CONSERVATIVE DATA TRANSFER BETWEEN NON-MATCHING PLANAR INTERFACES.

BY

RYAN TOMOKIYO

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Aerospace Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Advisor:

Professor Daniel J. Bodony
Abstract

This thesis presents the theory and implementation of a parallel software package for transferring data between non-matching planar interfaces. The method used for transferring data is based on the common refinement technique developed by Jiao and Heath in 2004 [X. Jiao and M.T. Heath. Common-refinement-based data transfer between non-matching meshes in multiphysics simulations. *Int. J. Numer. Meth. Engng.*, 61:2402-2427, 2004]. A modification is made to the base method that provides strict conservation between participating meshes. It is shown that the modified common refinement based method provides accuracy that is equivalent to the standard method while strictly enforcing the form of conservation that is desired. A parallel implementation of the data transfer method is shown to have good scaling when the number of unknowns per processor exceeds 1000. Solving the linear system is the primary aspect of the data transfer process that determines the scalability of the method.
Acknowledgements

I would first like to thank my advisor Professor Bodony for all his support throughout my time at the University of Illinois. When I first began work on this project it was his advice and encouragement that helped me overcome the seemingly inevitable roadblocks that pop up when writing parallel code. His abilities as a researcher, teacher, and mentor can not be overstated and I thank him for helping me to grow both professionally and personally.

I would also like to thank my labmates for providing such a fun and friendly place to work. Being able to turn around or peek over the cubical walls at any time to have a conversation about anything helped make the days fly by.

Thank you to all my friends here who have become my family away from home. Mike and Bernadette, thank you for welcoming Amy and I into your lives and I wish you all the best as you move to your new home in North Carolina. Sam and Alex, the Kreig family gatherings have reshaped my idea of what it means to be a family, our holidays will not be the same without you.

Finally, I thank Amy for always being there for me whenever I needed. These past two years have been a roller-coaster, and I couldn’t have managed without you constantly pushing me to be a better person. I can’t wait to see where we’ll go from here.
# Table of Contents

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td></td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Problem Description and Nomenclature</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Organization</td>
<td>5</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>6</td>
</tr>
<tr>
<td>2.1 $L_2$ Minimization</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Constrained $L_2$ Minimization</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Numerical Algorithms</td>
<td>8</td>
</tr>
<tr>
<td>2.3.1 Constructing the Mass Matrix</td>
<td>9</td>
</tr>
<tr>
<td>2.3.2 Constructing the Load Vector</td>
<td>11</td>
</tr>
<tr>
<td>2.3.3 Solving the Linear System</td>
<td>14</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Design Philosophy</td>
<td>16</td>
</tr>
<tr>
<td>3.2 Required Data</td>
<td>16</td>
</tr>
<tr>
<td>3.3 External Packages</td>
<td>18</td>
</tr>
<tr>
<td>3.4 Package Implementation</td>
<td>19</td>
</tr>
<tr>
<td>3.4.1 Initialize the CDT interface manager</td>
<td>19</td>
</tr>
<tr>
<td>3.4.2 Initialize the interface sides</td>
<td>20</td>
</tr>
<tr>
<td>3.4.3 Separate interface processors</td>
<td>20</td>
</tr>
<tr>
<td>3.4.4 Add nodes and cells to each side of an interfaces</td>
<td>20</td>
</tr>
<tr>
<td>3.4.5 Create the data transfer matrices for each interface</td>
<td>21</td>
</tr>
<tr>
<td>3.4.6 Update, transfer, and extract data</td>
<td>22</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>23</td>
</tr>
<tr>
<td>4.1 Accuracy and Conservation</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Parallel Performance</td>
<td>26</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>35</td>
</tr>
<tr>
<td>5.1 Recommended Future Work</td>
<td>35</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>37</td>
</tr>
<tr>
<td>References</td>
<td></td>
</tr>
</tbody>
</table>
## List of Figures

1.1 An overlay of two non-matching meshes showing the common refinement subelements. .................................................. 2
1.2 Sample rotor-stator grids for a turbine. Image courtesy of Rolls-Royce, with permission. .................................................. 3

2.1 Isoparametric mapping for a quadrilateral. ................................................. 9
2.2 Visual of 7-point quadrature rule for triangles. .............................................. 11
2.3 Example of overlapping triangles producing regions with 3-6 sides. .............. 12

3.1 Example mesh for creating an adjacency matrix. ........................................... 17

4.1 MATLAB’s *peaks* function. ........................................................................... 24
4.2 Repeated transfer between a quadrilateral mesh (Mesh A) and a triangular mesh (Mesh B). .................................................. 25
4.3 Computational domain with x-z density contour of the initial condition. ....... 27
4.4 Visualization of an initial pulse propagating through a non-matching interface. 27
4.5 Setup time scaling. ......................................................................................... 29
4.6 Mass matrix setup time scaling. ........................................................................ 30
4.7 Mixed mass matrix setup time scaling. ............................................................ 31
4.8 Strong scaling of total iteration time. ............................................................... 32
4.9 Strong scaling of total interpolation time. ....................................................... 33
4.10 Total iteration time vs. the number of nodes per processor. ......................... 34

A.1 Repeated transfer between a quadrilateral mesh (Mesh A) and a quadrilateral mesh (Mesh B). .................................................. 39
A.2 Repeated transfer between a triangular mesh (Mesh A) and a triangular mesh (Mesh B). .................................................. 40
A.3 Repeated transfer between a triangular mesh (Mesh A) and a quadrilateral mesh (Mesh B). .................................................. 41
A typical approach to modern multiphysics simulations is to use a partitioned approach where regions involving different physical models are solved on separate grids. This type of approach is advantageous because it allows the user to leverage already existing, well tested solvers for each of the sub-domains. The main issue then becomes one of how to transfer data between the sub-domains. For node-based solvers, one solution is to have matching node locations so data can be transferred directly from one grid to the other. However, when matching nodes are used, one grid may become unnecessarily fine to match with a second grid in regions where high resolution is needed. For example, in fluid-structure interaction, a refined fluid mesh at the structure interface maybe needed to capture the boundary layer accurately, causing the structural mesh to be over-resolved.

