SAND2007-6431 C
SAND2007-6431C

An HPC Component for Parallel, Heterogeneous, and Dynamic unstructured Meshes (phdMesh)

H. Carter Edwards
Sandia National Laboratories

CompFrame 2007 (with ooPSLA'07)
October 21-22, 2007
Montreal, Canada

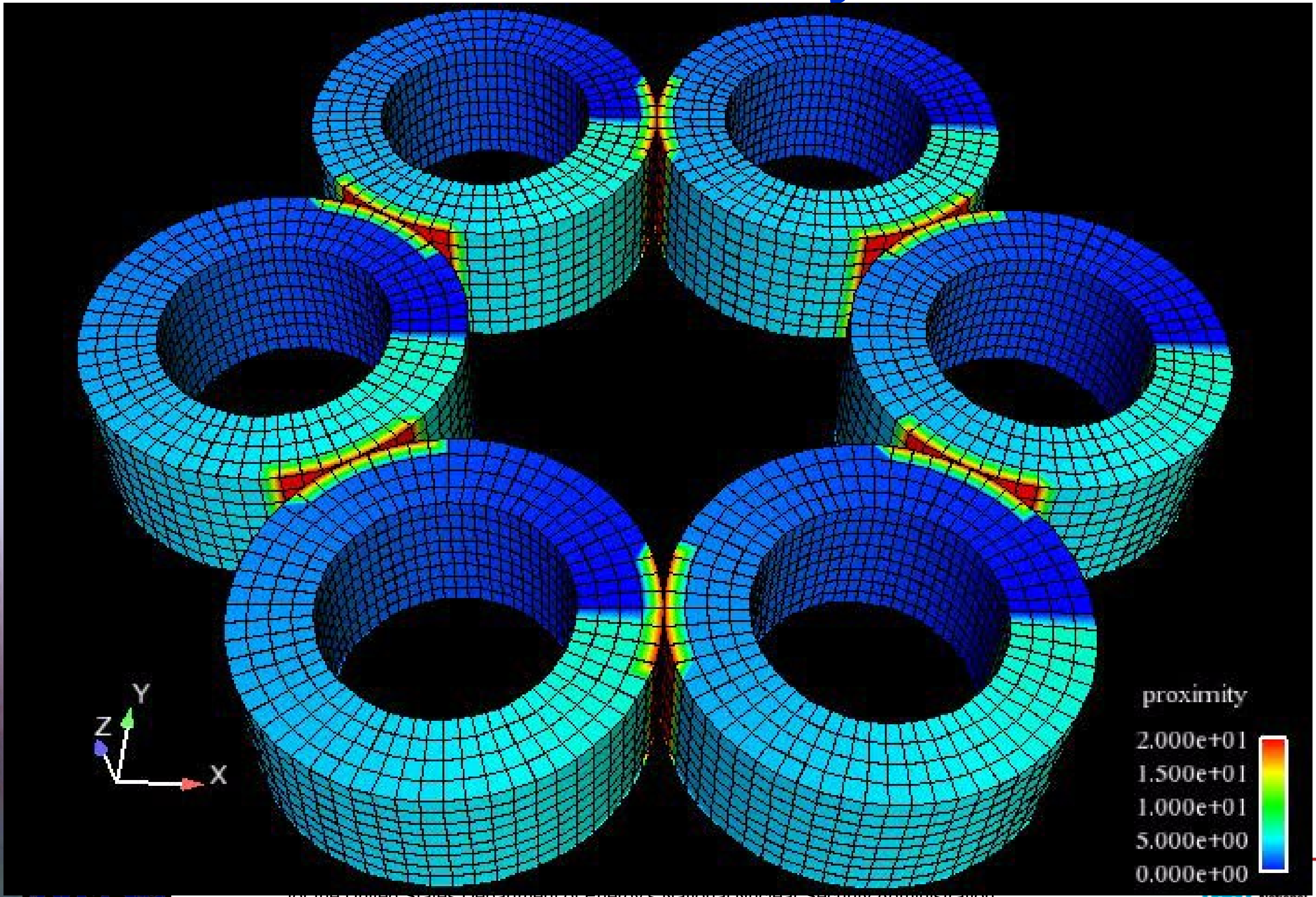
Discovery at the Interface of Science and Engineering: Science Matters!

