ExaM2M: Scalable and Adaptive Mesh-to-Mesh Transfer

Los Alamos National Laboratory: Jozsef Bakosi
Charmworks: Eric Bohm, Eric Mikida, Nitin Bhat

Quinoa

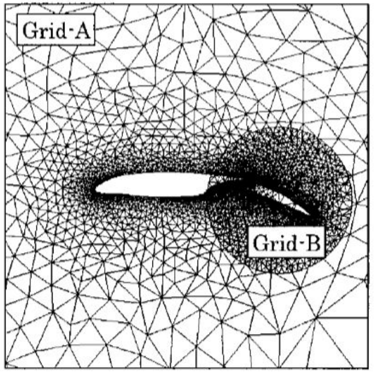
- Quinoa developed by Jozsef Bakosi at Los Alamos National Lab
- Original CFD application developed natively in Charm++
- Open Source license
- http://quinoacomputing.org/

Long Term Goal:

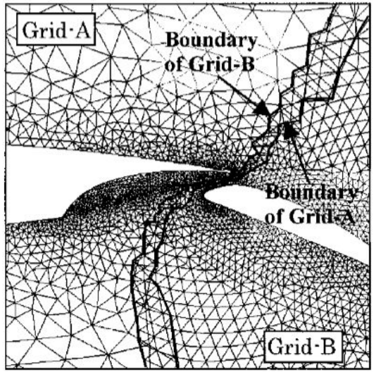
- Combined CFD with Structural Mechanics
  - Parallel collision detection for Mesh-to-Mesh transfer

ExaM2M: Scalable Mesh-to-Mesh Transfer

- Library for performing mesh-to-mesh transfers on unstructured meshes
- Sequential algorithm developed by Jozsef Bakosi
- Parallel version being implemented in Charm++ as a collaborative effort
  - Utilizes an existing collision detection library in Charm++



a) Two unstructured grids are overset



b) Enlarged view after hole cutting

Nakahashi, K., Togashi, F., & Sharov, D. (2000). Intergrid-Boundary Definition Method for Overset Unstructured Grid Approach *AIAA Journal, 38*(11), 2077-2084.

Basic Algorithm

1. Setup mesh data
   - Standalone application: read meshes from file, partition with Zoltan
   - Library: receive mesh data from calling application
   - Mesh data stored in a Worker chare array (virtualized)
2. Pass mesh data to Charm++ collision detection library
   - Source of the transfer submits bounding boxes for each tetrahedron
   - Destination of the transfer submits bounding boxes containing its vertices
   - Returns a list of potential collisions
3. Distribute potential collision list to destination mesh
4. Send vertices that potentially collide to the source mesh
5. Check for actual collisions and interpolate solution

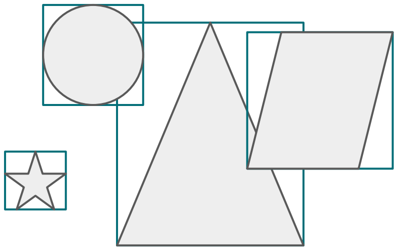
Basic Algorithm

1. Setup mesh data
    - Standalone application: read meshes from file, partition with Zoltan
    - Library: receive mesh data from calling application
    - Mesh data stored in a Worker chare array (virtualized)
2. **Pass mesh data to Charm++ collision detection library**
    - **Source of the transfer submits bounding boxes for each tetrahedron**
    - **Destination of the transfer submits bounding boxes containing it vertices**
    - **Returns a list of potential collisions**
3. Distribute potential collision list to destination mesh
4. Send vertices that potentially collide to the source mesh
5. Check for actual calculation and interpolate solution

Two-Phase Collision Detection

Phase 1 - Broad Phase

- Find potential collisions by colliding bounding boxes
- General case - Handled by library
- Fast to determine potential collisions



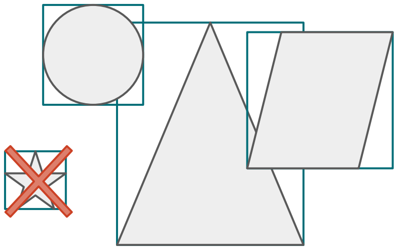
Phase 2 - Narrow Phase

- Weed out false positives from Phase 1
- Application specific
- Fewer collisions to check due to Phase 1

Two-Phase Collision Detection

Phase 1 - Broad Phase

- Find potential collisions by colliding bounding boxes
- General case - Handled by library
- Fast to determine potential collisions