If, instead, tailored grids are to be used for each sub-domain, then the interface between them will generally be non-matching. In these cases an interpolation scheme is required when transferring data from one grid to the other. For smooth solutions, polynomial or point-wise interpolation can be accurate, but there is no guarantee of conservation. In order to impose conservation the Galerkin method can be use to transform the interpolation into an set of integral equations [5],

\[ \int_{\Omega} wg \, dx = \int_{\Omega} wf \, dx, \]  

(1.1)

where \( f \) is a source function and \( g \) is the corresponding interpolated target function to be determined. If the weight functions \( w \) are chosen to be the basis functions of the target
mesh $\phi$, then this form is conservative due to the basis functions summing to one [6]. The challenge in this formulation then comes from the evaluation of the integral $\int_\Omega \phi f \, dx$ because the functions $\phi$ and $f$ are defined on the target and source meshes respectively. Both Jiao and Heath [5] and Jaiman et al. [6] assess several methods for computing this integral which include source-based, target-based, and common refinement-based discretization methods. It was shown that both source based and target based methods have a strong dependence on the amount of mis-match between grids and can also lead to oscillations in the interpolated solution [6]. Both studies show the common refinement discretization proves to be an improvement in both accuracy and conservation over the other methods. The improvement from the common refinement based methods comes from the fact that the integral $\int_\Omega \psi_i f \, dx$ can be computed exactly over the subelements that are defined by the intersection of the source and target basis functions, see Fig. 1.1. By integrating over the subelements, the common refinement based methods avoid violating the regularity condition necessary for numerical quadrature [5].

![Figure 1.1: An overlay of two non-matching meshes showing the common refinement subelements.](image-url)
1.1 Motivation

The motivation for this project stems from the need for strictly conservative data transfer between rotor and stator grids in a gas turbine engine simulation, as shown in Fig. 1.2. Typically, the background fluid grids for both rotor and stator are rigid and the rotor grid will translate relative to the stator grid. The interface between the rotor and stator grids, called a sliding plane, consists of the abutting planar surface meshes from both grids. In this

Figure 1.2: Sample rotor-stator grids for a turbine. Image courtesy of Rolls-Royce, with permission.
type of simulation node locations across the sliding plane will frequently be non-matching as the grids move with respect to one another.

1.2 Problem Description and Nomenclature

The data transfer method presented here was developed for simulations in which data values are stored at discrete locations called nodes which are the vertices for the cells or elements that form the 2D interface mesh. For most unstructured grids, the interface mesh will consist of either triangles or quadrilaterals for which linear basis functions can be used with the nodal values to produce a piecewise-linear polynomial over the mesh. The data transfer problem then reduces to determining the nodal values on a target mesh given the nodal values of a source mesh along with the basis functions for each mesh.

Let $f$ be a function defined on the source mesh with basis functions $\phi$ such that the piecewise-linear discretization of $f$ becomes $f = \sum_{i=1}^{N} f_i \phi_i$ where $N$ is the number of source nodes. Similarly, let $g = \sum_{i=1}^{M} g_i \psi_i$ be the discretization of the interpolated function on the target mesh with $M$ nodes and basis functions $\psi$. The goal of this thesis is to find target nodal values $g_i$ that minimizes the residual $r = f - g$ while being conservative in the sense that,

$$\int_{\Omega_{src}} f \, dx = \int_{\Omega_{tgt}} g \, dx,$$

where the first term represents the integral of the source solution computed over the source grid and the second term is the integral of the transfered data computed over the target grid.
1.3 Organization

The remainder of this thesis will continue as follows. Chapter 2 describes the derivation of a basic common-refinement based data transfer method which minimizes the $L_2$ norm of error. It is then shown how the base derivation can be modified to strictly enforce conservation between participating grids. Chapter 2 also describes the general steps for performing the conservative data transfer using the new method, along with more detailed algorithms which could then be implemented numerically. Chapter 3 then describes in detail the development of the parallel software package which implements the conservative data transfer method. Numerical results along with closing remarks are then presented in Chapters 4 and 5, respectively.
Chapter 2

Common Refinement-Based Data Transfer

In Section 1.2 the relevant quantities were defined along with one of the objectives of the data transfer methods which was to reduce the residual, \( r = f - g \). In this thesis the residual is reduced through minimization in the \( L_2 \) norm, where the \( L_2 \) norm is defined as \( \int_{\Omega} (f - g)^2 \, dx \). In [5] it was shown that other norms, such as a Sobolev norm, could be used in order to achieve smoothed solutions when the data is rapidly varying.

