

d

LOAD BALANCING UNSTRUCTURED MESHES FOR MASSIVELY PARALLEL
TRANSPORT SWEEPS

A Thesis

by

TAREK HABIB GHADDAR

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Chair of Committee,	Dr. Jean Ragusa
Committee Members,	Dr. Jim Morel
	Dr. Bojan Popov
Head of Department,	Dr. Yassin Hassan

May 2016

Major Subject: Nuclear Engineering

Copyright 2016 Tarek Habib Ghaddar

ABSTRACT

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer lectus quam, condimentum quis bibendum eu, sollicitudin eget lacus. Praesent non sodales odio. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla ac luctus sapien. Morbi cursus sapien eget lorem fermentum hendrerit. Nam ac erat dui, in cursus velit. Vivamus hendrerit porttitor nisi, ut porttitor lorem volutpat eget. In ligula ligula, euismod ut condimentum sit amet, pulvinar sit amet diam. Pellentesque interdum, ipsum ullamcorper consequat dignissim, sem arcu egestas mauris, vitae interdum sem tortor ut ante. Nunc blandit laoreet nisi, non rutrum lorem hendrerit quis. Cras nunc diam, convallis et feugiat at, auctor id libero. Nunc facilisis massa eu eros imperdiet vestibulum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec non velit vitae tortor blandit semper.

Etiam vitae dolor nulla. Ut eros odio, rhoncus eget placerat vitae, elementum ac ante. Proin vitae odio eu nisl pharetra mattis. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Phasellus fermentum lacus consectetur neque consequat ullamcorper. Cras blandit urna non dui consequat molestie. Curabitur viverra nibh at nisi semper faucibus. Nam egestas mauris a enim dignissim nec consectetur tortor rutrum. Mauris at nisi in est luctus congue ut mattis est. Ut pretium, mi quis elementum cursus, ante eros suscipit ligula, ut porttitor elit leo sed turpis. Nam sed dui ligula.

ACKNOWLEDGEMENTS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer lectus quam, condimentum quis bibendum eu, sollicitudin eget lacus. Praesent non sodales odio. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla ac luctus sapien. Morbi cursus sapien eget lorem fermentum hendrerit. Nam ac erat dui, in cursus velit. Vivamus hendrerit porttitor nisi, ut porttitor lorem volutpat eget. In ligula ligula, euismod ut condimentum sit amet, pulvinar sit amet diam. Pellentesque interdum, ipsum ullamcorper consequat dignissim, sem arcu egestas mauris, vitae interdum sem tortor ut ante. Nunc blandit laoreet nisi, non rutrum lorem hendrerit quis. Cras nunc diam, convallis et feugiat at, auctor id libero. Nunc facilisis massa eu eros imperdiet vestibulum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec non velit vitae tortor blandit semper.

Etiam vitae dolor nulla. Ut eros odio, rhoncus eget placerat vitae, elementum ac ante. Proin vitae odio eu nisl pharetra mattis. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Phasellus fermentum lacus consectetur neque consequat ullamcorper. Cras blandit urna non dui consequat molestie. Curabitur viverra nibh at nisi semper faucibus. Nam egestas mauris a enim dignissim nec consectetur tortor rutrum. Mauris at nisi in est luctus congue ut mattis est. Ut pretium, mi quis elementum cursus, ante eros suscipit ligula, ut porttitor elit leo sed turpis. Nam sed dui ligula.

NOMENCLATURE

B/CS	Bryan/College Station
HSUS	Humane Society of the United States
P	Pressure
T	Time
TVA	Tennessee Valley Authority
TxDOT	Texas Department of Transportation

This page is optional.

TABLE OF CONTENTS

	Page
ABSTRACT	ii
ACKNOWLEDGEMENTS	iii
NOMENCLATURE	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	vi
LIST OF TABLES	vii
1. INTRODUCTION	1
2. THE TRANSPORT EQUATION	1
3. PARALLELIZATION OF TRANSPORT SWEEPS	1
3.1 KBA Partitioning for Structured Grids	3
3.2 The Structured Transport Sweep in PDT	4
3.3 The Unstructured Transport Sweep	7
3.3.1 Cycle Detection	9
4. MOTIVATION AND METHODS	1
5. RESULTS	1
5.1 2D and 2D Extruded Meshing Capability	1
5.2 Test Cases for Metric Behavior and Convergence	1
5.3 Metric Behavior and Convergence	4
5.4 Solution Verification	13

LIST OF FIGURES

FIGURE	Page
2.1 A demonstration of a sweep on a structured and unstructured mesh. . . .	4
2.2 A task dependence graph of the unstructured mesh example in Fig. ?? . . .	5
3.1 Three different partitioning schemes in 2D, from left to right: KBA, volumetric non-overloaded, and volumetric overloaded.	5
4.1 A PSLG describing a fuel lattice, and with a orthogonal “subset” grid imposed on on the PSLG.	1
5.1 The first .poly file used in order to test effectiveness and convergence of the load balancing metric.	3
5.2 The second .poly file used in order to test effectiveness and convergence of the load balancing metric.	3
5.3 The third .poly file used in order to test effectiveness and convergence of the load balancing metric.	4
5.4 The metric behavior of the first test case run with no load balancing iterations.	5
5.5 The metric behavior of the first test case run with 10 load balancing iterations.	6
5.6 The difference in metric behavior between no iteration and 10 iterations. The closer the z-value to zero, the better the improvement.	7
5.7 The metric behavior of the second test case run with no load balancing iterations.	9
5.8 The metric behavior of the second test case run with 10 load balancing iterations.	11
5.9 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.	12

LIST OF TABLES

TABLE	Page
5.1 The parameters of the 162 inputs for each case.	2

1. INTRODUCTION

When running any massively parallel code, 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 important in order to minimize idle time for all processors by equally distributing (as much as possible) the work each processor has to do.

The concepts and results presented in this thesis are used by PDT, Texas A&M University's massively parallel deterministic transport code. It is capable of multi-group simulations and employs discrete ordinates for angular discretization. PDT features steady-state, time-dependent, criticality, and depletion simulations. It solves the transport equation for neutron, thermal, gamma, coupled neutron-gamma, electron, and coupled electron-photon radiation. PDT has been shown to scale on logically Cartesian grids out to 750,000 cores. Logically Cartesian grids contain regular convex grid units that allow for vertex motion inside them, in order to conform to curved shapes.

