric is 22.6654 and occurs when Fig. 6 is run with 8x8 subsets with a maximum triangle area of 1.8 cm^2. The minimum metric is 1.0024 and occurs when Fig. 6 is run with 2x2 subsets with a maximum triangle are of 0.01 cm^2.

Figure 12 shows the metric behavior for Fig. 6 after ten load balancing iterations. The maximum metric is 3.9929 and occurs when Fig. 6 is run with 10x10 subsets with a maximum triangle area of 1.8 cm^2. The minimum metric is 1.0024 and occurs when Fig. 6 is run with 2x2 subsets with a maximum triangle are of 0.01 cm^2.

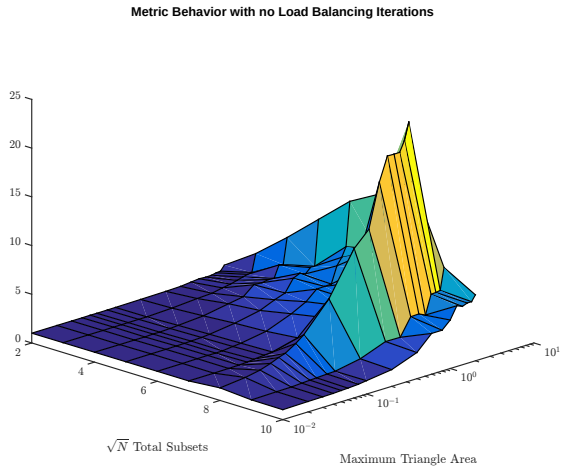

Fig. 11. The metric behavior of the second test case run with no load balancing iterations.

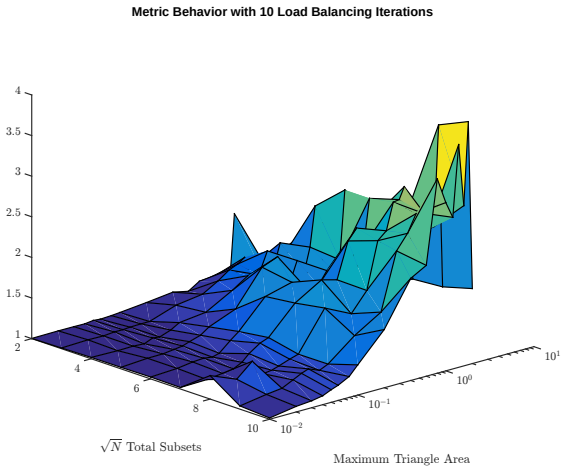

Fig. 12. The metric behavior of the second test case run with 10 load balancing iterations.

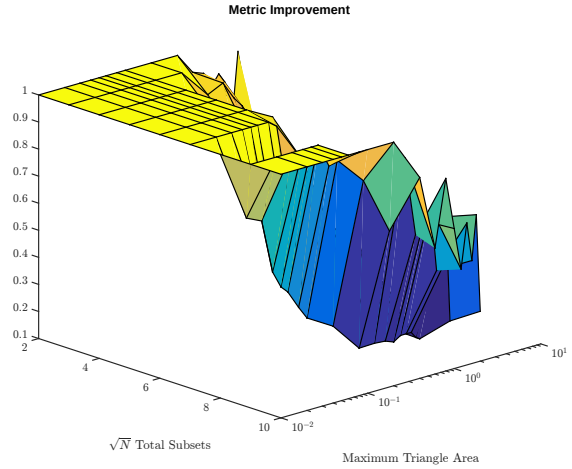

Fig. 13. The difference in metric behavior of the second test case with no iteration and 10 iterations. The closer the z-value to zero, the better the improvement.

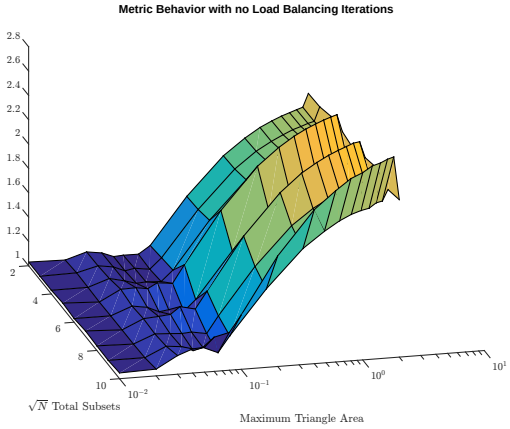

Fig. 14. The difference in metric behavior of the third test case with no load balancing iterations.

