Progress Towards a More Efficient Initialization for Discontinuous Galerkin FE Codes Based on Interface Elements

Andrew Seagraves

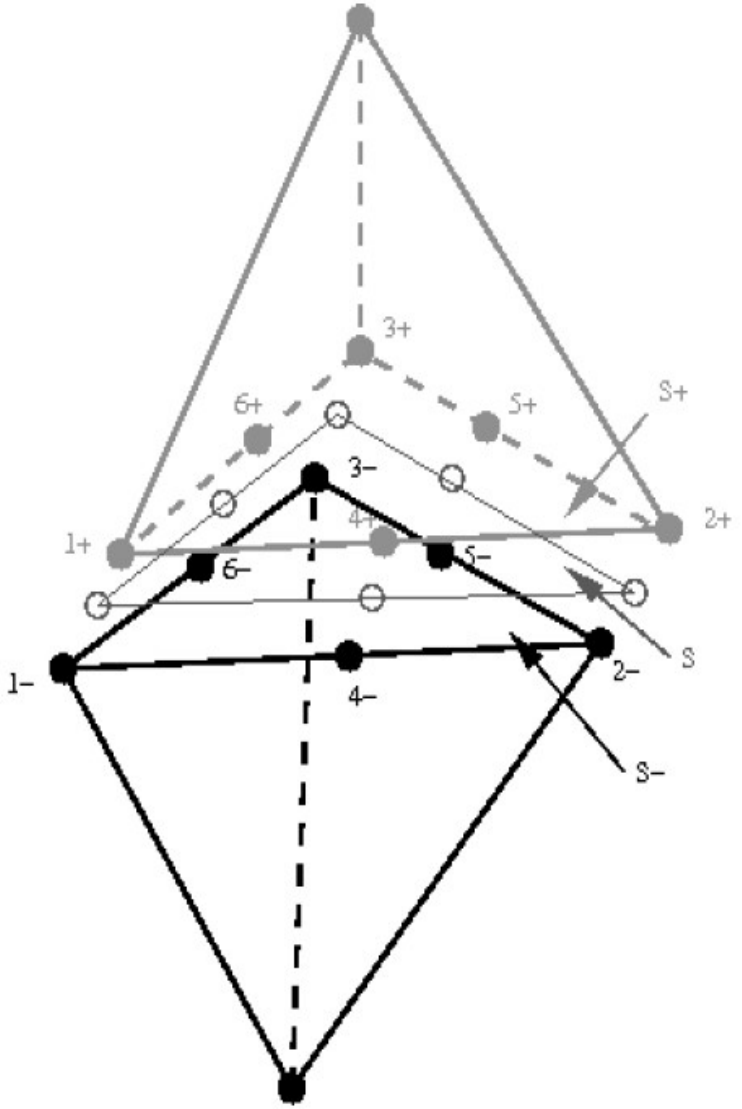
18.337 Final Project

My Parallel DG Code

- Computes the solution to non-linear elasticity problems allowing discontinuous displacement jumps at all the interelement boundaries
- This requires a completely discontinuous mesh where no elements are connected to any other elements
- It also requires the insertion of interface elements between the discontinuous volume elements

Interface Elements

- Interface elements are surface elements which live on either side of an element facet as shown ---------->



Background on the Current DG Code

- Interface elements must be inserted at all interior facets in the serial case with the boundaries ignored
- Interface elements must be inserted at all interior facets in each processor and also at the processor boundaries in the parallel case
- The solver requires that the processor boundary interface elements live only in processors with lower processor Id

DG Parallel Initialization – 6 Basic Steps

- Starting from a partitioned CG (i.e. fully connected) mesh
- 1. Break up the mesh in all the processors introducing new nodes to each element. Recreate the nodal connectivity of the elements with the newly assigned local node Ids.
- 2. Renumber the global Ids of all the nodes across the processors.

DG Parallel Initialization – 6 Basic Steps cont.

- 3. Insert the interface elements at all interior facets in each processor.
- 4. Insert the interface elements at all boundary facets which live in a neighboring processor with a higher processor Id. This requires the creation of new nodes in this processor which also exist in the neighboring processor.
- 5. Transfer the global Ids from the neighboring processor to label these newly created processor boundary nodes.

DG Parallel Initialization – 6 Basic Steps cont.