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

R&D Project Spin-off

- Spin-off from Sandia National Laboratory's R&D project for "HPC Application Performance Analysis and Prediction"
- Purpose: Enable improved decision making for next generation computer systems and applications.
- Approach: Provide targeted compact, highly portable, "mini application" tools that approximate real application performance. Provide guidance to system and application designers. Establish visibility in research community.
- phdMesh is a component developed for a "mini-application" intended to approximate performance of parallel geometric proximity search and dynamic load balancing

Model Problem: Distributed Dynamic Surface-Surface Proximity Detection



for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

phdMesh exceeded its R&D charter

- A full capability (instead of approximate) parallel heterogeneous dynamic mesh library
  - Distributed memory HPC database, not a mesh file format
  - Parallel geometric proximity search algorithm
  - Dynamic load balancing
  - Based upon cumulative concepts and lessons learned from many unstructured mesh data models / projects
  - Two guiding principles:
    1. Keep it simple, i.e. lean & clean
    2. Have a well-defined conceptual model

- Requirements → *conceptual model* → software design → implementation

phdMesh honored intent of R&D charter

- Small, compact, highly portable
- Anticipate leading edge HPC architectures
  - Clusters of manycore nodes
- Available in the public domain
  - Must be easily understood without consultation
  - A new package within Trilinos: http://trilinos.sandia.gov
- Research – think "outside the box"
  - Question a-priori assumptions of previous efforts

Conceptual Overview

Fundamental Concept: Mesh Database

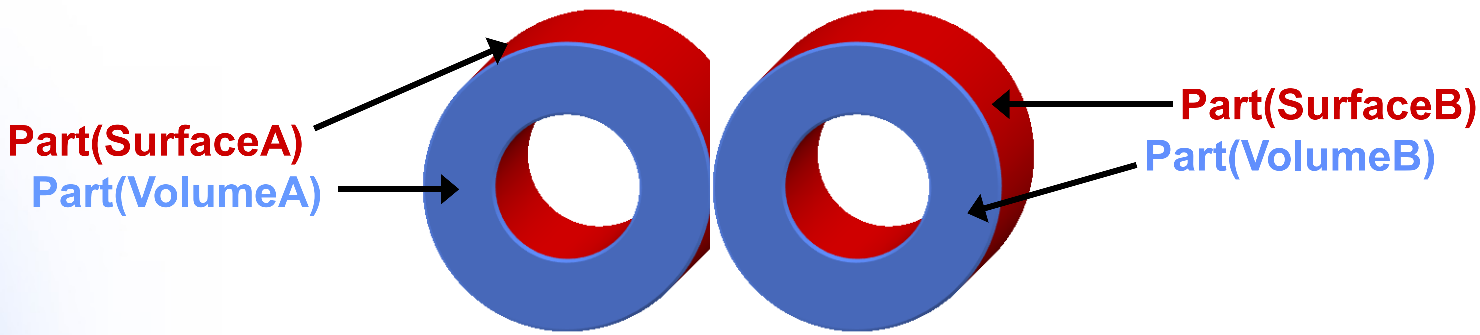
- Mesh Database
  - *Mesh*: weblike pattern or construction that fills a domain Ω
  - *Database*: large collection of data organized for efficient storage, retrieval, and update

- Database = Schema + Bulk Data
  - Schema is the specification for data to be managed
  - Bulk data is stored, retrieved, and updated

➢ **Mesh Schema + Mesh Bulk Data**

Mesh Schema

- A *specification* for the mesh data to be managed
- *Parts* (subsets) of the problem domain, $\{ \Omega_A \subset \Omega \}$
- Parts' subset / superset relationships, $\Omega_A \subset \Omega_S$