Figure 13 shows the difference in metric behavior for Fig. 6. The maximum improvement has a value of 0.1090 and occurs for Fig. 6 is run with 8x8 subsets with Triangle's coarsest possible mesh generation settings. The minimum improvement has a value of very close to 1.0 and occurs for many of the inputs.

Figure 14 shows the metric behavior for Fig. 7. The maximum metric is 2.6489 and occurs when Fig. 7 is run with 10x10 subsets with a maximum triangle area of 1.8 cm^2. The minimum metric is 1.0179 and occurs when Fig. 7 is run with 2x2 subsets with a maximum triangle are of 0.08 cm^2.

Figure 15 shows the metric behavior for Fig. 7 after ten

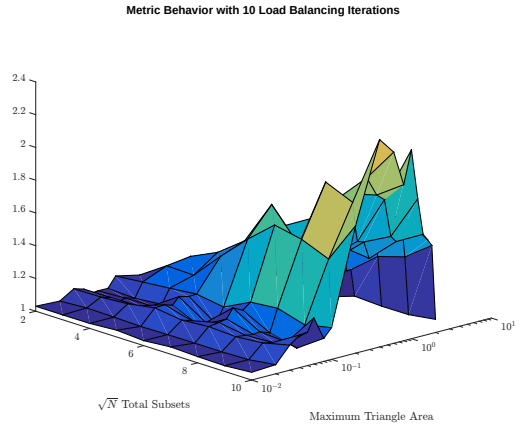

Fig. 15. The difference in metric behavior of the third test case after ten load balancing iterations.

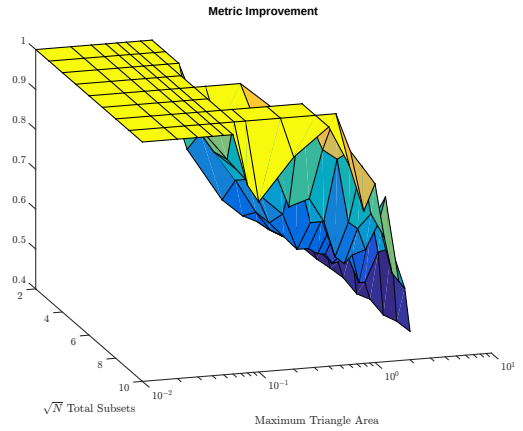

Fig. 16. The difference in metric behavior of the third test case with no iteration and 10 iterations. The closer the z-value to zero, the better the improvement.

load balancing iterations. The maximum metric is 2.2660 and occurs when Fig. 7 is run with 10x10 subsets with a maximum triangle area of 0.4 cm^2. The minimum metric is 1.0021 and occurs when Fig. 7 is run with 2x2 subsets with the Triangle's coarsest possible mesh.

Figure 16 shows the difference in metric behavior for Fig. 7. The maximum improvement has a value of 0.4476 and occurs for Fig. 7 is run with 2x2 subsets with Triangle's coarsest possible mesh generation settings. The minimum improvement has a value of very close to 1.0 and occurs for many of the inputs.

Because Fig. 7 has more features and is more symmetric of a problem, the initial load balancing metric will not be as large as the load balancing metric of Figs. 6 and 5. As a result, the improvement in the load balancing metric after 10 iterations will not be as great in problems similar to Fig. 7.

Good improvement is seen throughout all three test cases for all three inputs, particularly the first two test cases, which were initially very unbalanced. However, there were many inputs run that had problems with $f > 1.1$, which means many problems were unbalanced by more than 10%. The user will not always have the luxury of choosing the number of subsets they want the problem run with, as this directly affects the number of processors the problem will be run with. Certain problems will require more processors and will require minimizing the total number of cells in the domain for the problem to complete running in a reasonable amount of time. As a result, improvements to the algorithm must be made.

This can be done by changing how the cut lines are redistributed. Instead of changing entire row and column widths, the cut lines can be moved on the subset level. However, this can sacrifice the strict orthogonality that PDT currently utilizes to scale so well on a massively parallel scale[1]. Changes to the performance model and the scheduler would have to be made.