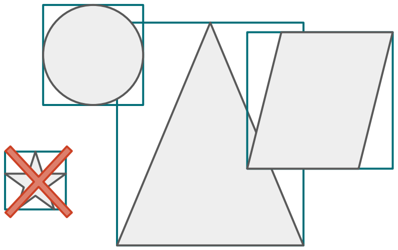
Phase 2 - Narrow Phase

- Weed out false positives from Phase 1
- Application specific
- Fewer collisions to check due to Phase 1

Two-Phase Collision Detection

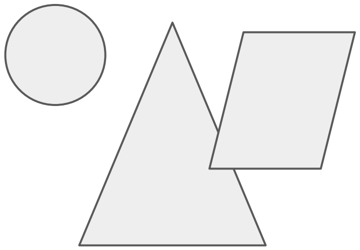
Phase 1 - Broad Phase

- Find potential collisions by colliding bounding boxes
- General case - Handled by library
- Fast to determine potential collisions



Phase 2 - Narrow Phase

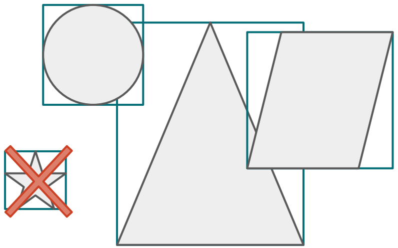
- Weed out false positives from Phase 1
- Application specific
- Fewer collisions to check due to Phase 1



Two-Phase Collision Detection

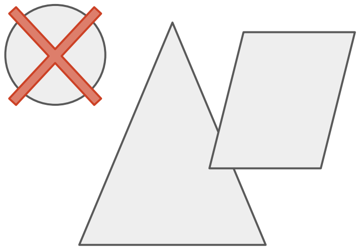
Phase 1 - Broad Phase

- Find potential collisions by colliding bounding boxes
- General case - Handled by library
- Fast to determine potential collisions

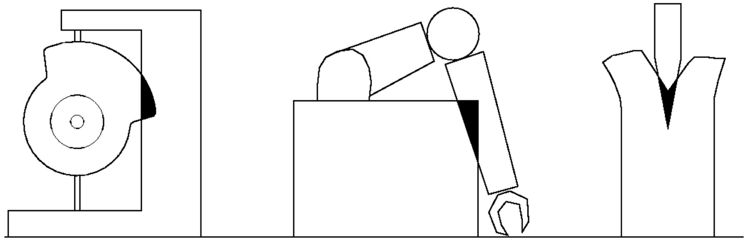


Phase 2 - Narrow Phase

- Weed out false positives from Phase 1
- Application specific
- Fewer collisions to check due to Phase 1



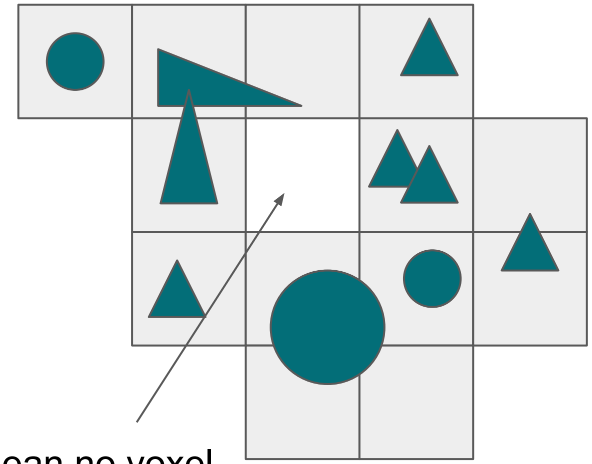
Collision Detection Library



- Detect collisions (intersections) between objects scattered across processors
- Finds applications in many domains: computer graphics, computational physics, robotics, computer aided design etc.
- In our case we are colliding the tetrahedrons of the source mesh with vertices of the destination mesh