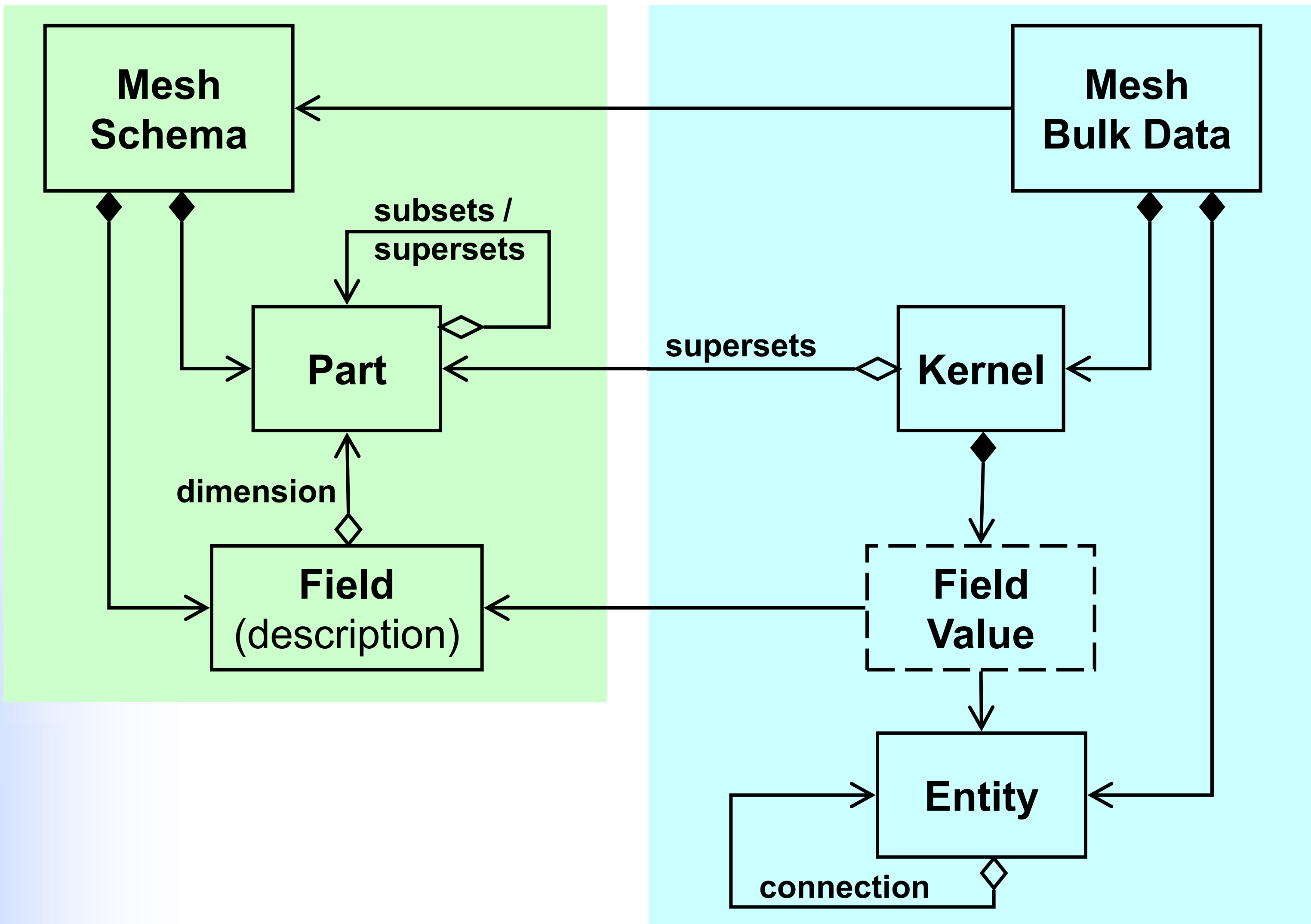


- *Fields* in the discretization, $\{ F_J \}$
  - *Description* of independent, dependent, and auxiliary variables of the problem to be solved over the domain
- Field dimension map, $(F_J , \Omega_A) \rightarrow Dim_{JA}$
  - Variables may have polymorphic dimension