Another option is to implement domain overloading[1], which is the logical extension of the work presented in this paper. This would involve processors owning different numbers of subsets, with no restriction on these subsets being contiguous. This would be the most effective method at perfecting this algorithm, and would lead to less problems being unbalanced by more than 10%.

3. Solution Verification

For solution verification, two simple problems were chosen: a 1D pure absorber slab and a 1D pure scatterer slab. These problems were chosen because their analytical solutions are easily obtained, thus making a comparison between PDT's solution and the analytical solution easy and informative. The same geometry and mesh were used for both problems, and are shown in Figure 17.

The problem geometry is a 1 cm by 1 cm square. In order to simulate a 1D slab, opposing reflecting boundaries were placed on the both y boundaries, effectively forcing the problem to be infinite in the y direction. At the x minimum boundary, an incident isotropic flux was used, and a vacuum boundary was enforced at the x max boundary. No source was used in either problem.

In order to measure how close the numerical and analytical solutions an error estimate, represented by, Eq. (18), is used:

$$\epsilon = \sqrt{\sum_{k} \sum_{q} w_q \cdot (\phi_h(x_q) - \phi_e(x_q))^2} \qquad (18)$$

where k is the number of cells, q is the number points per cell used in a Gauss-Legendre quadrature, ϕ_h is the numerical flux solution and ϕ_e is the analytical flux solution.

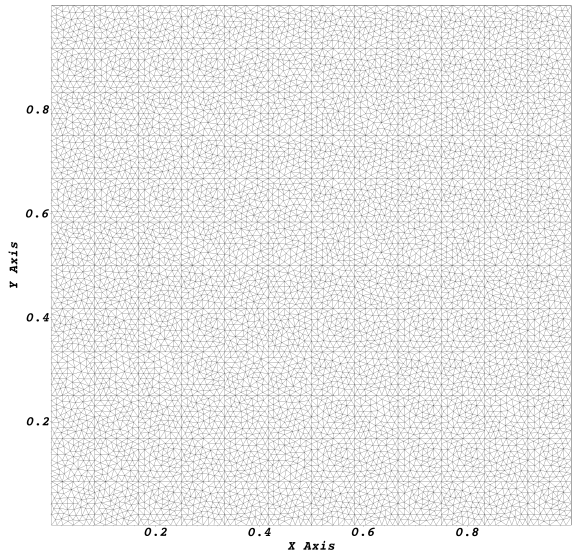

Fig. 17. The geometry and mesh used in solution verification problems.

4. The 1D Pure Absorber Slab

For monoenergetic neutrons, a source free, 1D pure absorber slab, the transport equation is represented by Eq. (19):

$$\mu \frac{d\psi(x,\mu>0)}{dx} + \Sigma_a \psi(x,\mu>0) = 0, \tag{19}$$

where ψ is the angular flux, Σ_a is the macroscopic absorption cross section, and μ is the cosine of the polar angle. The boundary conditions for this problem are expressed in Eq. (20):

$$\psi(0,\mu>0) = \int_0^{2\pi} d\gamma \int_0^1 \frac{\psi_0}{4\pi} d\mu = \frac{\psi_0}{2} = \psi_{inc} \text{ (incident isotropic)}$$
$$\psi(x_{\max},\mu<0) = 0 \text{ (vacuum)}, \tag{20}$$

where ψ_0 is the user defined value of the incident isotropic angular flux, and ψ_{inc} is the angular flux at $x = 0$. Equation (21) solves the transport equation via separation of variables to get the angular flux for this problem:

$$\begin{aligned}
\frac{d\psi(x,\mu>0)}{dx} &= -\frac{\Sigma_a}{\mu}x \\
\frac{d\psi(x,\mu>0)}{\psi(x,\mu>0)} &= -\frac{\Sigma_a}{\mu}x dx \\
\int_{\psi(0,\mu>0)}^{\psi(x,\mu>0)} \frac{d\psi(x,\mu>0)}{\psi(x,\mu>0)} &= \int_0^x -\frac{\Sigma_a}{\mu}x' dx' \\
\ln[\frac{\psi(x,\mu>0)}{\psi(0,\mu>0)}] &= -\frac{\Sigma_a}{\mu}x \\
\psi(x,\mu>0) &= \psi(0,\mu>0)\exp(-\frac{\Sigma_a}{\mu}x) \\
\psi(x,\mu>0) &= \psi_{inc}\exp(-\frac{\Sigma_a}{\mu}x)
\end{aligned} \tag{21}$$