Step One: Populate Voxels

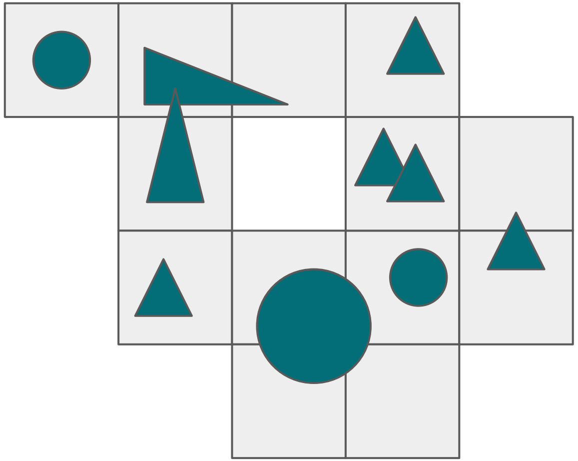
- The collision detection library is based on a sparse grid of voxels
- A voxel is a 3D cell in a regular, axis-aligned, sparse grid
- Voxels are chare array elements that utilize demand creation -- They are created in step 1 of the algorithm as objects are added



No objects mean no voxel

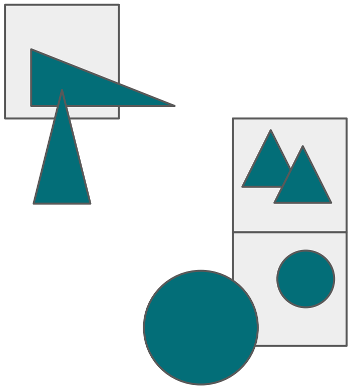
Step Two: Serial Collision Detection

- Voxels run serial collision detection (using bounding boxes) on the objects they know



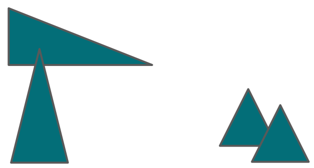
Step Two: Serial Collision Detection

- Voxels run serial collision detection (using bounding boxes) on the objects they know



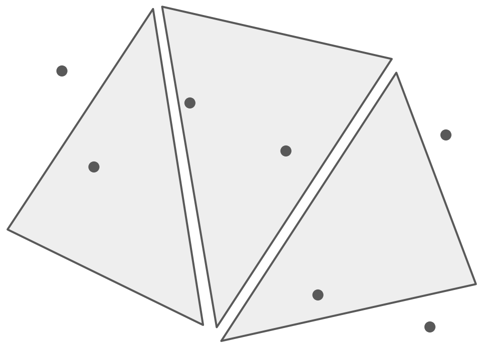
Step Three: Return List of Potential Collisions

- Reduction concatenates a global list of possible collisions to return to caller for further processing



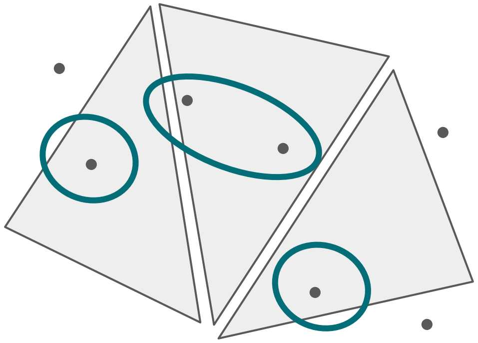
ExaM2M - Narrow Phase Collision Processing

1. Distribute collisions back to destination mesh
2. Each destination mesh chare distributes its own potential collisions to source mesh -- may have multiple potentials per source chare
3. Source mesh determines actual collisions, interpolates solution, and returns results to the destination mesh



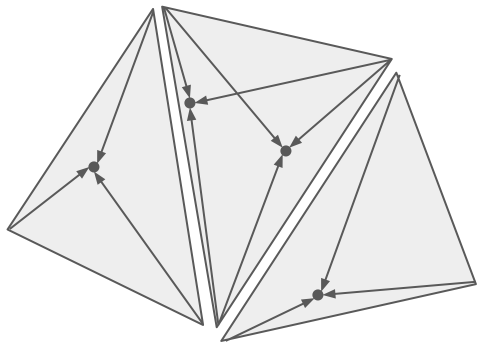
ExaM2M - Narrow Phase Collision Processing

1. Distribute collisions back to destination mesh
2. Each destination mesh chare distributes its own potential collisions to source mesh -- may have multiple potentials per source chare
3. Source mesh determines actual collisions, interpolates solution, and returns results to the destination mesh