2.1 \( L_2 \) Minimization

As stated above, the \( L_2 \) norm of the error can be written as \( \int_{\Omega} (f - g)^2 \, dx \). Replacing \( g \) with the discretized form and expanding the integrand, the \( L_2 \) minimization proceeds by setting the derivative with respect to \( g_i \) equal to zero as follows:

\[
\frac{\partial}{\partial g_i} \int_{\Omega} (f - \sum_{j=1}^{M} g_j \psi_j)^2 \, dx = \frac{\partial}{\partial g_i} \int_{\Omega} f^2 \, dx - 2 \frac{\partial}{\partial g_i} \int_{\Omega} \sum_{j=1}^{M} g_j \psi_j f \, dx + \frac{\partial}{\partial g_i} \int_{\Omega} (\sum_{j=1}^{M} g_j \psi_j)^2 \, dx \tag{2.1}
\]

\[0 = 2 \sum_{j=1}^{M} \int_{\Omega} g_i \psi_i \psi_j \, dx - 2 \int_{\Omega} \psi_i f \, dx \tag{2.2}\]

\[\sum_{j=1}^{M} \int_{\Omega} \psi_i \psi_j \, dx \, g_i = \int_{\Omega} \psi_i f \, dx \tag{2.3}\]

The \( L_2 \) minimization reduces to a \( M \times M \) linear system \( Mx = b \), where \( M_{ij} \) is the consistent mass matrix defined by \( \int_{\Omega} \psi_i \psi_j \, dx \), \( b = \int_{\Omega} \psi_i f \, dx \) is the load vector, and \( x \) is the vector of unknowns \( g_i \). It can also be seen that system of equations that result from \( L_2 \) minimization are the same that arose from using the target basis functions for the weighting functions.
in the Galerkin formulation in Eq. 1.1. In [5], a detailed error analysis shows that the discretization error associated with the $L_2$ minimization is $O(h_s^2 + h_t^2)$ where $h_s$ and $h_t$ are the mesh resolutions of the source and target meshes, respectively. Jiao and Heath [5] also show that $L_2$ minimization provides strict conservation in the sense that $\int_\Omega (f - g) v d\mathbf{x} = 0$ because the error $(f - g)$ is orthogonal to any function $v$ spanned by the target basis functions which includes constant functions.

### 2.2 Constrained $L_2$ Minimization

In the previous section, conservation was defined such that $\int_{\Omega_{sub}} (f - g) v d\mathbf{x} = 0$ when the integration is performed over the subelements of the common-refinement, denoted by $\Omega_{sub}$. However, this definition of conservation may not be suitable for some simulations. It may be necessary for conservation to be defined such that $\int_{\Omega_s} f d\mathbf{x} = \int_{\Omega_t} g d\mathbf{x}$ where the integrals are performed using the source and target basis functions respectively. With this new definition of conservation, a Lagrange multiplier, $\lambda$, is introduced to enforce the constraint $\int_{\Omega_s} f d\mathbf{x} - \int_{\Omega_t} g d\mathbf{x} = 0$ along with $L_2$ norm of the error. The modified functional to be minimized is shown in Eq. (2.4),

$$
\mathcal{L} = \int_{\Omega_{sub}} (f - g)^2 d\mathbf{x} + \lambda \left( \int_{\Omega_s} f d\mathbf{x} - \int_{\Omega_t} g d\mathbf{x} \right). \tag{2.4}
$$

Again, replacing $g$ with the discretized form and setting the derivative of $\mathcal{L}$ with respect to $g_i$ and $\lambda$ equal to zero, the modified linear system becomes:

$$
\sum_{j=1}^{M} \int_\Omega \psi_i \psi_j d\mathbf{x} g_i - \frac{\lambda}{2} \int_\Omega \psi_i d\mathbf{x} = \int_\Omega \psi_i f d\mathbf{x} \tag{2.5}
$$

$$
\int_{\Omega_s} f d\mathbf{x} = \sum_{i=1}^{M} \int_{\Omega_t} g_i \psi_i d\mathbf{x} \tag{2.6}
$$
Here the modified linear system can be written as $\hat{M}\hat{x} = \hat{b}$ where the modified matrix and vectors are defined as,

$$
\hat{M} = \begin{bmatrix}
M & \int \psi_1 \\
\vdots & \vdots \\
\int \psi_M & 0 \\
\end{bmatrix}, \quad \hat{x} = \begin{bmatrix}
x \\
-\frac{\lambda}{2} \\
\int \int_{\Omega} f \, dx \\
\end{bmatrix}, \quad \hat{b} = \begin{bmatrix}
b \\
\end{bmatrix}. \quad (2.7)
$$

By introducing an additional degree of freedom we increase the problem size to that of a $(M + 1) \times (M + 1)$ system of equations. However, it can be seen that the last row of the modified linear system strictly enforces conservation between participating meshes.

### 2.3 Numerical Algorithms

In the previous section the constrained $L_2$ minimization data transfer method was shown to reduce to a system of linear equations, $\hat{M}\hat{x} = \hat{b}$. The main steps for implementing the data transfer method can be broken into three steps.