The following are the since completed goals of this thesis:

- Implement unstructured meshing capability in PDT for 2D and 2D extruded problems.
- Generate unstructured mesh in parallel using the same partitioning scheme and number of processors as the sweep.
- Perform stitching between meshed subdomains to preserve interface continuity.
- Implement load balancing algorithm for unstructured meshes in PDT. A load balanced problem would be defined by f in the pseudocode, or the subset with the maximum number of cells, divided by the average number of cells per subset.
- Verify and test code to prove load balancing algorithm effectiveness.

- Show results of the parallel transport sweeps for benchmark problems.

The unstructured meshing capability being added into PDT was of paramount importance because it allows the user to define the problem geometry without having to conform the mesh to the problem geometry. The need to move vertices within convex grids is no longer needed, so the user only needs to define the geometry of the problem. The unstructured mesh is created utilizing the Triangle mesh generator.

2. THE TRANSPORT EQUATION

The steady-state neutron transport equation describes the behavior of neutrons in a medium and is given by Eq. (2.1):

$$\vec{\Omega} \cdot \vec{\nabla} \psi(\vec{r}, E, \vec{\Omega}) + \Sigma_t(\vec{r}, E) \psi(\vec{r}, E, \vec{\Omega}) = \int_0^\infty dE' \int_{4\pi} d\Omega' \Sigma_s(\vec{r}, E' \rightarrow E, \Omega' \rightarrow \Omega) \psi(\vec{r}, E', \vec{\Omega}') + S_{ext}(\vec{r}, E, \vec{\Omega}), \quad (2.1)$$

where $\vec{\Omega} \cdot \vec{\nabla} \psi$ is the leakage term and $\Sigma_t \psi$ is the total collision term (absorption, outscatter, and within group scattering). These are the loss terms of the neutron transport equation. The right hand side of Eq. (2.1) represents the gain terms, where S_{ext} is the external source of neutrons and $\int_0^\infty dE' \int_{4\pi} d\Omega' \Sigma_s(E' \rightarrow E, \Omega' \rightarrow \Omega) \psi(\vec{r}, E', \vec{\Omega}')$ is the inscatter term, which represents all neutrons scattering from energy E' and direction $\vec{\Omega}'$ into energy dE about energy E and $d\Omega$ about direction $\vec{\Omega}$.

Without loss of generality for the research problem at hand, we assume isotropic scattering for simplicity. The double differential scattering cross section, $\Sigma_s(E' \rightarrow E, \Omega' \rightarrow \Omega)$, no longer depends on direction and is divided by 4π to reflect isotropic behavior. This yields:

$$\begin{aligned} \vec{\Omega} \cdot \vec{\nabla} \psi(\vec{r}, E, \vec{\Omega}) + \Sigma_t(\vec{r}, E) \psi(\vec{r}, E, \vec{\Omega}) &= \frac{1}{4\pi} \int_0^\infty dE' \Sigma_s(\vec{r}, E' \rightarrow E) \int_{4\pi} d\Omega' \psi(\vec{r}, E', \vec{\Omega}') + S_{ext}(\vec{r}, E, \vec{\Omega}) \\ &= \frac{1}{4\pi} \int_0^\infty dE' \Sigma_s(\vec{r}, E' \rightarrow E) \phi(\vec{r}, E') + S_{ext}(\vec{r}, E, \vec{\Omega}), \end{aligned} \quad (2.2)$$

where we have introduced the scalar flux as the integral of the angular flux:

$$\phi(\vec{r}, E') = \int_{4\pi} d\Omega' \psi(\vec{r}, E', \vec{\Omega}'). \quad (2.3)$$

The next step to solving the transport equation is to discretize in energy, yielding Eq. (2.4) the multigroup transport equation:

$$\vec{\Omega} \cdot \vec{\nabla} \psi_g(\vec{r}, \vec{\Omega}) + \Sigma_{t,g}(\vec{r}) \psi_g(\vec{r}, \vec{\Omega}) = \frac{1}{4\pi} \sum_{g'} \Sigma_{s,g' \rightarrow g}(\vec{r}) \phi_{g'}(\vec{r}) + S_{ext,g}(\vec{r}, \vec{\Omega}), \quad \text{for } 1 \leq g \leq G \quad (2.4)$$

where the multigroup transport equations now form a system of coupled equations.

Next, we discretize in angle using the discrete ordinates method[?], whereby an angular quadrature $(\vec{\Omega}_m, w_m)_{1 \leq m \leq M}$ is used to solve the above equations along a given set of directions $\vec{\Omega}_m$:

$$\vec{\Omega}_m \cdot \vec{\nabla} \psi_{g,m}(\vec{r}) + \Sigma_{t,g}(\vec{r}) \psi_{g,m}(\vec{r}) = \frac{1}{4\pi} \sum_{g'} \Sigma_{s,g' \rightarrow g}(\vec{r}) \phi_{g'}(\vec{r}) + S_{ext,g,m}(\vec{r}), \quad (2.5)$$

where the subscript m is introduced to describe the angular flux in direction m . We notice that the subscript is not added to our inscatter term because of the isotropic scattering assumption and because the scalar flux does not depend on angle. However, in order to evaluate the scalar flux, we employ the angular weights w_m and the angular flux solutions ψ_m to numerically perform the angular integration:

$$\phi_g(\vec{r}) \approx \sum_{m=1}^{m=M} w_m \psi_{g,m}(\vec{r}). \quad (2.6)$$

From Equation (2.4), it is clear that we are solving a sequence of transport equations, one equation per group. Therefore, all transport equations are of the following form:

$$\vec{\Omega}_m \cdot \vec{\nabla} \psi_m(\vec{r}) + \Sigma_t(\vec{r}) \psi_m(\vec{r}) = \frac{1}{4\pi} \Sigma_s(\vec{r}) \phi(\vec{r}) + q_m^{ext+inscat}(\vec{r}) = q_m(\vec{r}), \quad (2.7)$$

where the group index notation is omitted for brevity.

