The RPI team has started looking into the issues associated with making a "thread safe PUMI mesh" and the reordering of unknowns. We were originally thinking that something quick could be done to get us part way towards a thread safe assembly procedure. Although that is true, the performance and scalability will be limited. Thus we do want to start the discussion on the more extensive developments that will be required since it should involve the solvers tram from the beginning.

Providing ability to query a given mesh stored in PUMI in a thread safe manner requires no major developments since existing data is simply being accessed, not being modified. This is the case for M3D-C1 analysis runs (without mesh adaptation); the mesh topology and associated field data is 'read only'. Thread safe mesh modification operations are significantly more challenging. (We have a different version of our structures and adaptation procedure to support this, but other than lessons learned, it is not of importance for the current discussion/needs.)

The real issue is avoiding race conditions when writing matrix and vector entries during a threaded assembly of the global matrix and right hand side.

The current assembly procedure uses MPI based parallelism to assemble the element stiffness matrix terms into the global matrix based on a partitioned mesh directly using the solver assembly technologies.

Moving forward, a thread safe assembly procedure will require either:

- Coloring the elements in the partition on that process such that the elements of any specific color share no dof with any other elements of the same color. Thus, any of the elements of a given color can be assembled at the same time as any other element with the same color.

- Define an independent set of elements – that is elements that share no dof with others in that set. Process the elements in that set in parallel. Repeat until all elements are processed. We have an implementation of maximal independent set computation in PUMI.

In terms of a matrix assembly process based on either approach we have two options for the actual assembly.

A. Use solver support for threaded assembly. One possible approach would involve the normal preprocessing required to tell the solver what they need to know to set-up the structures for the "matrix" and vector to be assembled, followed by color-by-color (or set-by-set) processing using solver controlled threads, possibly using a callback mechanism to get the element contributions, or simply pre-computing them. A brief review of the PETSC and STRUMPACK documentation indicates that threaded assembly is not supported.

B. External creation of the process level assembled portions of the global system. Towards this, we would perform the preprocessing necessary to initialize the matrix and vector structures (as in A), then process each color/set using PUMI/M3D-C1 controlled threads. The process level assembled portions of the system are then passed to the equation solver using single threaded solver APIs.

It is not clear if the first approach is possible, and we would need substantial help from Sherry and/or Sam for the second. Clearly, we very much need Sherry and Sam involved in the decision process. Once we decide on the approach, we can determine who should do which parts and how we coordinate for debugging.

Note that given the level of effort and coding that will be involved to support threaded assembly, we propose doing the equation reordering work (as discussed a few weeks ago) concurrently instead of as a separate phase/effort.