1. Construct the Mass Matrix, $\hat{M}$
2. Construct the Load Vector, $\hat{b}$
3. Solve the linear system, $\hat{M}\hat{x} = \hat{b}$

The following sections will describe a general method for each of these steps.
2.3.1 Constructing the Mass Matrix

As shown in Section 2.2 the modified mass matrix takes the form:

\[
\hat{M} = \begin{bmatrix}
M & \int \psi_1 \\
\vdots & \vdots \\
\int \psi_M & 0
\end{bmatrix},
\]

(2.8)

where \( M = \int_\Omega \psi_i \psi_j d\mathbf{x} \) is the consistent mass matrix. The consistent mass matrix can be constructed simply by looping over each cell within the target mesh and performing Gaussian quadrature of sufficient order to compute the integral of the product of two shape functions exactly. For arbitrary quadrilaterals an isoparametric mapping can be used to map node locations from physical global coordinates to local coordinates, see Fig. 2.1. In local coordinates \((\varepsilon, \eta)\) which range from \(-1\) to 1, the shape functions for a bilinear rectangle are:

\[
N_1 = \frac{1}{4} (1 - \varepsilon)(1 - \eta)
\]

(2.9)

\[
N_2 = \frac{1}{4} (1 + \varepsilon)(1 - \eta)
\]

(2.10)

\[
N_3 = \frac{1}{4} (1 + \varepsilon)(1 + \eta)
\]

(2.11)

\[
N_4 = \frac{1}{4} (1 - \varepsilon)(1 + \eta)
\]

(2.12)
The quadrature points and weights for the rectangle in local coordinates can then be determined by using a four point quadrature scheme in each dimension. The four point quadrature points are $\pm \sqrt{(3 - 2\sqrt{6}/5)/7}$ with weight $(18 + \sqrt{30})/36$ and $\pm \sqrt{(3 + 2\sqrt{6}/5)/7}$ with weight $(18 - \sqrt{30})/36$.

For arbitrary triangles, quadrature points are typically given in terms of triangle coordinates $(\zeta_1, \zeta_2, \zeta_3)$. The mapping from global to local triangle coordinates can be written as Eq. 2.13, where the indices i, j, and k are modulo 3 cyclic.

\begin{align*}
\zeta_i &= \frac{1}{2A} (a_i + b_i x + c_i y) \\
a_i &= x_j y_k - x_k y_j \\
b_i &= y_j - y_k \\
c_i &= x_k - x_j
\end{align*}

(2.13)

The triangle coordinates $(\zeta_1, \zeta_2, \zeta_3)$ also have the property that $\zeta_1 + \zeta_2 + \zeta_3 = 1$ so they can be written in terms of two independent variables, $(\varepsilon, \eta)$, where $\zeta_1 = \varepsilon$, $\zeta_2 = \eta$, and $\zeta_3 = 1 - \varepsilon - \eta$. For integrating over triangle shape functions a fifth order 7-point quadrature rule is used where the points and weights are given in Table 2.1 [1], with a visual of the quadrature points in Fig. 2.2.

**Table 2.1: Points and weights for 5th order 7-point quadrature rule.**

<table>
<thead>
<tr>
<th>Weight</th>
<th>$\zeta_1$</th>
<th>$\zeta_2$</th>
<th>$\zeta_3$</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0.225$</td>
<td>$1/3$</td>
<td>$1/3$</td>
<td>$1/3$</td>
<td>$1$</td>
</tr>
<tr>
<td>$0.125939180544827$</td>
<td>$0.797426985353087$</td>
<td>$0.101286503723456$</td>
<td>$0.101286503723456$</td>
<td>$3$</td>
</tr>
<tr>
<td>$0.132394152788506$</td>
<td>$0.470142064105115$</td>
<td>$0.470142064105115$</td>
<td>$0.059715871789770$</td>
<td>$3$</td>
</tr>
</tbody>
</table>
2.3.2 Constructing the Load Vector

The construction of the load vector is the most involved step for implementing a common-refinement based data transfer method. The complexity comes from the need to compute the overlay between source and target elements. As it is written, the integral $\int_{\Omega_c} f \psi_i \, dx$ depends on both the geometry and the actual function values of the source solution. However, if the source function $f$ is replaced with the discretized form, the geometry of the source mesh can be separated from the actual source values as,

$$b = \int_{\Omega_c} f \psi_i \, dx = \int_{\Omega_c} \psi_i \phi_j \, dx \; f_j = Nf.$$  \hspace{1cm} (2.14)

The integral in Eq. 2.14 can then be written as a $M \times N$ matrix, $N$, called the mixed mass matrix. For the constrained data transfer problem, the modified mixed mass matrix becomes;

$$\hat{N} = \begin{bmatrix} N \\ \int \phi_1 \cdots \int \phi_M \end{bmatrix}.$$  \hspace{1cm} (2.15)
If the mixed mass matrix is constructed then the load vector can be easily created through a matrix-vector multiplication with a $N \times 1$ vector of source values. This is useful because by constructing the mixed mass matrix first, the mesh overlay problem need only be performed once when transferring multiple variables across an interface. Additionally, it can be seen that the mixed mass matrix for transferring data from say mesh A to mesh B, is the transpose of the mixed mass matrix for transferring data from mesh B to mesh A. This means that in a two-way coupled simulation, both mixed mass matrices can be constructed at the same time, again reducing the number of overlays that need to be performed.