In order to obtain the solution for this discrete form of the transport equation, an iter-

ative process called source iteration is introduced. This is shown by a simplified transport equation Eq. (2.8):

$$\vec{\Omega}_m \cdot \vec{\nabla} \psi_m^{(l+1)}(\vec{r}) + \Sigma_t \psi_m^{(l+1)}(\vec{r}) = q_m^{(l)}(\vec{r}), \quad (2.8)$$

where the right hand side terms of Eq. (2.5) 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. (2.8) must be discretized in space as well. This is done by meshing the domain and solving the spatial problem one cell at a time for a given direction. 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. Sweeping the mesh and solving one cell at a time is possible utilizing one of three popular discretization techniques: finite difference[?], finite volume[?], or discontinuous finite element[?]. Figure 2.1 shows the sweep ordering for a given direction on both a structured and unstructured mesh.

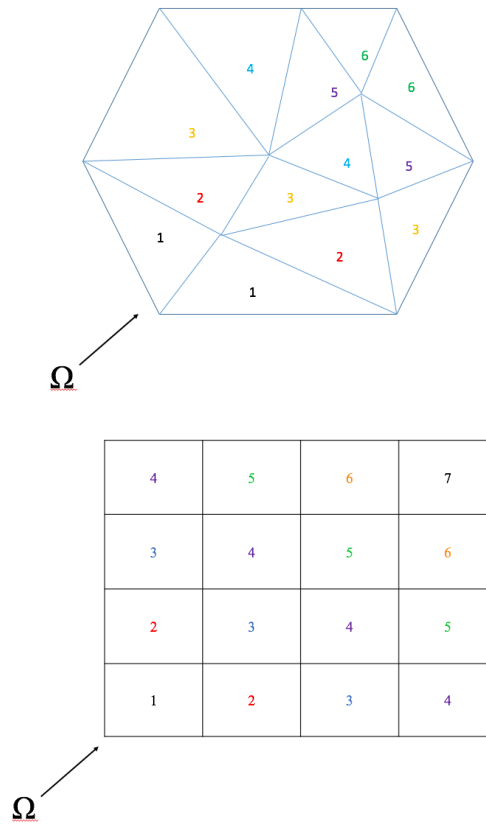


Figure 2.1: A demonstration of a sweep on a structured and unstructured mesh.

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 task dependence graph, shown in Fig. 2.2.

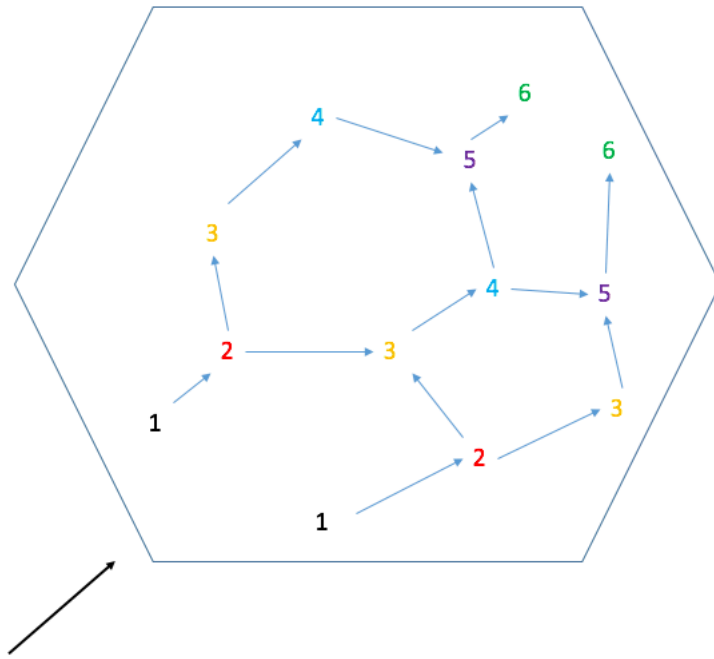


Figure 2.2: A task dependence graph of the unstructured mesh example in Fig. 2.1.

3. PARALLELIZATION OF TRANSPORT SWEEPS

As mentioned in the previous section, a transport sweep is set up by overlaying a domain with a finite element mesh. The sweep then solves the transport equation cell by cell using a discontinuous finite element approach. The order of which cell to solve first is given by a task dependence graph, as shown in Fig. 2.2. The transport sweep can be solved in parallel in order to obtain the solution faster, as well as distribute the memory to many processors for memory intensive cases. In PDT, a transport sweep can be performed on a structured Cartesian mesh, and the work proposed utilizes transport sweeps on an unstructured mesh. Performing a transport sweep on an unstructured mesh presents two big challenges: performing a transport sweep on a massively parallel scale in an efficient manner and keeping non-concave sub-domains due to the nature of the transport sweep itself. PDT has already proven the ability to perform massively parallel transport sweeps on structured meshes. As part of previous efforts in PDT, researchers have come to outline three important properties for parallel sweeps.

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

- partitioning: dividing the domain among available processors
- 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 (3.1) approximately defines parallel sweep efficiency. This can be calculated for specific machinery and partitioning parameters by substituting in values calculated

using Eqs. (3.5), (3.6), and (3.7).

$$\begin{aligned}\varepsilon &= \frac{T_{\text{task}}N_{\text{tasks}}}{[N_{\text{stages}}][T_{\text{task}} + T_{\text{comm}}]} \\ &= \frac{1}{[1 + \frac{N_{\text{idle}}}{N_{\text{tasks}}}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]}\end{aligned}\tag{3.1}$$

Equations (3.2) and 3.3 show how T_{comm} and T_{task} are calculated:

$$T_{\text{comm}} = M_L T_{\text{latency}} + T_{\text{byte}} N_{\text{bytes}}\tag{3.2}$$

$$T_{\text{task}} = A_x A_y A_z A_m A_g T_{\text{grind}}\tag{3.3}$$

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 model to match computational results, improvements should be made in code implementation.

