Preforming Gather/Scatter Operations On A 2D Grid Using Tensor Contractions and CTF

Peter Tatkowski

Laboratory for Parallel Numerical Algorithms
Department of Computer Science
University of Illinois at Urbana-Champaign

July 27, 2017

Outline

Finite Element Methods

- Numerical method for solving problems of engineering and compuational physics
- Involves taking a mesh and splitting it into many elements
  - Makes what used to be solving complicated differential equations rather easy to approximate
- This has many advantages
  - Accurate representation of complex geometry
  - Easy representation of the total solution
  - Capture of local effects
- Spectral Element Methods are similar to Finite Element Methods, except that they use different nodes and different differential equations

Unstructured Grids

- I will focus on unstructured grid setup
  - This means that any individual element can be connected with any other element along its edges
  - Grid may be disconnected
- This kind of setup allows us to take into account lots of different grid possibilities

Figure 1

Figure 1: An example of a Finite Element Method

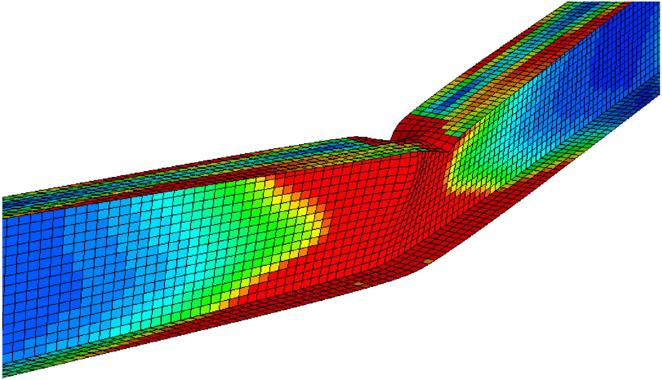
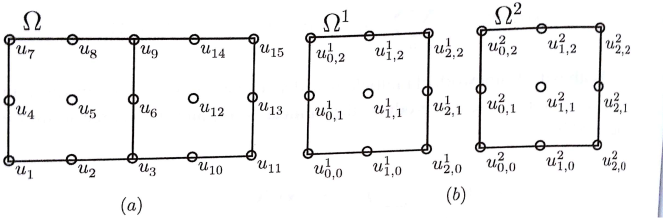


Figure 2

Figure 2: 2 elements with $n = 3$ degrees of freedom



Deville, M.O., Fischer, P.F., Mund, E.H. (2002) *High-Order Methods for Incompressible Fluid Flow.* New York, NY: Cambridge University Press.

Boundary Conditions

- When preforming a Finite Element Method in 2D on an unstructured grid, certian degrees of freedom repeat themselves
  - This replication happens because of a particular approach taken where it is highly convinient
- For each element:
  - The edges can repeat at most twice
  - The corners can repeat an infinite amount of times
- Is there a way to account for this repetition using tensor contractions?

Figure 3

Figure 3: Regular Grid

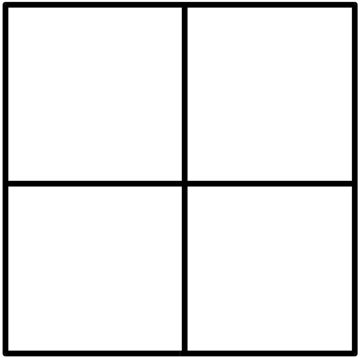
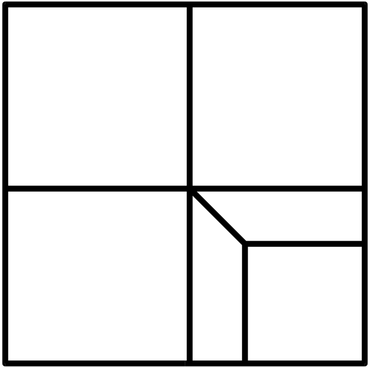