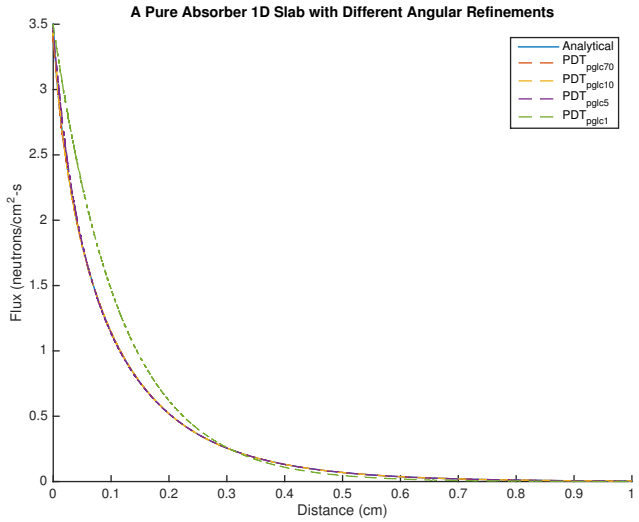

Fig. 18. The pure absorber solution with four different angular refinements.

Using the fact that the scalar flux in this pure absorber is simply the angular flux integrated for $\mu > 0$, the scalar flux with our boundary conditions is represented by Eq. (22):

$$\begin{aligned}
\phi(x) &= \int_0^1 \psi(x,\mu>0)d\mu \\
&= \int_0^1 \psi_{inc}\exp(-\frac{\Sigma_a}{\mu}x)d\mu = \psi_{inc}E_2(\Sigma_a x),
\end{aligned} \tag{22}$$

where ϕ is the scalar flux and E_2 is the exponential integral function with $n = 2$.

The pure absorber was run with $\psi_{inc} = 3.5\frac{\text{n}}{\text{cm}^2\text{-s-ster}}$ and $\Sigma_a = 5\ \text{cm}^{-1}$. Figure 18 shows a comparison of the analytical solution with PDT's solution for four different angular refinements. All four PDT runs used only 1 azimuthal angle per quadrant, but varied the number of positive polar angles, because the problem is not azimuthally dependent. The number of positive polar angles used were 1,5,10, and 70.

It is immediately clear that not many polar angles are necessary for agreement with the analytical solution. Figure 19 examines the 70 positive polar angle case exclusively in comparison with the analytical solution. It is immediately clear graphically that the solutions are in agreement. Table I shows the error convergence as average mesh size in the problem decreases. This data shows that the error shows first order convergence, which is not the expected second order convergence. In the future, more work will be done to analyze and potentially correct the error calculation for unstructured meshes and to investigate why we don't see the second order convergence that is expected.

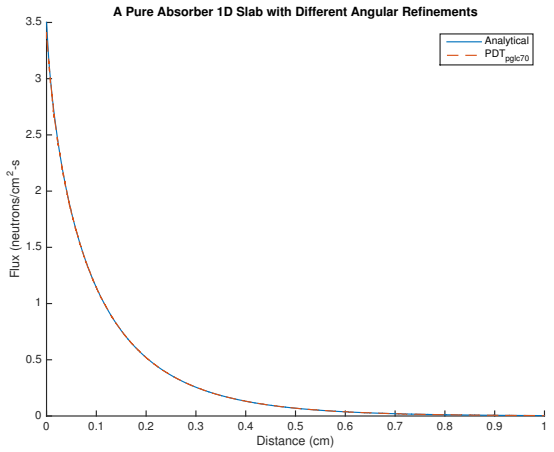

Fig. 19. The pure absorber solution run with 70 positive polar angles.

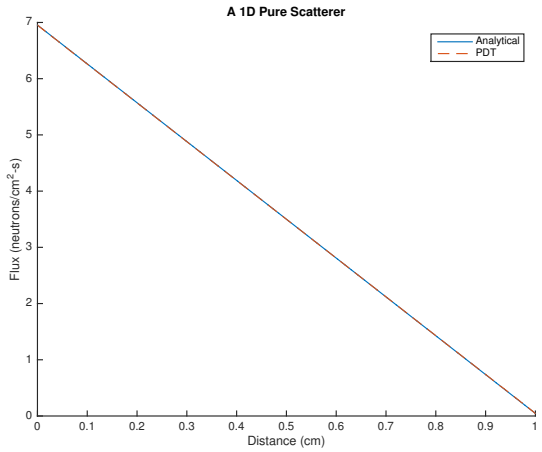

Fig. 20. The pure scatterer solution run with 40 positive polar angles.