3.1 KBA Partitioning for Structured Grids

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

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:

$$\epsilon_{KBA} = \frac{1}{\left[1 + \frac{4(P_x + P_y - 2)}{8MN_k}\right] \left[1 + \frac{T_{\text{comm}}}{T_{\text{task}}}\right]} \quad (3.4)$$

3.2 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 (3.5), (3.6), and (3.7) define N_{fill} , N_{idle} , and N_{tasks} :

$$N_{\text{fill}} = \frac{P_x + \delta_x}{2} - 1 + \frac{P_y + \delta_y}{2} - 1 + N_k \left(\frac{P_z + \delta_z}{2} - 1 \right) \quad (3.5)$$

$$N_{\text{idle}} = 2N_{\text{fill}} \quad (3.6)$$

$$N_{\text{tasks}} = 8N_m N_g N_k \quad (3.7)$$

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

Figure 3.1 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 cells owned by a processor are contiguous, where as volumetric non-overloaded partitioning does not have this restriction.

9	10	11	12	13	14	15	16	11	12	15	16	11	12	11	12	15	16	15	16
9	10	11	12	13	14	15	16					9	10	9	10	13	14	13	14
9	10	11	12	13	14	15	16	9	10	13	14	11	12	11	12	15	16	15	16
9	10	11	12	13	14	15	16					9	10	9	10	13	14	13	14
1	2	3	4	5	6	7	8	3	4	7	8	3	4	3	4	7	8	7	8
1	2	3	4	5	6	7	8					1	2	1	2	5	6	5	6
1	2	3	4	5	6	7	8	1	2	5	6	3	4	3	4	7	8	7	8
1	2	3	4	5	6	7	8					1	2	1	2	5	6	5	6

Figure 3.1: Three different partitioning schemes in 2D, from left to right: KBA, volumetric non-overloaded, and volumetric overloaded.

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 $P_x P_y P_z = P$, 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 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 situations. At large processor counts, the product $\omega_m \omega_g$ must be large, which requires $N_m N_g$ 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. (3.8).

$$\epsilon_{opt} = \frac{1}{\left[1 + \frac{P_x + P_y + P_z - 6}{\omega_g \omega_m \omega_r}\right] \left[1 + \frac{T_{comm}}{T_{task}}\right]} \quad (3.8)$$

3.3 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 specifically we initially subdivide the domain into subsets, which are just rectangular subdomains. Within each subset, an unstructured mesh is created. This creates a pseudo-regular grid. These subsets become the N_x, N_y, N_z equivalent for an unstructured 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)[?] .

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 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. Further work is being done for scaling to thousands of processors.

3.3.1 Cycle Detection

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.

4. 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 4.1 demonstrates this functionality, with the first two subsets meshed using the Triangle Mesh Generator[?], a 2D mesh generator.

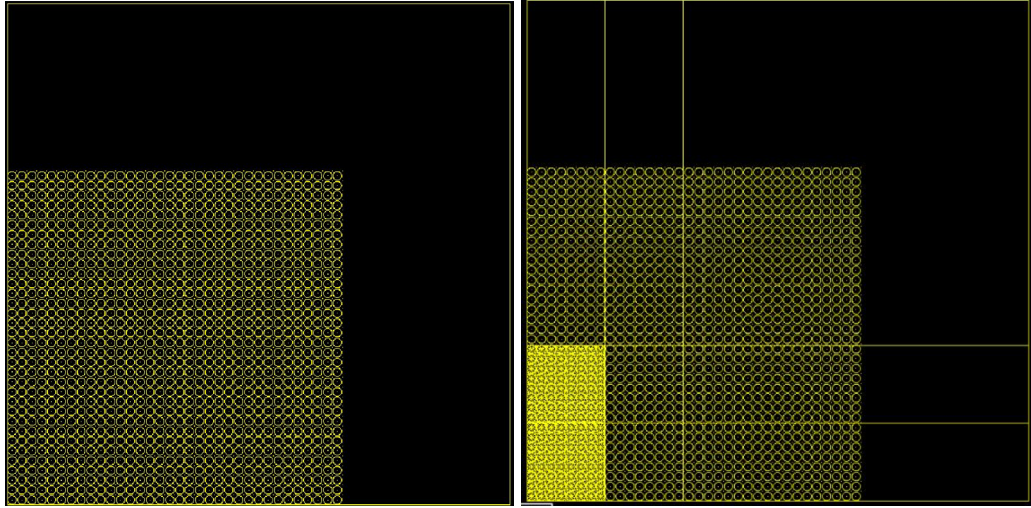


Figure 4.1: A PSLG describing a fuel lattice, and with a orthogonal “subset” grid imposed on on the PSLG.

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). 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 a unstructured tetrahedral mesh, but for many problems, it is a great capability to have.

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 processors by equally distributing (as much as possible) the work each processor has to do. For the purposes of unstructured meshes in PDT, we are looking to “balance” the number of cells. Ideally, each processor 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 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.

Load Balance: A load balancing algorithm that equalizes the number of triangles per subset.

I, J subsets specified by user

Mesh all subsets

N_{tot} = total number of triangles

N_{ij} = number of triangles in subset ij

$f = \max_{ij} (N_{ij}) / \frac{N_{tot}}{I \cdot J}$ { //Check if all subsets meet the tolerance }

if $f < \text{tol}_{\text{subset}}$ **then**

 DONE with load balancing

else

$f_I = \max_i [\sum_j N_{ij}] / \frac{N_{tot}}{I}$

$f_J = \max_j [\sum_i N_{ij}] / \frac{N_{tot}}{J}$

if $f_I > \text{tol}_{\text{row}}$ **then**

Redistribute(X_i)

end if

if $f_J > \text{tol}_{\text{col}}$ **then**

Redistribute(Y_j)

end if

if redistribution occurred **then**

 REMESH and repeat algorithm

end if

end if

if There is still a discrepancy amongst subsets **then**

 Move cutplane segments on the subset level and remesh (may require changes to scheduling algorithm)

end if

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, a pArray containing number of triangles in each row or column

Input: The total number of triangles in the domain, N_{tot}

stapl::array_view num_tri_view, over num_tri_row/column

stapl::array_view offset_view

stapl::partial_sum(num_tri_view) {Perform prefix sum} {We now have a cumulative distribution stored in offset_view}

for $i = 1 : \text{CutLines.size()} - 1$ **do**

vector <double> pt1 = [CutLines(i-1), offset_view(i-1)]

vector <double> pt2 = [CutLines(i), offset_view(i)]

ideal_value = $i \cdot \frac{N_{tot}}{\text{CutLines.size()} - 1}$

X-intersect(pt1, pt2, ideal_value) {Calculates the X-intersect of the line formed by pt1 and pt2 and the line $y = \text{ideal_value}$.}

CutLines(i) = X-intersect

end for

5. RESULTS

give the dimension of your domains. I suppose since XS are in 1/cm, the domain and the trinalge areas are in cm² but say it

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

5.1 2D and 2D Extruded Meshing Capability

not sure what you want to start with what could be the best results. Why not build up momentum and show this as the grand finale?