In order to construct the mixed mass matrix each element on the target side must know the elements on the source side with which it shares common area. This requirement is often a major issue for implementing geometry-based interpolation methods in parallel. For now, assume that a small subset of source elements have been identified as potentially overlapping for each target element. In order to determine if a source and target element overlap the Computational Geometry Algorithms Library (CGAL) is used to accurately compute the overlay between two elements [8]. The face-extended overlay algorithm within CGAL takes in the vertices of two arrangements (elements) as inputs and returns the vertices corresponding to the overlapping region [8]. In general it will not be known how many sides the resulting overlapping region will have. For example, the overlapping area between two triangles can have between 3 and 6 sides, see Fig. 2.3. In order to perform the required integration,

![Figure 2.3: Example of overlapping triangles producing regions with 3-6 sides.](image)

quadrature points for these arbitrary regions need to be defined. For this the overlapping regions are first triangulated using a Constrained Delaunay Triangulation algorithm also from CGAL [9]. The integral over the entire overlapping region is then simply the sum of
the integrals over each of the triangles in the triangulation. To interpolate the source and target shape functions, an inverse mapping is needed to determine the corresponding \((\varepsilon, \eta)\) coordinates of the quadrature points. For this, Newton’s Method can be used to solve the system,

\[
x - \sum_{i=1}^{n} x_i N_i(\varepsilon, \eta) = 0
\]

\[
y - \sum_{i=1}^{n} y_i N_i(\varepsilon, \eta) = 0.
\]

Using Newton’s method, the \((\varepsilon, \eta)\) coordinates can be found by iterating

\[
\begin{bmatrix}
\varepsilon \\
\eta
\end{bmatrix}_{k+1} = \begin{bmatrix}
\varepsilon \\
\eta
\end{bmatrix}_k - J^{-1} \begin{bmatrix}
f_1 \\
f_2
\end{bmatrix}_k,
\]

until convergence. Here the Jacobian matrix, \(J\), is defined as:

\[
J = \begin{bmatrix}
\frac{\partial x}{\partial \varepsilon} & \frac{\partial y}{\partial \varepsilon} \\
\frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta}
\end{bmatrix}
\]

with inverse:

\[
J^{-1} = \frac{1}{|J|} \begin{bmatrix}
\frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \varepsilon} \\
-\frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \varepsilon}
\end{bmatrix}
\]

Combining all these parts, the construction of the mixed mass matrix can be written as Algorithm 1.
Algorithm 1: Constructing the Mixed Mass Matrix, $\hat{N}$.

for $i = 1 : \# \text{ of target cells}$ do
  for $j = 1 : \# \text{ of potentially overlapping Source Cells}$ do
    - Use CGAL’s face extended overlay routine to determine if the two cells intersect.
    if the source and target cells intersect then
      - Use CGAL’s Constrained Delaunay Triangulation to triangulate the overlapping region.
      for each triangle in the triangulation do
        - Compute the Gaussian quadrature points and weights
      end for
      - Return all quadrature points and weights for the overlapping region.
      - Map quadrature points to local coordinates $(\varepsilon, \eta)$ for both source and target elements.
      - Use quadrature points and weights to compute $\hat{N}_{ij} = \int \psi_i \phi_j \, dx$.
    end if
  end for
end for
- Loop over source cells and compute the constraining row of the mixed mass matrix, $\hat{N}_{M+1,j} = \int \phi_j \, dx$.

2.3.3 Solving the Linear System

With methods for constructing the modified mass matrix and load vector the final step in the data transfer method is to solve the resulting linear system. The unmodified mass matrix is symmetric positive definite [5], however because an additional row and column have been added, the modified system is now indefinite so iterative methods like the conjugate-gradient method which require positive definiteness can not be used. There are still many other iterative and direct methods that remain which can be used to solve the linear system. For three dimensional simulations the interface mesh between regions may be small compared to the surrounding volume meshes. In these cases it may be more efficient to use a direct factorization of the mass matrix, i.e. LU or Cholesky factorization, especially in simulations when the sides of the interface are non-deforming. If, however, the cost of a direct factorization is prohibitive then an iterative method such as the Generalized Minimal Residual method (GMRES) or the Minimal Residual method (MINRES) could be used. Both GM-
RES and MINRES are Krylov subspace based iterative methods that reduce the 2-norm of the residual each iteration. The MINRES method takes advantage of the symmetry of the system by using a three-term recurrence relation for constructing an orthogonal basis for the Krylov subspace [3]. In Chapter 4 the scalability of these iterative methods are tested along with two direct factorization methods, LU and Cholesky.
Chapter 3
Conservative Data Transfer Package

This chapter presents the parallel software package developed to implement the data transfer method presented in this thesis. Throughout the remainder of this thesis the library will be referred to as the Conservative Data Transfer (CDT) package.

3.1 Design Philosophy

The CDT package was developed to work with solver codes that use various discretization methods, i.e. finite difference, finite element, or finite volume methods. In order to handle these different methods the CDT package contains a separate internal data structure which treats the input meshes as unstructured and stores all the information required to perform the data transfer. The main task for the application (main solver code) is to supply the CDT package with the information it requires. Once the required data are supplied and the data transfer is performed, the application can then extract the transferred results.

3.2 Required Data

The majority of data required by the CDT package is based on the meshes of the two sides of the interface. This is due to the fact that the common refinement data transfer method is geometrically based. For each side of an interface the following information is required by the CDT package.

- A MPI communicator containing at least all interface processors.
• The number of interface nodes and cells, both locally and globally.
• The coordinate values of each node.
• The nodes that form each cell in the mesh.
• An adjacency matrix.