- 6. Recreate the C++ object called the Communication Maps in each processor

Schematic of Parallel DG Initialization

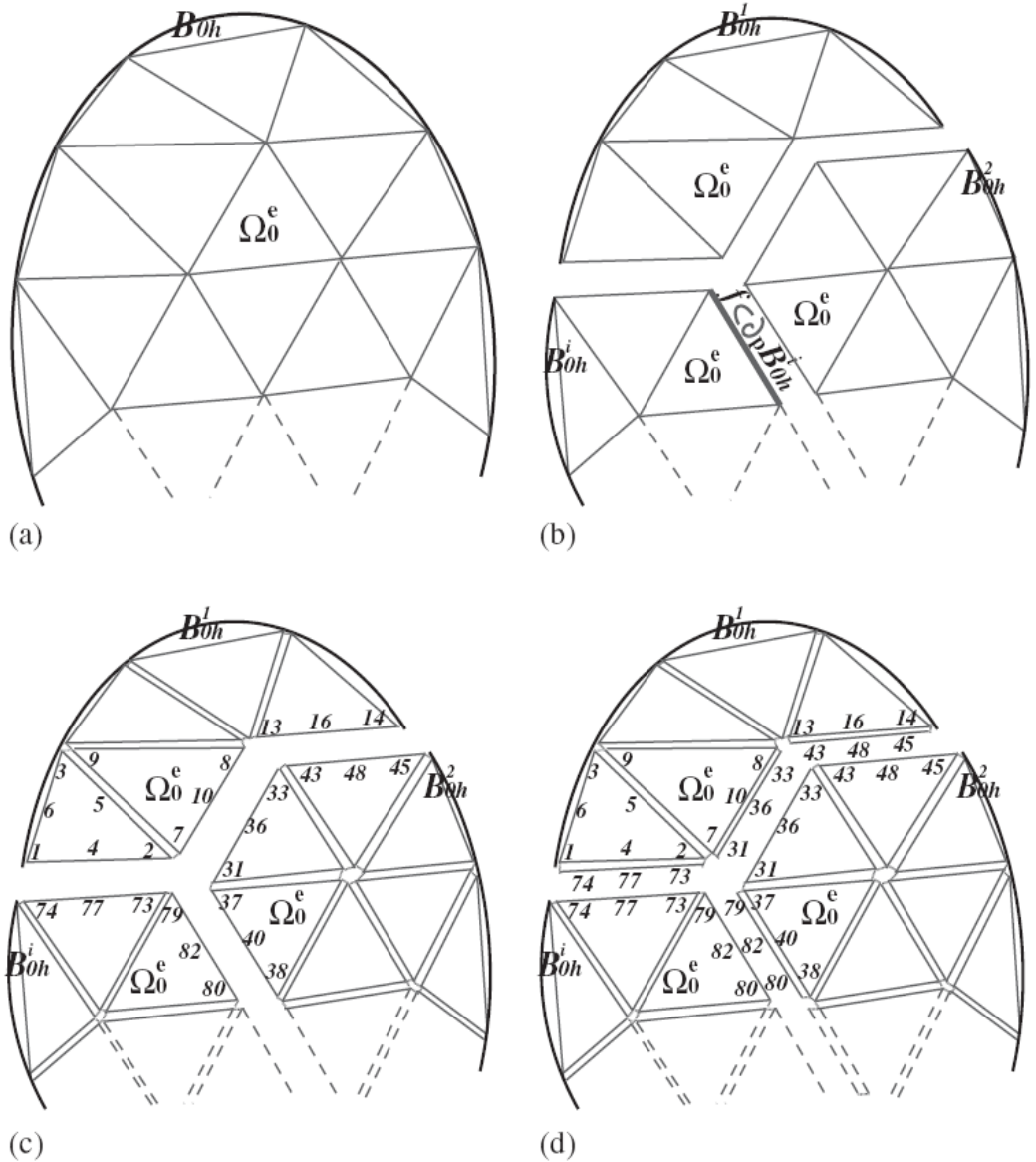


Figure 3. Creation of the partitioned discontinuous mesh (schematic): (a) initial discretization; (b) partitioned mesh; (c) interfaces inside partitions; and (d) interfaces between partitions.