Figure ?? showcases the 2D meshing capability, and Fig. ?? showcases the same mesh extruded in the z direction.

5.2 Test Cases for Metric Behavior and Convergence

In order to showcase the behavior of the load balancing metric it would be good to provide the equation number for the metric, for clarity, three test cases are presented. Figure 5.1 shows the first test case, a large domain with two pins in opposite corners of the domain. Figure 5.2 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 5.3 shows a lattice and reflector, which due to its denser and repeated geometry, theoretically is a more balanced problem.

A series of 162 inputs was constructed for each case. Table 5.1 shows the the parameters that will change in each input. The maximum triangle area varied from the coarsest possible to 0.01, and the number of subsets, N , varies from 2x2 to 10x10.

Table 5.1: The parameters of the 162 inputs for each case.

Area test is coarse so fuck it N = 4 N = 9 N = 16 N = 25 N = 36 N = 49 N = 64 N = 81

Coarse

1.8

1.6

1.4

1.2

1

0.8

0.6

0.4

0.2

0.1

0.08

0.06

0.05

0.04

0.03

0.02

0.01

do not call first and second poly files. Be creative, come up with names for the examples, be it only Example 1,2,3, or Example I,II,III, or Example A, B, C

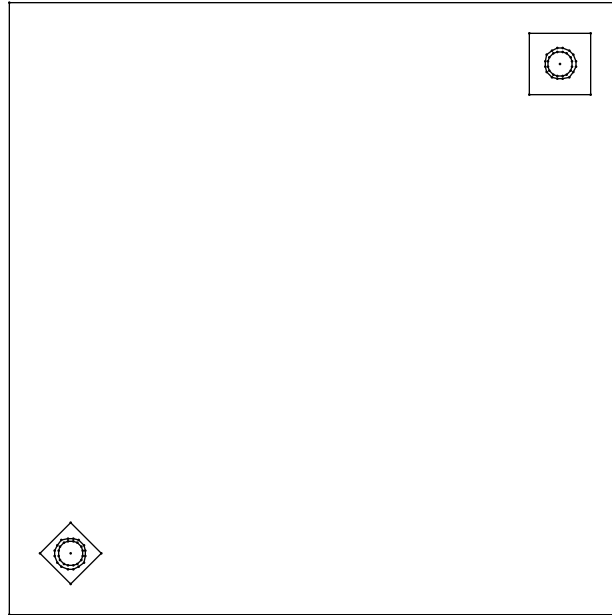


Figure 5.1: The first .poly file used in order to test effectiveness and convergence of the load balancing metric.

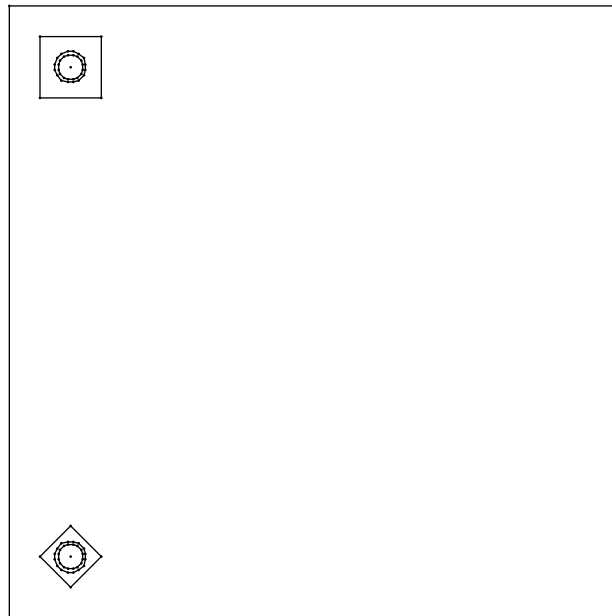


Figure 5.2: The second .poly file used in order to test effectiveness and convergence of the load balancing metric.

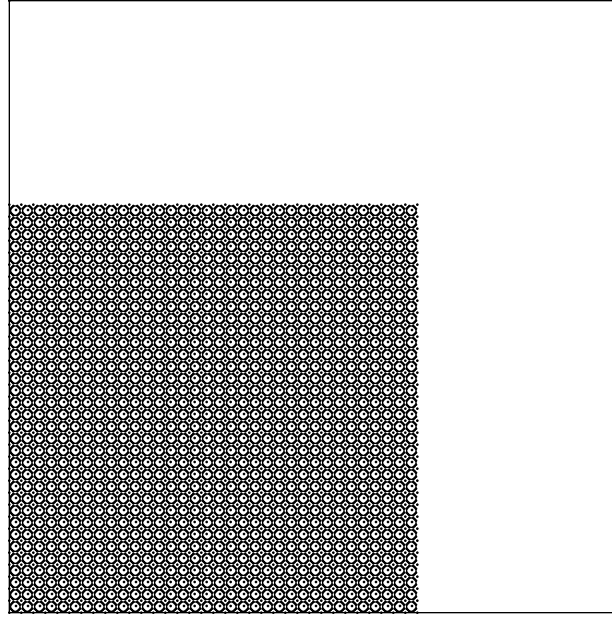


Figure 5.3: The third .poly file used in order to test effectiveness and convergence of the load balancing metric.

5.3 Metric Behavior and Convergence

For each test case, the 162 input parameters outlined in Table 5.1 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 5.4 shows the metric behavior for Fig. 5.1. The maximum metric value is 24.7650, and occurs when Fig. 5.1 is run with 8x8 subsets and a maximum triangle area of 1.6 cm². The minimum metric value is 1.0016 and occurs when Fig. 5.1 is run with 4x4 subsets and a maximum triangle area of 0.04 cm².