Many of these required data should already be known or should be easy to obtain at run time. The only item that may present a challenge is the adjacency matrix. The adjacency matrix is a sparse matrix that describes the cell-cell connectivity of the mesh. This information is used by the external package ParMETIS to create an internal partitioning of the interface meshes. For the mesh given in Fig. 3.1, the corresponding adjacency matrix is given in Eq. 3.1 [2].

\[
\text{AdjMat} = \begin{bmatrix}
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
\end{bmatrix}
\] (3.1)

By having a separate internal partitioning over which the data transfer is performed, the CDT library is completely agnostic to original partitioning of either side of the interface. In 3D simulations, large volume meshes are partitioned such that the interior domains are well
balanced without regard to how many processors lie on the boundaries. The CDT package is designed to incorporate all the processors that lie on a particular interface. For example, if one side of an interface is partitioned between 3 processors while the other side is partitioned between only 2, then all 5 interface processors will participate in the data transfer process.

### 3.3 External Packages

The CDT package relies on several external packages to perform the data transfer. First, it is assumed that the main application code uses an MPI based implementation for parallel processing. One required piece of information stated in Section 3.2 was a MPI communicator that contains at least all of the processors that lie on the interface. Once the communicator is passed into the CDT package all communication and data distribution is handled internally using MPI.

As mention in Section 2.3.2, CGAL is used for accurately computing the overlay of two elements and the triangulation of the resulting region. The two sections of CGAL that are used within the CDT package are, the 2D Arrangement routines for the overlay function, and the 2D Triangulation routines for the Delaunay triangulations. CGAL also offers similar functionality for 3D arrangements and triangulations.

The final step in the data transfer process requires a linear system to be solved, for this the Portable Extensible Toolkit for Scientific Computations (PETSc) is used. PETSc offers a collection of data structures and routines for solving linear systems in parallel that are efficient and scalable [2]. PETSc also provides an interface to the ParMETIS parallel partitioning package that is used internally to partition the interface mesh. In order to access the parallel direct solvers, PETSc must be configured and compiled with either the MUMPS or SUPERLU_dist external packages.
3.4 Package Implementation

This section will describe how the CDT package can be implemented within an existing solver code. The CDT package contains several “high level” function calls that can be used to simplify the requirements on the application programmer. However, if greater control is required, the low level functions can also be accessed. The CDT package is written using the Fortran90 language. The following subroutines and data structures are accessed by “using” the modules from the CDT package.

3.4.1 Initialize the CDT interface manager

call Initialize_Interface_Manager(iMan, nInterfaces, myComm)

The CDT interface manager (iMan) is the data structure that contains all of the information about each of the interfaces. The required inputs to initialize the interface manager are the interface manager itself (defined in the CDT data structure module), the total number of interfaces that will be using the CDT package, and a MPI communicator. After the interface manager is initialized, several logical flags should be set for each interface that are used in later subroutine calls. These logical flags are used to specify whether a particular interface is static or non-deforming:

- is_static_interface
- non_deforming_mesh

A static interface is one where both sides of the interface remain fixed in space. A non-deforming interface is where the nodes on each side of the interface remain fixed relative to each other.
3.4.2 Initialize the interface sides

call Initialize_INTERFACE_SIDE(iMan, Iid, Sid, n_iNodes, n_iCells, 
n_pNodes, n_pCells)

Each interface within the interface manager contains two sides, 0 and 1. The user is responsible for determining and assigning the interface to which a side belongs. For example, specific boundary conditions or grid identifiers within the main solver code can be used to uniquely specify the sides within each interface. In order to accurately allocate memory, each processor must know the number of nodes and cells that are locally owned (n_pNodes, n_pCells). Additionally, the user must input the total number of nodes and cells that lie on each side of the interface (n_iNodes, n_iCells).

3.4.3 Separate interface processors

call gather_INTERFACE_PROCESSES(iMan, Iid)

When a processor initializes a side of an interface a flag is set so the processors that contain interface data can be separated from the rest of the processors in the simulation. When this subroutine call is made, separate communicators are created for each interface and for each side within each interface.

3.4.4 Add nodes and cells to each side of an interfaces

call Add_INTERFACE_NODE(iMan, Iid, Sid, ind, id, x, y, z)
call Add_INTERFACE_CELL(iMan, Iid, Sid, ind, id, 
        nodes_per_cell, node_inds)

Each processor containing interface data will add the nodes and cells that it owns. The CDT package takes in cell and node data as if each processors owns a unique set of cells.
Additionally, each processor should own all the nodes that belong to any cell it adds to the interface.

When adding node location data to the interface, the user must specify to which interface and side the node belongs, the local index of the node, the global side ID of the node, and the coordinates for the node. The global side ID should range from 1 to the total number of interface nodes on each side of an interface. The CDT package stores $x$, $y$, and $z$ coordinates for each node, however, only planar interfaces are supported so only the $x$ and $y$ coordinates are used in the data transfer routines. For 3D simulations planar interfaces should be mapped to a local $(x, y)$-plane.

When adding cell information the user must specify the interface and side to which the cell belongs, the local index of the cell, the global side ID of the cell, the number of nodes that form the cell, and an array of the node indexes that form the cell. The number of nodes that form each cell must be specified to allow for both triangles and quadrilaterals to be contained within a side.