Mesh Bulk Data

- **Large collection of discretization data conforming to a Mesh Schema**
  - **Many possible discretizations for a single schema**
- ***Entities* of a discretization, e.g. nodes, elements**
- ***Connections* between entities, e.g. element→node**
- ***Field Values* associated with entities**
  - **Per-entity numerical values conforming to field descriptions**
  - **E.g. basis function coefficients, state/material properties**
- ***Kernels* – "chunks" of similar field values**
  - **A contiguous block of memory for arrays of field values**
  - **Needed for performance**

High Level Class Diagram



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Conceptual Model: Non-Parallel Portion

Mesh Schema

- ***Part***: application defined subset of the domain
  - Unique text name (for i/o)
  - Subset / superset of other parts, $\Omega_A \subset \Omega_B$
  - Subset relationships are automatically transitive, declare $\Omega_A \subset \Omega_B$ and $\Omega_B \subset \Omega_C$ then $\Omega_A \subset \Omega_C$ is enforced
  - Circular subset declarations detected and erroneous

- ***Field***: application defined variable
  - Unique text name (for i/o)
  - Associated with a type of entity (e.g. node, element)
  - Defines a multidimensional array of a numeric type
  - … with polymorphic dimension-sizes

    field(n1,n2,n3) such that { n1, n2, n3 } may vary

Mesh Schema

- **Field dimension map, $(F_J , \Omega_A) \rightarrow Dim_{JA}$**
  - **Field dimension-sizes vary with the associated part; however, the *number* of dimensions does not vary**

    cardinality, 3 = #{ n1, n2, n3 }, does not vary
  - **Examples: varying degree interpolation polynomal, integration quadrature rules, number of mixed materials**
  - **Invariant when defined on the universal set Ω**

- **Existence and consistency**
  - $(F_J , \Omega_A) \rightarrow$ ***undefined*, until defined by the application**
  - $(F_J , \Omega_A) \rightarrow Dim_{JA}$ and $\Omega_B \subset \Omega_A$ then $(F_J , \Omega_B) \rightarrow Dim_{JA}$
  - $(F_J , \Omega_B) \rightarrow Dim_{JB}$ and $\Omega_B \subset \Omega_A$ then
    - $(F_J , \Omega_A) \rightarrow$ ***undefined* OR**
    - $(F_J , \Omega_A) \rightarrow Dim_{JB}$

Mesh Bulk Data

- ***Entity*: an "atomic" member of a discretized domain**
  - **Node, edge, face, element/cell, constraint, etc.**
  - **Unique identifier global from creation to deletion**
  - **Member of universal set Ω and other parts, $m \in \{ \Omega_A \}$**

- ***Connection*: $m_D \xrightarrow{\alpha} m_R$**
  - **α = (purpose , identifier)**
  - **A relation (domain $\rightarrow$ range) with a purpose**
    - Element uses node ; node usedBy element
  - **Application defined connection-identifier**
    - Relevant with respect to the domain entity
    - Element uses nodes for vertices, id = #0, #1, #2, #3, …
  - **Connection data: $m_D \rightarrow \{ (\alpha , m_R) \}$**

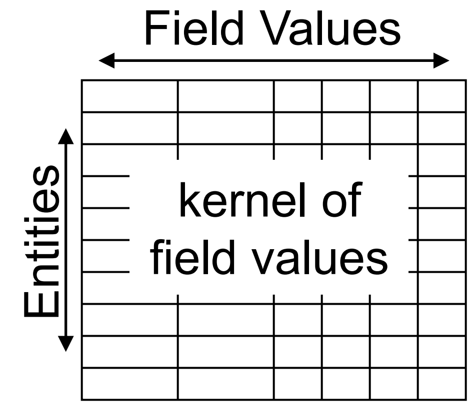
Mesh Bulk Data

- **Field Value: $v_{iJ} \rightarrow (m_i , F_J)$**
  - **Value associated with an entity and defined by a field**
  - **Multidimensional array defined via field dimension map**

- **Existence and polymorphic dimension:**
  - **If** $\exists (F_J , \Omega_A) \rightarrow Dim_{JA}$ and $m_i \in \Omega_A$ then
    $\exists v_{iJ} \rightarrow (m_i , F_J)$ with dimension Dim_{JA}