ExaM2M - Narrow Phase Collision Processing

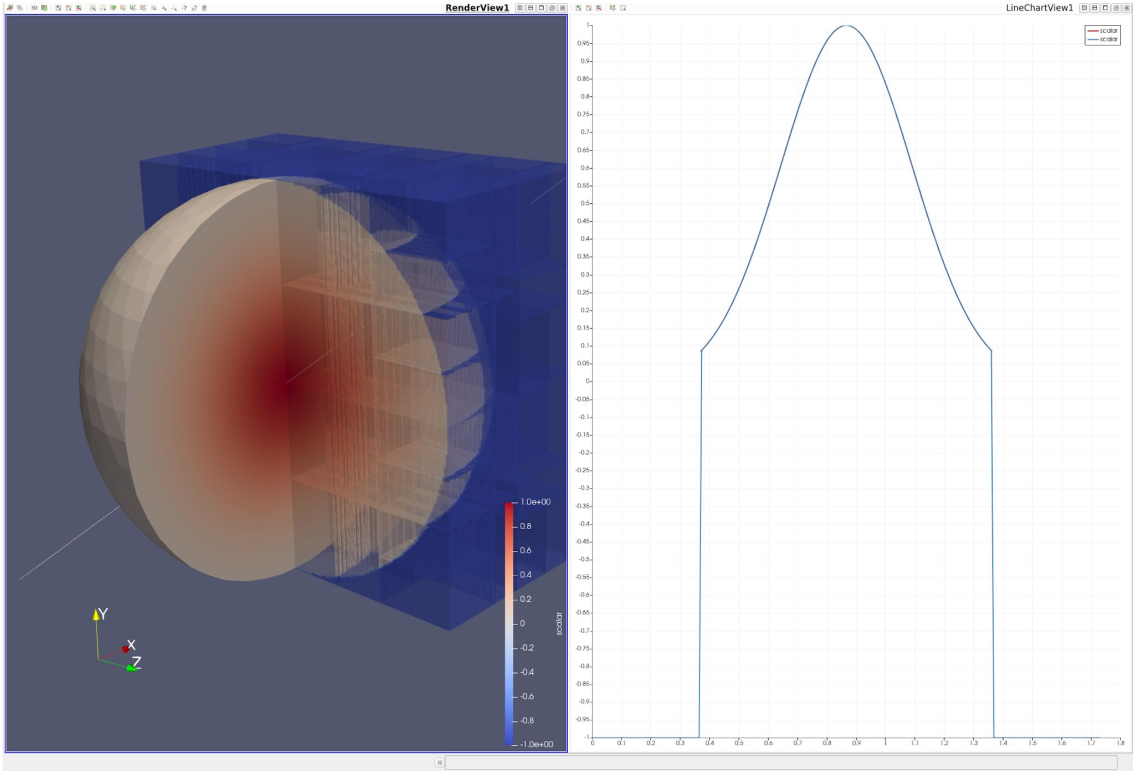
1. Distribute collisions back to destination mesh
2. Each destination mesh chare distributes its own potential collisions to source mesh -- may have multiple potentials per source chare
3. Source mesh determines actual collisions, interpolates solution, and returns results to the destination mesh