Metric Behavior with no Load Balancing Iterations

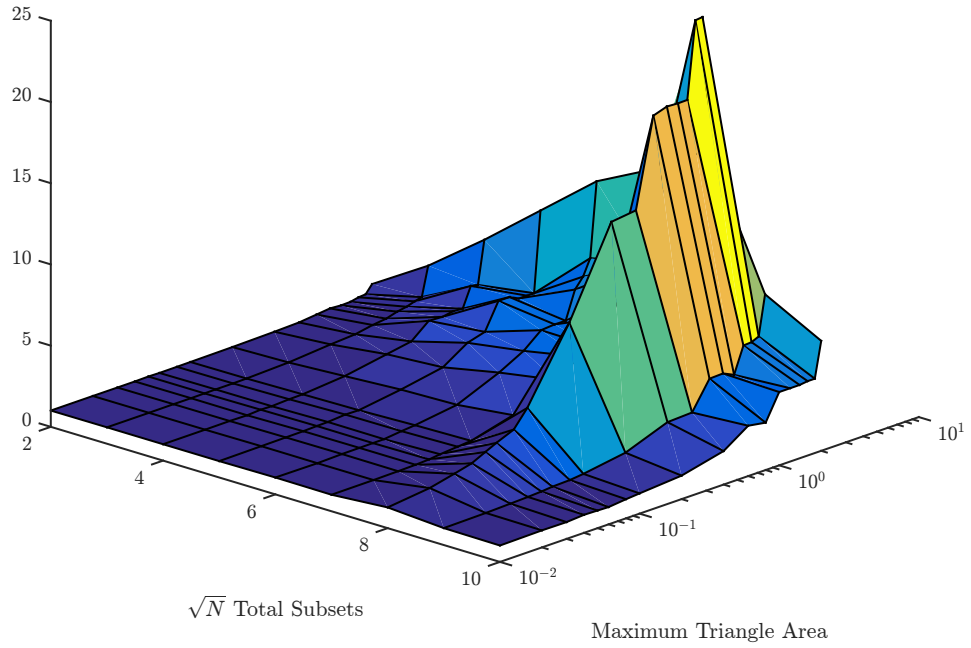


Figure 5.4: The metric behavior of the first test case run with no load balancing iterations.

Figure 5.5 shows the metric behavior for Fig. 5.1. The maximum metric value is 5.0538 and occurs when Fig. 5.1 is run with 10x10 subsets and a maximum triangle area of 1.2 cm². The minimum metric value is 1.0017 and occurs when Fig. 5.1 is run with 4x4 subsets and a maximum triangle area of 0.04 cm².

Metric Behavior with 10 Load Balancing Iterations

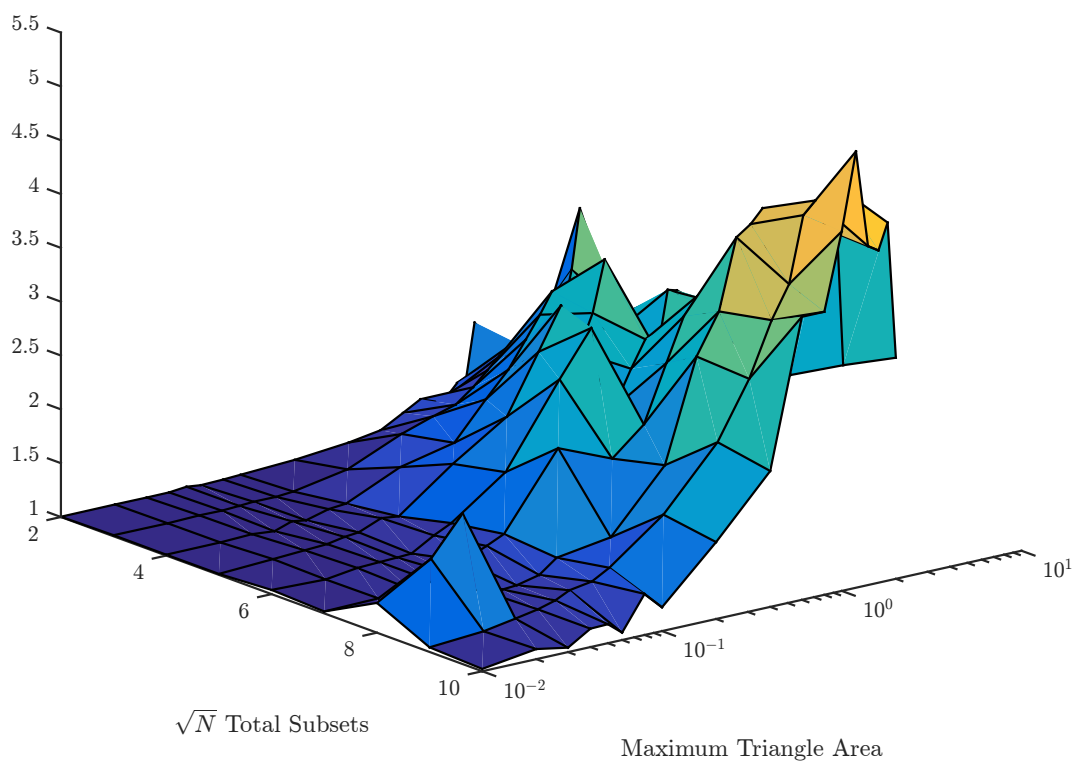


Figure 5.5: The metric behavior of the first test case run with 10 load balancing iterations.

Figure 5.6 shows the difference in metric behavior for Fig. 5.1. 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. 5.1 is run with 8x8 subsets with a maximum triangle area of 1.6 cm². The minimum improvement has a value of very close to 1.0 and occurs for many of the inputs.