3.4.5 Create the data transfer matrices for each interface

call Create_Interface_Matrices(iMan, Iid)

When a call is make to create_interface_matrices, the CDT package will first check to see if the interface (Iid) has been specified as either static or non-deforming. If the interface is static, meaning both sides remain fixed in space and time, then the interface matrices are only created once. If the interface is non-deforming, where the relative node coordinates within a side are fixed, the mass matrix is only built once for each side while the mixed mass matrices must be reconstructed whenever the two sides move with respect to each other. Finally, if the interface is neither static nor non-deforming, then both the mass matrix and mixed mass matrix must be reconstructed when the grids move.

If more control is desired for when the interface matrices are created, the user can use
the following lower level calls.

```plaintext
call Create_Mass_Matrix_Parallel(iMan, Iid)
call Create_Mixed_Matrix_Parallel(iMan, Iid)
```

### 3.4.6 Update, transfer, and extract data

```plaintext
call Update_Interface_Data(iMan, Iid, Sid, ind, val)
call Transfer_Interface_Data(iMan, Iid, src_Sid, tgt_Sid)
call Extract_Interface_Data(iMan, Iid, Sid, ind, val)
```

Once the interface matrices have been created for each interface, the nodal values can then be passed to the CDT package through a call to *update_interface_data*. When all the nodal values have been updated, the data can then be passed from a source side (`src_Sid`) to a target side (`tgt_Sid`). For two-way coupling the *transfer_interface_data* routine should be called twice with the side ID arguments switched so that data from both sides of the interface are transferred. When the transfer of data is complete, the *extract_interface_data* routine can then be used to extract the transferred values from the CDT package.
Chapter 4

Numerical Results

In this chapter the performance of the data transfer method will be discussed. The first set of test cases will show how the accuracy and conservation measures of the new method compare to that of the standard common refinement based method developed by Jiao and Heath [5]. The second test case is used to assess the parallel performance of the CDT package.

4.1 Accuracy and Conservation

In order to verify the accuracy and conservation of the new data transfer method, a 2D test case similar to that presented in [5] was used. In this test an analytic function is passed repeatedly between two 2D meshes. The analytic function chosen was the *peaks* function from MATLAB defined on a \([-3, 3] \times [-3, 3]\) square domain, see Fig. 4.1.

During this test, the *peaks* function is originally defined on one mesh, mesh A, and is passed from mesh A to a secondary mesh B. The two meshes are chosen so that the node locations of mesh A and mesh B are non-matching. Additionally, in two of the test cases the meshes are composed of different elements. For example, mesh A will consist of quadrilateral elements while mesh B consists of triangular elements.

One iteration for this test will involve the analytic function being passed from mesh A to mesh B and back to mesh A. After each iteration the transferred data on mesh A is compared to the original data that was defined on mesh A. The relative error is then computed using Eq. 4.1.

\[
E_{rel} = \frac{||f_i - f_0||_2}{||f_0||_2}
\]  

(4.1)
Conservation error for the method will be measure by comparing the original integral of the function defined on mesh A to the integral of the transfered function on each mesh after each iteration. During each iteration Eq. 4.2 and 4.3 are used to compute the absolute difference between the original integral and the transfered integral.

\[
E^c_A = \left| \int_{\Omega_A} f_0 \, dx - \int_{\Omega_A} f_i \, dx \right| \quad (4.2)
\]

\[
E^c_B = \left| \int_{\Omega_A} f_0 \, dx - \int_{\Omega_B} g_i \, dx \right| \quad (4.3)
\]

Here the modification that was made to the data transfer method should strictly enforce equality between the integrals in Eq. 4.2 and 4.3.

Figure 4.2 shows the accuracy and conservation results for the repeated transfer between a quadrilateral mesh (Mesh A) and a triangular mesh (Mesh B). Mesh A consists of 45 nodes in each the x and y directions while mesh B consists of 32 nodes in each direction. Figure
Figure 4.2: Repeated transfer between a quadrilateral mesh (Mesh A) and a triangular mesh (Mesh B).
4.2a shows that the relative error for the constrained $L_2$ minimizing data transfer method is unchanged from the original $L_2$ minimizing method. Additionally, it can be seen from Figs. 4.2b and 4.2c that the modified data transfer method provides strict conservation when integration is performed over the individual meshes. The absolute difference between the integrals is based on the error tolerances used within the linear solver. Appendix A shows the performance figures for the other three accuracy and conservation test cases where the elements of the two meshes varied between quadrilaterals and triangles.

4.2 Parallel Performance

The parallel performance of the CDT package was tested by integrating the package into an existing parallel multi-physics solver, MPSolver. MPSolver is a code developed by Professor Bodony at the University of Illinois at Urbana-Champaign for solving compressible fluid flow problems for research in aeroacoustics and turbulence.

The test case chosen was a 3D simulation of the propagation of an acoustic pulse. The computational domain along with a density contour of the initial condition is shown in Fig. 4.3. The computational domain is composed of two grids (cubes) with an interface formed by the abutting faces located at $x = 1$. The first grid ranging from $([0,1],[0,1],[0,1])$ consists of 1 million nodes with 100 nodes in each the $x$, $y$, and $z$ directions. The second grid ranging from $([1,2],[0,1],[0,1])$ consists of roughly 2 million nodes with 128 nodes in each direction. Figure 4.4 shows a $x-z$ slice of the initial conditions along with the solution after 100 iterations. The non-matching interface is located at $x = 1$ and it can be seen that the solution propagates through the interface with no visible distortion.