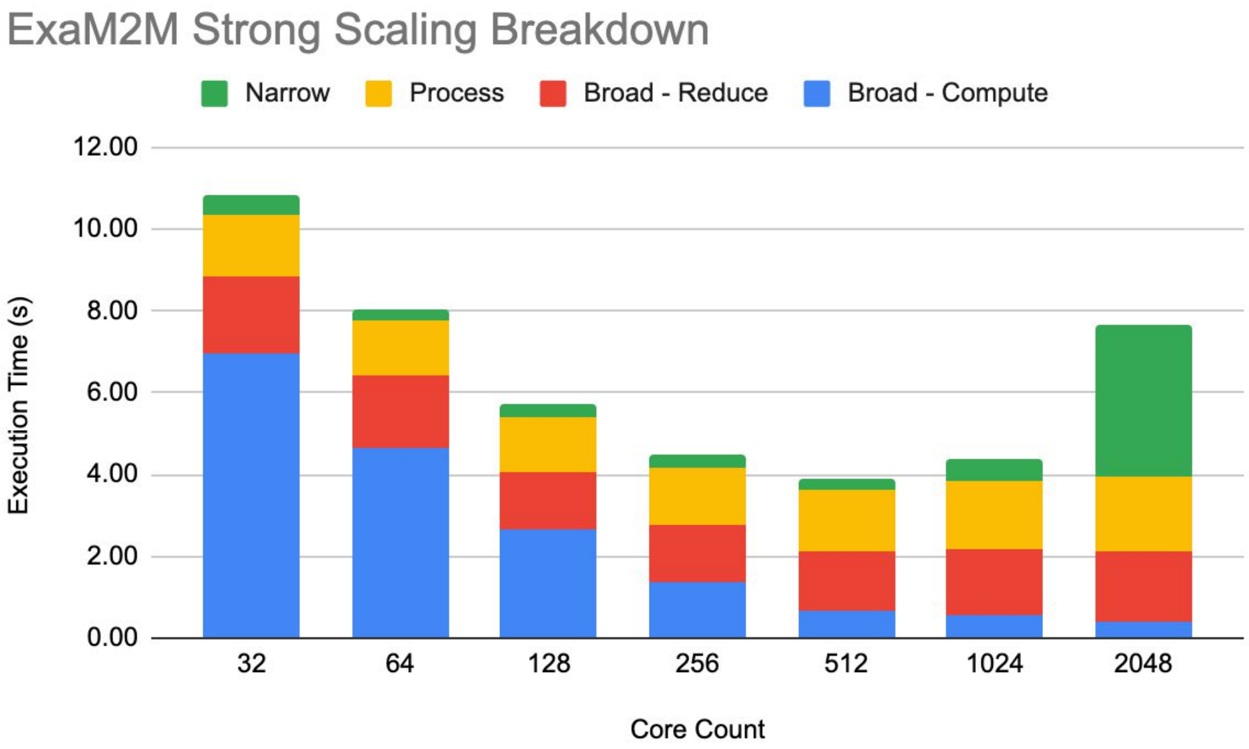
Initial Results

Two 48 million cell meshes

Runs on Cori up to 2048 cores

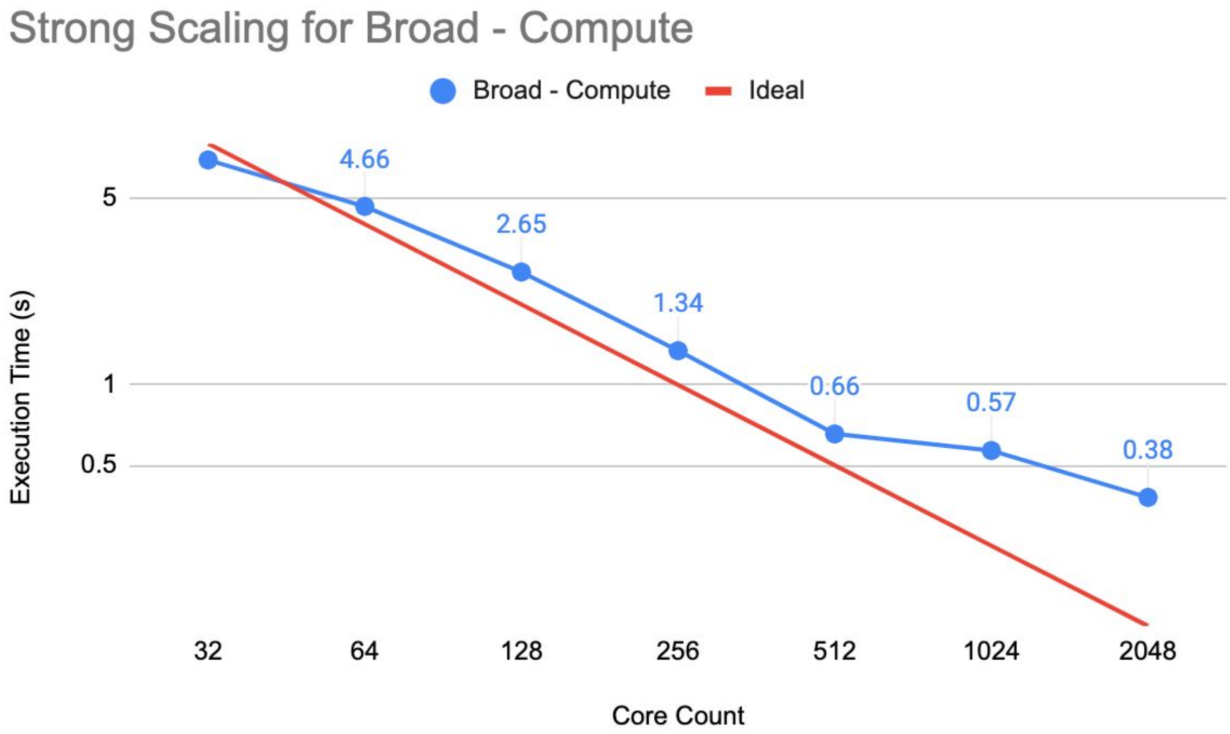


Initial Results - Centralized Collision Reporting



Charmworks

Initial Results - Centralized Collision Reporting



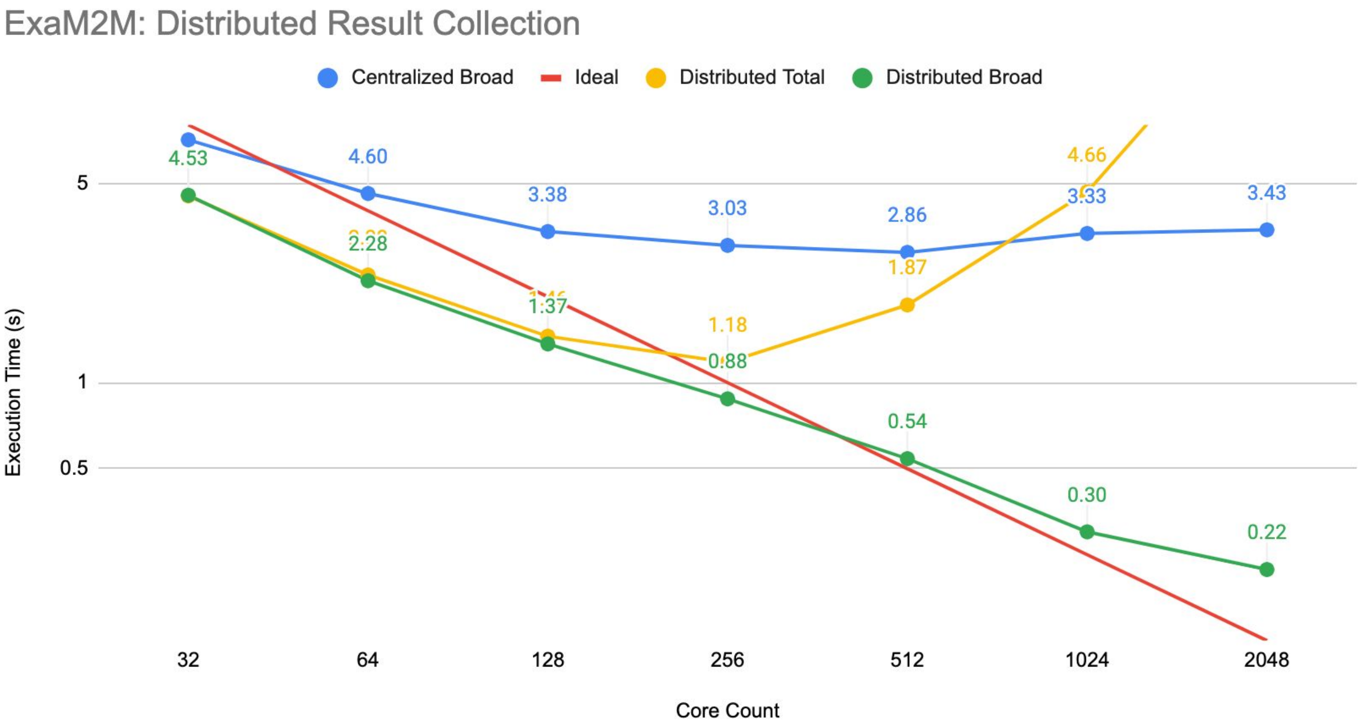
Charmworks

Dealing with the Bottleneck

- Original use case for collision detection library expected very few collisions
- For mesh-to-mesh, many times we expect a lot of overlap
  - Results in a costly reduction, and serial processing of collisions
  - Consumers of this data are distributed, no need to centralize it

Solution: Keep results distributed across PEs, with each PE reporting to the relevant mesh chares

Initial Results



Additional Milestones

- Converted ExaM2M to a library
  - Completion detection still W.I.P.
- Tested within Quinoa
  - Need to add support for multiple iterations
  - Need to add support for multiple meshes
  - Partially asynchronous operation - want fully asynchronous eventually

Future Work

- Large scale tests with Quinoa
- Diagnose and address narrow phase performance issues
- More robust synchronization within the library
- Load balancing w.r.t. (persistent) voxels