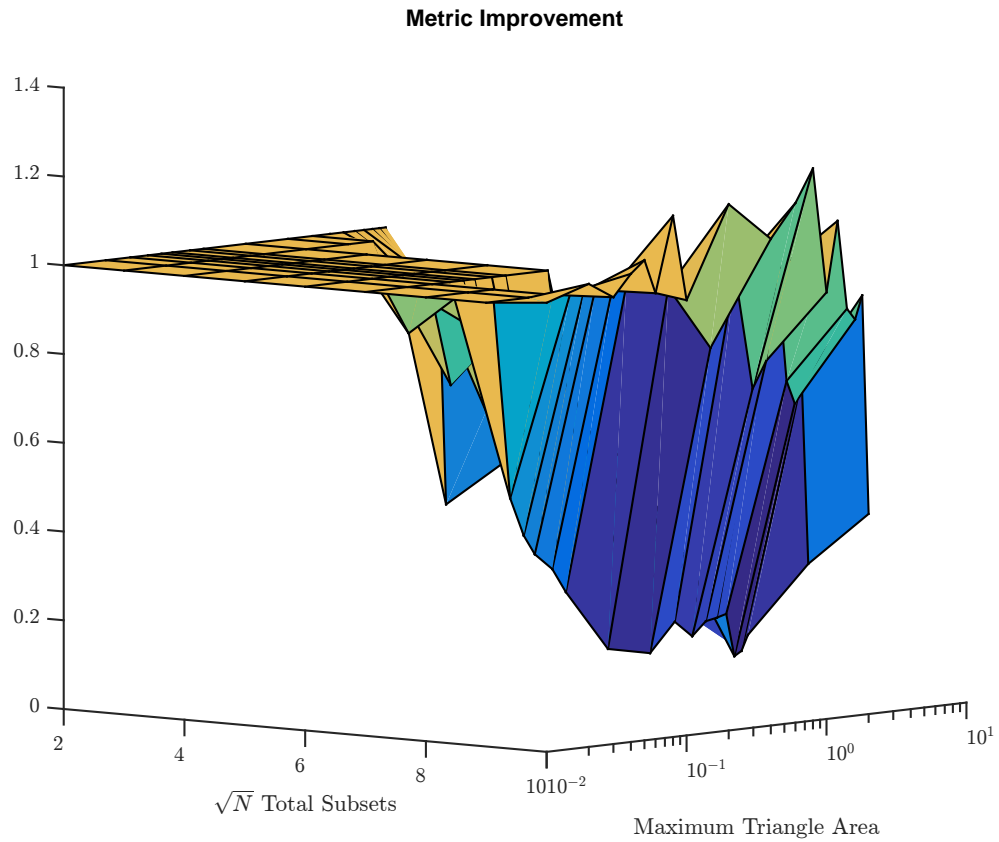


Figure 5.6: The difference in metric behavior between no iteration and 10 iterations. The closer the z-value to zero, the better the improvement.

Figure 5.7 shows the metric behavior for Fig. 5.2. The maximum metric is 22.6654 and occurs when Fig. 5.2 is run with 8x8 subsets with a maximum triangle area of 1.8 cm². The minimum metric is 1.0024 and occurs when Fig. 5.2 is run with 2x2 subsets with a maximum triangle area of 0.01 cm².

Metric Behavior with no Load Balancing Iterations

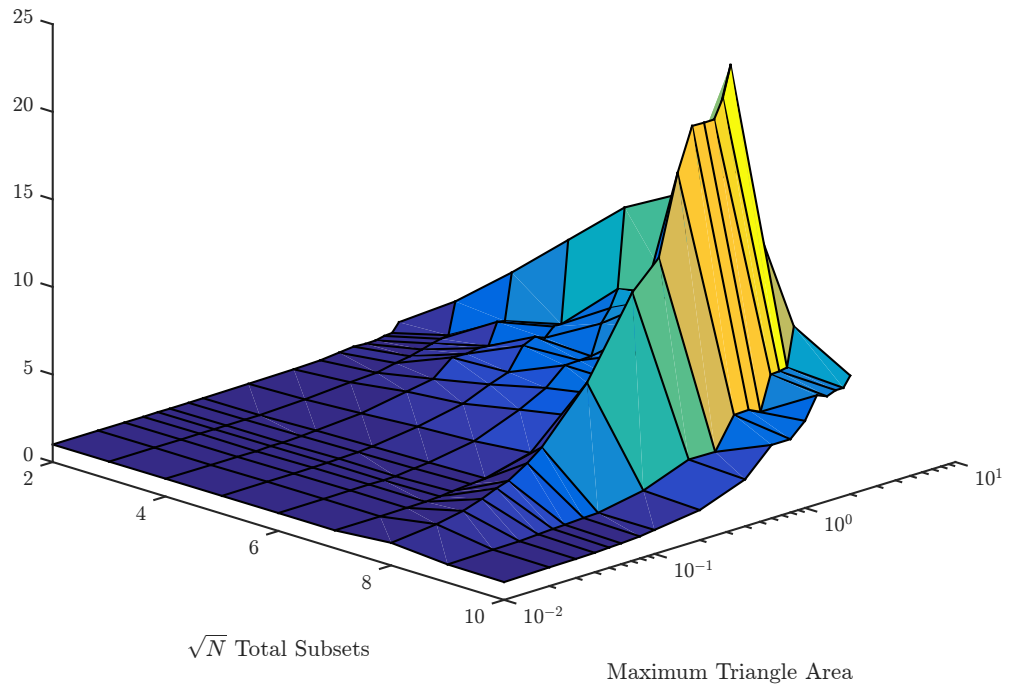


Figure 5.7: The metric behavior of the second test case run with no load balancing iterations.

Figure 5.8 shows the metric behavior for Fig. 5.2. The maximum metric is 3.9929 and occurs when Fig. 5.2 is run with 10x10 subsets with a maximum triangle area of 1.8 cm².

The minimum metric is 1.0024 and occurs when Fig. 5.2 is run with 2x2 subsets with a maximum triangle area of 0.01 cm^2 .