Communication Maps C++ Object

- A container defined in each processor which holds a set of lists of the local node Ids of the nodes which live in other processors.
- Each neighboring processor contains a matching list with the same “ordering” but its own local Ids
- These Communication Map structures allow for send and receive buffers to be communicated automatically between neighboring processors during the calculation which contain information concerning the shared boundary nodes

Communication Maps C++ Object

- In the DG initialization, the old Communication Maps which are created for the CG mesh are no longer valid to describe the DG mesh
- These Maps must be reconstructed in each processor to reflect the local Ids of the newly shared boundary nodes.
- Although they must be reconstructed in the last step, I still make use of the old Communication Maps to do the parallel DG initialization!

Winged Facet C++ Object

- A C++ object created in each processor for the CG initialization which serves as a data structure to define each facet within the processor
- This structure points to the two adjacent tetrahedra to the facet if internal, and to the one adjacent tetrahedra if external
- This structure also stores the six old node Ids belonging to each facet in the original CG mesh
- This structure is critical for the parallel DG initialization!

Notes On The Old Parallel DG Initialization

- Based on adaptive algorithm which was designed originally to insert interface elements at arbitrary element facets within the domain dynamically during the calculation
- Extremely costly data structures are utilized by the algorithm which we do not need
- The algorithm does everything in an incremental fashion when this is unnecessary
- Hence it is extremely slow!

A New Serial DG Initialization Algorithm

- Developed by my advisor.
- Uses only the original Winged Facets, and CG mesh information (thus avoiding the creation and usage of unnecessary, and costly data structures) to
  - 1. Break up the mesh inside of a single processor, creating the new connectivity array from scratch
  - 2. Insert the interface elements at each interior facet inside of the processor
- This algorithm is orders of magnitude faster than the original algorithm on a single processor

Extending the Serial Algorithm to the Parallel Case

- The main idea of this serial algorithm is that it uses the minimum amount of information required to initialize the mesh.
- Keeping with this paradigm, I tried to directly extend this algorithm to the parallel case using only the information available after the CG partitioning
- I found that by utilizing the old Communication Maps, I was able to extend this algorithm to insert the interface elements at the processor boundaries (i.e. step 4)

Successes In Extending the Serial Algorithm to the Parallel Case

- I modified my advisor's serial algorithm to also insert the interface elements at the appropriate facets on the processor boundary which live in a neighboring processor with a higher Id (step 4)
- This is done by searching for the 6 old node numbers of the boundary Winged Facets in each of the old Communication Maps for processors with higher Id
- If these node numbers are all located in one of these Communication Maps, then an interface element is created at that boundary facet

Difficulties in Extending the Serial Algorithm to the Parallel Case

- Although I was able to extend the serial algorithm to complete step 4 of the parallel initialization in a straightforward manner, I soon ran into a huge hurdle presented by the following paradox:
  - Starting with my implementation for completing steps (1-4) in the parallel initialization, step (5) cannot be completed easily without step (6) while step (6) cannot be easily completed with doing step (5)

Difficulties in Extending the Serial Algorithm to the Parallel Case

- This paradox can be restated simply as:
  - The newly recreated Communication Maps are required to transfer the global Ids between processors, while on the other hand the global Ids must have been transferred already to recreate the Communication Maps in any straightforward manner.

A Possible Solution

- In each processor, all of the coordinates for the nodes of each boundary facet living in a lower rank partition must be assembled into arrays which are sent to the lower partitions
- The lower partitions must receive these arrays and then match their new local boundary nodes to the coordinate sets sent in this buffer
- The global Ids of these nodes must also be assembled into arrays with the same ordering as the coordinates and sent to the lower rank processors
- The lower rank processors can then match each of their new local nodes to the corresponding global Id in the receive buffer thus completing step (5)
- Given step (5), the global Ids can be used to recreate the Communication Maps in each processor

Issues With This Approach

- There will be a cost incurred in assembling the send and receive buffers and in passing the messages
- Probably worse, the nodes will have to be matched in the lower rank processor using coordinate searches which are notoriously slow
- The implementation of this algorithm is also not so easy to do since it requires a lot of coordinated message passing

Conclusions and Future Work

- I was able to successfully extend my advisor's serial algorithm to complete step 4 of the parallel case without using any additional information available after the CG partitioning step
- While my implementation should be fast at completing steps (1-4), the further extension of this algorithm to complete steps 5 and 6 is not straightforward
- It may unavoidably require coordinate searches within the processors as I have envisioned
- My next step in developing the parallel initialization is to determine conclusively if it is indeed necessary to do these coordinate searches
- This will probably determine whether or not I continue developing this specific algorithm, or whether I will go back to the drawing board