TABLE I. The convergence of the error as the number of cells increases.

Mesh size (cm)	ϵ
1/4	0.3035
1/8	0.1865
1/16	0.09168
1/32	0.0387
1/64	0.0169
1/128	0.006963

A. The 1D Pure Scatterer Slab

no sure we need to spent that much time on solution verification. this feels too long For an optically thick, source free 1D pure absorber with monoenergetic neutrons, the transport solution will reach the diffusion limit. The diffusion equation for this problem is represented by Eq. (23):

$$\frac{d^2\phi}{dx^2} = 0, \tag{23}$$

where ϕ is the scalar flux. The boundary conditions for this problem are expressed in Eq. (24):

$$\begin{aligned} \phi(-2D) &= 4j_{inc} \\ \phi(x_{\max} + 2D) &= 0, \end{aligned} \tag{24}$$

where j_{inc} is the incident partial current and D is the diffusion coefficient, which is equivalent to $\frac{1}{3\Sigma_t}$, where Σ_t is the total macroscopic cross section. The first boundary condition is the extrapolated boundary condition, and the second is the extrapolated vacuum condition. The incident partial current is calculated from the incident angular flux, as shown in Eq. (25):

$$j_{inc} = \int_0^{2\pi} d\gamma \int_0^1 \mu \frac{\psi_{inc}}{4\pi} d\mu = \frac{\psi_{inc}}{4}. \tag{25}$$

The integral over polar angles in Eq. (25) is the result of computing the angular quadrature with an infinite number of polar angles. Table II shows the value of j_{inc} converging to the integral value as the number of polar angles is increased.

TABLE II. The convergence of j_{inc} as the number of polar angles increase.

Number of Positive Polar Angles	j_{inc}
1	2.0207
2	1.8244
5	1.7632
10	1.7534
20	1.7509
40	1.7502
Infinite	1.750

Equation (26) solves Eq. (23):

$$\begin{aligned} \frac{d\phi(x)}{dx} &= A \\ \phi(x) &= Ax + B, \end{aligned} \tag{26}$$

where A and B are integration constants. Using our boundary conditions in Eq (24) to solve for A and B:

$$\begin{aligned} \phi(-2D) &= -2DA + B = 4j_{inc} \\ \phi(x_{\max} + 2D) &= A(x_{\max} + 2D) + B = 0, \end{aligned}$$

the scalar flux, represented by Eq. (27), is:

$$\phi(x) = \frac{4j_{inc}}{1 + 4D}(-x + x_{\max} + 2D). \tag{27}$$

This problem was run with $\Sigma_t = 100\,\text{cm}^{-1}$ and $j_{inc} = \frac{7}{4}\frac{\text{n}}{\text{cm}^2\text{-s}}$. Figure 20 shows the agreement between the analytical solution and PDT's solution. An angular refinement of 40 polar angles was used, with one azimuthal angle in each quadrant.

It is immediately clear graphically that the two solutions are in agreement, and the relative error of the numerical solution is 4.25E-04, as defined by Eq. (18).

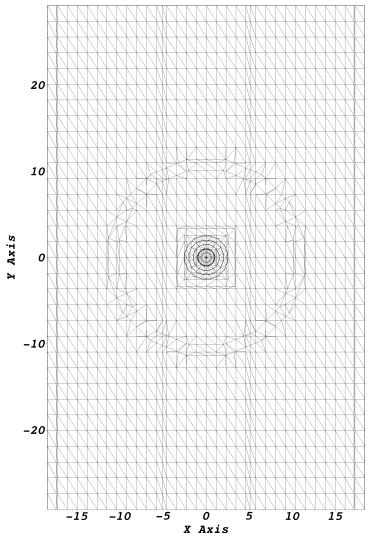

Fig. 21. The 2D mesh of the IM1 problem.