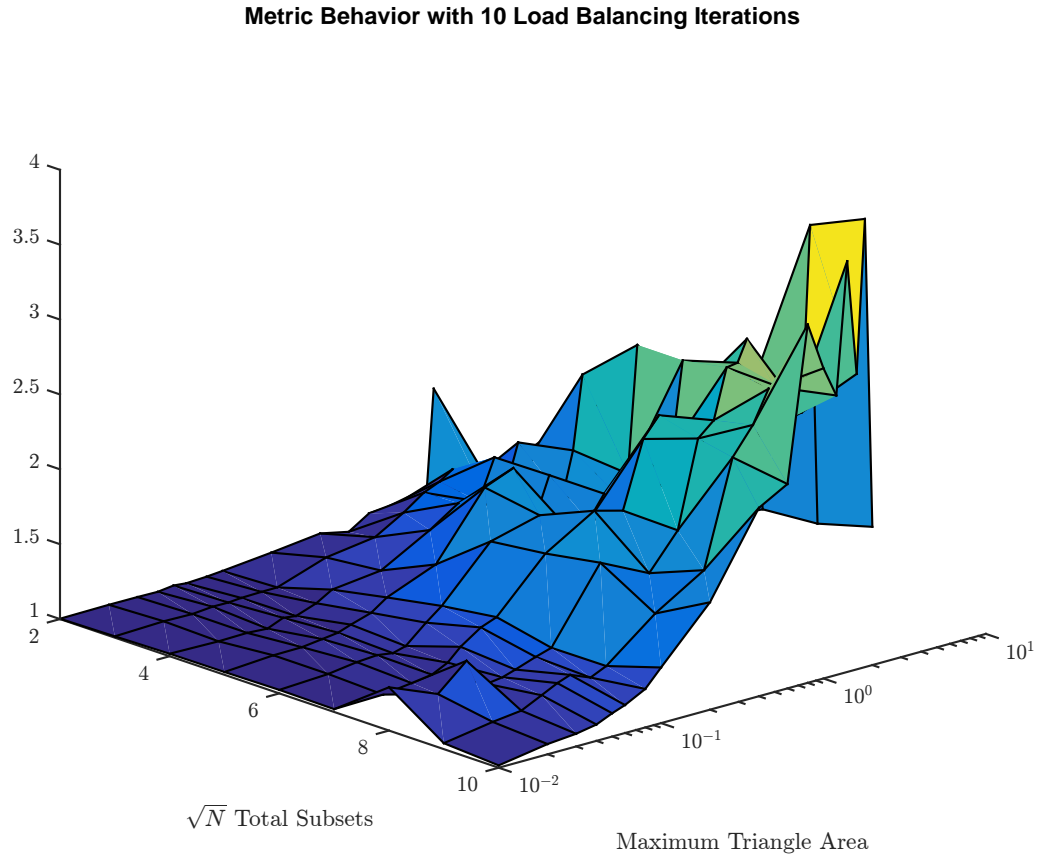


Figure 5.8: The metric behavior of the second test case run with 10 load balancing iterations.

Figure 5.9 shows the difference in metric behavior for Fig. 5.2. The maximum im-

provement has a value of 0.1090 and occurs for Fig. 5.2 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.

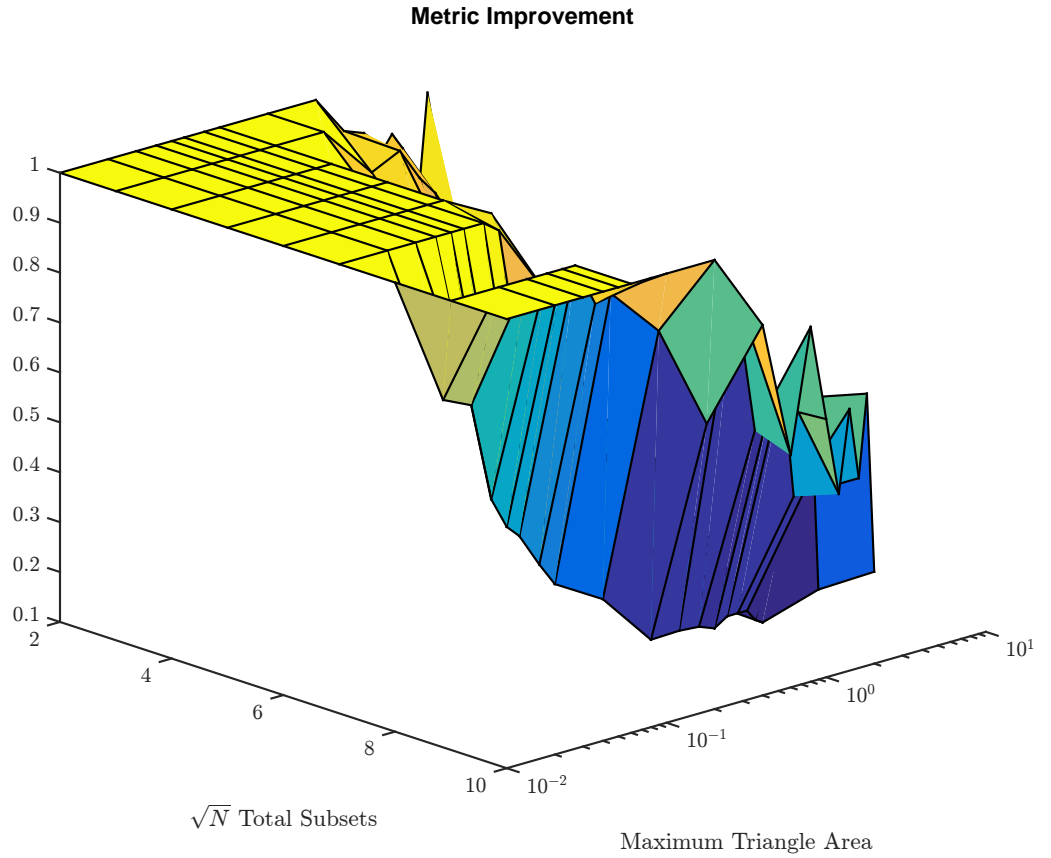


Figure 5.9: The difference in metric behavior of the second test case with no iteration and 10 iterations. The closer the z-value to zero, the better the improvement.

5.4 Solution Verification

The test problem ran for solution verification is a pure absorbing slab with $\Sigma_a = 0.3\text{cm}^{-1}$. It has reflecting boundary conditions on the top and the bottom, an incident isotropic flux on the left, and a vacuum boundary on the right. The reflecting boundaries are implemented to simulate an infinite slab in the y direction. The analytical solution to this problem is exponential **give the analytical expression and compare the ratio of analytical values (inlet/outlet) with the numerical ones**, and with the flux placed on a logarithmic scale, we see the flux linearly decrease in the x direction.

fix spacing in bibliography, if any...