Figure 4

Figure 4: Irregular Grid



Problem Statement

- After computing calculations on individual degrees of freedom in each element, can we find an effecient way to sum up each degree of freedom's neighbors using tensor contractions?
- Also, how does this scale with multiple processes?
- More focused on long term preformance rather than short term setup
- Key idea: since internal degrees of freedom don't have any neighbors, we can effectively ignore them in our calculations.
  - We can reduce the tensor, as shown in the details later

Motivation

- Finite Element Analysis is used with many engineering applications
  - Computational fluid dynamics
  - Car crash simulation
  - Several chemical, fluid, structural applications as well
- Focus on implementing a simple, algebaic approach to the problem with tensor contractions. Advantageous because:
  - Simple to implement and understand
  - Can be computed with different meshes quickly
  - Easy to implement the methods using high-level linear algebra programming abstractions (like CTF)
  - Easy extension to higher dimensional domains

Tensor Implementation of Grid

- The grid u has dimensions $elems, n, n$
  - $elems$ is the number of finite elements in the grid
  - n is the number of degrees of freedom in each direction of the mesh
- Grid is this shape because it is unstrucuted
  - Each wall can connect to only one other wall
  - Each corner can have any number of neighbors
- Tensor can be filled with double values

Element Connectivity

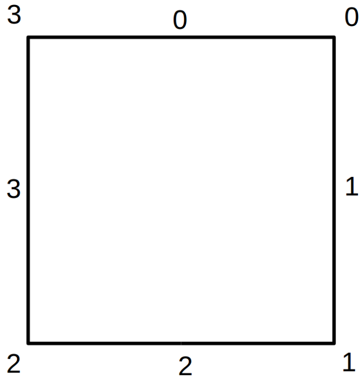
- We need some way to determine connectivity between elements
- First, number all elements in the mesh from $[0, elems - 1]$
- We allocate a global array of dimension $elems, 4$
    - This shape allows us to account for each element and each of its respective four edges
    - This array allows us to determine each element's neighbors
    - If an index contains a natural number, the current element's corresponding wall neighbors the other element referenced by the index
    - If an index contains a negative number, the current element's corresponding wall doesn't neighbor anything

Implicit Numbering

- The walls and corners in the tensor have an implicit numbering to them based on its coordinates in the *u* tensor
- This numbering will make it easier to not only compute wall-wall connectivity, but also corner connectivity as well
- It is important that this numbering is maintained within the grid, or else this method doesn't work
  - Several checks in place in the code to make sure that the mesh is correctly structured

Figure 5

Figure 5: How Walls Were Implicitly Defined



Wall Connectivity In The Mesh

- Each wall can only connect with a certain set of other walls, as indicated below:

0	1	2	3
1	0	0	1
2	3	3	2

- This is because the degrees of freedom will be backwards if this is not honored
- User just has to make sure that all elements have this connectivity between other elements, works for all 2D meshes

General Form For Contractions

- We want to achieve a result like this for both the corners and the walls:

$$u_{walls} = P_{walls}^T G_{walls} P_{walls} u_{orig}$$

$$u_{corners} = P_{corners}^T G_{corners} P_{corners} u_{orig}$$

$$u = u_{orig} + u_{walls} + u_{corners}$$

- Each P tensor takes respective elements out of u_{orig}
- Each G tensor adds up shared elements

Wall Gather Operation

- The form of the u_{walls} contraction is:

$$u_{ijk} = P_{cdzjk} G_{iwabcd} P_{abzxy} u_{wxy}$$

- These tensors have shapes:
  - P: $2, 2, n, n, n$
  - G: $elems, elems, 2, 2, 2, 2$
- G_{iwabcd} contains 1 iff wall (c, d) of element i borders wall (a, b) of element w
  - This is simple one to one mapping: $(0, 0) = 0$, $(0, 1) = 1$, $(1, 1) = 2$, $(1, 0) = 3$