To assess the parallel performance of the data transfer method the acoustic pulse test case was run for 20 iterations with the number of processors ranging from 2 to 108. For each processor count the total number of processors was divided evenly between the two grids. Each grid was then partitioned in the $y$ and $z$ directions so that all processors within
Figure 4.3: Computational domain with x-z density contour of the initial condition.

(a) Initial condition  
(b) Solution after 100 iterations

Figure 4.4: Visualization of an initial pulse propagating through a non-matching interface.
a grid own interface data. In addition to varying the number of processors used in the simulation, multiple linear system solvers within PETSc were tested to determine the effect on parallel performance. Figures 4.5 - 4.7 show the strong scaling of the total setup time, mass matrix setup time, and mixed mass matrix setup time respectively. In the simulations where an iterative method was used to solve the linear system, the preconditioner is also shown in the legend of each figure. For example, MinRes - ILU(1) denotes that the minimal residual iterative method was used with a 1 level incomplete LU preconditioner. The time required to setup the linear system and preconditioner are recorded in the mass matrix setup time. For this simulation the setup of the direct factorization methods from the MUMPS or SuperLU packages are faster than setting up the incomplete LU preconditioner from the Hypre package. The mixed mass matrix setup consists of the time to perform the internal ParMETIS partitioning and data distribution along with the time to compute the element overlays. When fewer than 24 processors are used, the total setup time is driven by the element overlays during the construction of the mixed mass matrix.

The strong scaling of the total iteration time and the total interpolation time are shown in Fig. 4.8 and 4.9 respectively. In Fig. 4.8 the dashed line represents the strong scaling of the base fluid solver without the use of an interpolation scheme. Of the six linear system solvers tested, the LU and Cholesky factorization methods from the MUMPS package provide faster solve times than any of the iterative methods. For scalability PETSc recommends at least 10,000 unknowns per processor when using the linear system solvers. Figure 4.10 shows the total time to complete 20 iterations verses the number of nodes per processor. The largest linear system being solved in these simulations contains roughly 16,000 unknowns which is why the interpolation times, which consists primarily of solving the linear system, show scaling loss when greater than 24 processors are used. When the number of processors increases, the interpolation times begin to rise because the total communication time required increases while the amount of work on each processor decreases.
Figure 4.5: Setup time scaling.
Figure 4.6: Mass matrix setup time scaling.
Figure 4.7: Mixed mass matrix setup time scaling.
Figure 4.8: Strong scaling of total iteration time.
Figure 4.9: Strong scaling of total interpolation time.
Figure 4.10: Total iteration time vs. the number of nodes per processor.
Chapter 5
Conclusion

This thesis presents a method for the transfer of data between non-matching planar interfaces. With a slight modification to the data transfer method developed by Jiao and Heath [5], this new method provides strict conservation between participating meshes while maintaining an order of accuracy equivalent to the base method. A parallel software package is also introduced which implements the new data transfer method. The package was integrated into an existing fluid solver where the parallel performance was then assessed through a simulation of an acoustic pulse propagating in air. It was shown that the parallel performance of the data transfer method depends primarily on the linear system solver. For problems where the interface consists of a small subset of the entire domain, direct methods such as LU and Cholesky factorization provide the fastest solve times. If direct factorization methods become prohibitive, the method used to solve the linear system can be easily changed through the use of PETSc.

5.1 Recommended Future Work

The focus of this thesis was on transferring data between planar interface. However, in many multi-physics simulations the surface meshes between computational domains will be curved or be allowed to deform as in a fluid-structure interaction problem. In these types of simulations the data transfer problem becomes more difficult because non-matching curved meshes will have differing geometric realizations [5, 7]. Both [5] and [7] show that a reference mesh defined by the convex combination of the two participating meshes can be used in order
to provide both conservation and accuracy. An algorithm for constructing such meshes is presented in [4], however, the CGAL package may provide functionality that can simplify the process.
Chapter 6

References


Appendix A

Additional Figures

This appendix contains additional figures corresponding to accuracy and conservation test cases similar to that presented in Section 4.1. These test cases show the results for the same test, performed on various meshes. In all cases, mesh A consists of 32 equally spaced grid points in both the $x$ and $y$ directions, while mesh B consists of 45 equally spaced grid points in each direction. For meshes consisting of triangular elements, the MATLAB `delaunay()` function was used to create the triangular mesh based on the grid points. All figures show results consistent with those presented in Section 4.1.
Figure A.1 shows the accuracy and conservation measures for passing data between a quadrilateral mesh A, consisting of 32 nodes in each direction, and a quadrilateral mesh B, consisting of 45 nodes in each direction.

![Figure A.1](image_url)

(a) Relative Error

(b) Mesh A Conservation

(c) Mesh B Conservation

Figure A.1: Repeated transfer between a quadrilateral mesh (Mesh A) and a quadrilateral mesh (Mesh B).
Figure A.2 shows the accuracy and conservation measures for passing data between a triangular mesh A, consisting of 32 nodes in each direction, and a triangular mesh B, consisting of 45 nodes in each direction.

Figure A.2: Repeated transfer between a triangular mesh (Mesh A) and a triangular mesh (Mesh B).
Figure A.3 shows the accuracy and conservation measures for passing data between a triangular mesh A, consisting of 32 nodes in each direction, and a quadrilateral mesh B, consisting of 45 nodes in each direction.

Figure A.3: Repeated transfer between a triangular mesh (Mesh A) and a quadrilateral mesh (Mesh B).