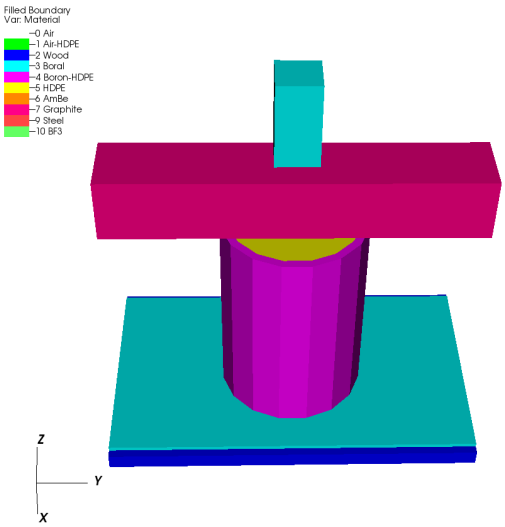

Fig. 22. The 2D extruded view of the IM1 problem.

V. 2D AND 2D EXTRUDED MESHING CAPABILITY

I do not like showcase. use illustrate. it's better To showcase, the newly implemented unstructured meshing capability in PDT, Texas A&M Nuclear Engineering's Impurity Model 1 (IM1) problem is used. Figure 21 showcases the 2D mesh of the IM1 problem,

In order to get from the 2D mesh to the 2D extruded mesh, an extrusion file is supplied to PDT. This extrusion file supplies two critical pieces of information: the number of z layers and their locations, and how each region of the 2D mesh is mapped to these z layers. The combination of the 2D mesh and the extrusion file yield the full 3D problem, shown in Fig. 22.

we need to put some results here. this needs to be a bit more quantitative. and you need to cut elsewhere. this is too long. I think section II is too long and the verification is too long as well

VI. CONCLUSIONS

not read yet. if you feel you need to amend based on my comments, go ahead

In conclusion, the load balancing algorithm outlined in the Motivation and Methods section works well for more symmetric problems with a lot of features, and even works well for particularly unbalanced problems. As shown in the results, its effectiveness depends on the maximum triangle area used, and the number of subsets the user chooses to decompose the problem domain into.

Good improvement is seen throughout all three test cases for all three inputs, particularly the first two test cases, which were initially very unbalanced. However, there were many inputs run that had problems with $f > 1.1$, which means many problems were unbalanced by more than 10%. The user will not always have the luxury of choosing the number of subsets they want the problem run with, as this directly affects the number of processors the problem will be run with. Certain problems will require more processors and will require minimizing the total number of cells in the domain for the problem to complete running in a reasonable amount of time. As a result, improvements to the algorithm must be made.

This can be done by changing how the cut lines are redistributed. Instead of changing entire row and column widths, the cut lines can be moved on the subset level. However, this can sacrifice the strict orthogonality that PDT currently utilizes to scale so well on a massively parallel scale[1]. Changes to the performance model and the scheduler would have to be made. This method is currently being implemented.

Another option is to implement domain overloading[**?**], which is the logical extension of the work presented in this paper. This would involve processors owning different numbers of subsets, with no restriction on these subsets being contiguous. This would be the most effective method at perfecting this algorithm, and would lead to less problems being unbalanced by more than 10%.

VII. ACKNOWLEDGMENTS

This material is based upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002376.

REFERENCES

1. M. A. ET AL, "Provably Optimal Parallel Transport Sweeps with Non-Contiguous Partitions," in "ANS MC2015-Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method," (2015).
2. J. SHEWCHUK, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," in M. C. LIN and D. MANOCHA, editors, "Applied Computational Ge-

ometry: Towards Geometric Engineering," Springer-Verlag, *Lecture Notes in Computer Science*, vol. 1148, pp. 203–222 (May 1996), from the First ACM Workshop on Applied Computational Geometry.

3. M. A. ET AL, "Provably Optimal Parallel Transport Sweeps on Regular Grids," in "International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering," (2013).
4. T. E. ET AL., "Denovo: A New Three-Dimensional Parallel Discrete Ordinates Code in Scale," *Nuclear Technology*, **171**, 171–200 (2010).
5. R. ALCOUFFE, R. BAKER, S. TURNER, and J. DAHL, "PARTISN manual," Tech. rep., LA-UR-02-5633, Los Alamos National Laboratory (2002).
6. S. PAUTZ, "An Algorithm For Parallel Sn Sweeps on Unstructured Meshes," *Los Alamos National Laboratory Publications*, **LA-UR-01-1420**.