Wall Gather Operation
Continued

$$u_{ijk} = P_{cdzjk} G_{iwabcd} P_{abzxy} u_{wxy}$$

- P_{abzxy} contains 1 iff the wall corresponding to a, b is properly indexed by either x or y, and the one that it is not indexed by has the same value as z. For example, on an $n = 3$ grid:
  - $P_{0,0,1,1,2}$ would be one because index $1, 2$ is on wall 0 $(0, 0)$, and indicies z, x match
  - $P_{1,0,2,0,2}$ would be one because index $0, 2$ is on wall 3 $(1, 0)$, and indicies z, y match
  - $P_{0,1,1,1,2}$ would be zero because index $1, 2$ is on wall 0 $(0, 0)$, but the index is supposed to be on wall 1 $(0, 1)$
- P_{abzxy} reduces the tensor to just be its walls, while P_{cdzjk} brings the tensor back to its original shape $(elems, n, n)$

Corner Gather Operation

- The Form of the $u_{corners}$ contraction is:

$$u_{ijk} = P^{(1)}_{ak} P^{(2)}_{aj} G_{iwab} P^{(2)}_{bq} P^{(1)}_{br} u_{wqr}$$

- The shapes of these tensors are:
  - $P^{(1)}, P^{(2)}$: $4, n$
  - G: $elems, elems, 4, 4$
- G_{iwab} contains 1 iff corner a of element i borders corner b of element w

Corner Gather Operation

Continued

$$u_{ijk} = P^{(1)}_{ak} P^{(2)}_{aj} G_{iwab} P^{(2)}_{bq} P^{(1)}_{br} u_{wqr}$$

- $P^{(1)}_{br}$ brings each element of u to its left and right walls, which are repeated twice
- $P^{(2)}_{bq}$ reduces this even further, reducing the left and right walls to their first and last elements
  - This reduces every element of u to its corners in a $2, 2$ grid
- After G_{iwab} preforms its operation on u, $P^{(1)}_{ak} P^{(2)}_{aj}$ brings u back to its original form, $elems, n, n$

Filling The Corner Gather Tensor

- So far, we have only wall to wall connectivity
- How do we achieve corner connectivity with just wall connectivity?
- If the grid is set up correctly, the corner connectivity can be discovered using an algorithm to traverse the walls in $\mathcal{O}(elems^2)$, $\Omega(elems)$ time
  - Not necessarily a tight upper bound, but tight lower bound, depends on grid sturucture

The Algorithm

Algorithm For Corner Traversal

Algorithm 1 Corner Neighbor Gather

```
procedure CORNER TRAVERSAL
    let cornerarr ← length 4
    let edges ← wall connectivity
loop i through all corners:
    let cornerarr[i] ← mutable vector
    let traversepos ← current corner
    let currelem ← original element
    while edges[currelem][traversepos] ≠ origelem do
        let newelem ← edges[currelem][traversepos]
        if newelem is negative then Break
        let lastelem ← edges[origelem][(i+1) mod 4]
        if Not first iteration and newelem ≠ lastelem then
            push newelem to cornerarr[i]
        currelem ← newelem
        traversepos ← traversepos + 3 mod 4
```

The Algorithm
Continued

- This algorithm, if done over all corners, produces a *elems*, 4, *vector* array, where the *vector* index can be of variable length
  - The variable length accounts for the variable amount of elements one corner can connect to
- This algorithm is then used to fill the $G_{corners}$ tensor, by filling the appropriate indicies with 1
- Future work will involve a tensor-based version of this algorithm, allowing for maximum parallelization

Correctness Check

- The gather operation was tested by comparing every index in our calculated tensor with what the values should have been
- The true values were computed by adding up adjacent walls, and the corners were evaluated using a similar method to filling the gather corner tensor.
- Meshes tested were Figure 4 and also a larger version of Figure 3, both of which passed

Scaling

- Currently small scale testing being preformed right now
  - Mainly done with changing the number of elements and degrees of freedom
  - Changing number of elements, *elems*, changes run time more drastically than changing number of degrees of freedom, *n*
- Will test scalability with multiple processes on Blue Waters in the forseeable future

Summary

- Two tensor contractions preform the gather operation
- Operation can be preformed efficently in parallel
- Designed for many contractions, so setup time can be overlooked

- Outlook
  - Tensor version of Algorithm 1
  - Test scaliability of this method on Blue Waters
  - Transition this method over to 3D