- **Non existence:**
  - **Let** $m_i \in \{ \Omega_A \}$, **If** for all Ω_A $! \exists (F_J , \Omega_A) \rightarrow Dim$ then
    $! \exists v_{iJ} \rightarrow (m_i , F_J)$
  - **Only allocate storage for field values that exist**

Mesh Bulk Data



- **Kernel**
  - **Homogeneous collection of field values**
  - **Entities: same type and part membership**
  - ➢ **Field values have same dimension**
- **Kernel is a subset: $\Omega_K \subset \Omega$**
  - **Defined by the intersection of a set of parts: $\Omega_K \rightarrow \cap\ \Omega_A$**
  - **Application defined upper bound for #entities $\in \Omega_K$**

    field(n1 , n2 , n3 , #entities) ; #entities $\leq$ bound
  - **Natural "chunk" for task-level parallelism (multicore)**
- **Kernel-based algorithm**
  - **Outer loop to select kernels, e.g. if $\Omega_K \subset \Omega_A$**
  - **Thread/core given (computation , Ω_K) for inner loop**

Mesh Bulk Data Modifications

- Create and delete entities
  – Insert into and remove from the universal set
- Modify entities
  – Insert into, and remove from, parts' subsets
- Create and delete connections
  – Uses / usedBy converse connection automatically created or deleted
- Field Values / Kernels automatically created, deleted, resized
  – As entities created, deleted, or modified
  – As defined by the field dimension map

Connectivity Induced Subsets

- $uses(m_D) = \{ m_R : \exists\, m_D \xrightarrow{uses} m_R \}$
- $usedBy(m_R) = \{ m_D : \exists\, m_D \xrightarrow{uses} m_R \}$
- $uses(\Omega_*) = \Omega_* \cup \{ uses(m_D) \ \ \forall\, m_D \in \Omega_* \}$
- $usedBy(\Omega_*) = \Omega_* \cup \{ usedBy(m_R) \ \ \forall\, m_R \in \Omega_* \}$
- $patch(\bullet) = uses(\ usedBy(\ uses(\bullet)\ )\ )$
  - will revisit ‘patch’ in parallel distribution discussion

Discovery at the Interface of Science and Engineering: Science Matters!

Conceptual Model: Parallel Portion

Parallelization Principles

- **Assumption: Distributed Memory Parallelism**
  - **Not to preclude local task-level (multicore) parallelism**
- **Mesh Schema: replicated on all processors**
- **Mesh Bulk Data: partitioned and distributed among processors**
- ***Minimize impact of global data distribution and local data ordering on algorithms***
  - **Algorithms "see" minimal parallel bookkeeping**
  - **Parallel decomposition and local data ordering *should not effect results,* but may effect performance**
  - **Ideally results are insensitive to global & local parallelism**

Distribution of Mesh Bulk Data (entities)

- A mesh entity *resides* on one, or more, processors
- *Globally unique identification* of an entity, independent of where the entity resides
- *Ownership* of an entity by *one processor* on which it resides
- *Sharing* of an entity among two or more processors that use the entity
  - Processors are “equals” in computations with that entity
- *Aura* for a processor’s entities
  - Concise definition for the concept of parallel ghosting
  - Provide decomposition-independent neighborhoods

Parallel Distribution Subsets (Parts)

- $\Omega_{OWNS}(p)$ = the set of entities owned by processor 'p'
- $\Omega_{USES}(p)$ = *uses*($\Omega_{OWNS}(p) \cup$ { selected entities })
- $\Omega_{RESIDE}(p)$ = *uses*(*usedby*($\Omega_{USES}(p)$))
  = *patch*($\Omega_{OWNS}(p) \cup$ { selected entities })
- $\Omega_{SHARES}(p,q) = \Omega_{USES}(p) \cap \Omega_{USES}(q)$
- $\Omega_{AURA}(p) = \Omega_{RESIDE}(p) \setminus \Omega_{USES}(p)$

Parallelization Concepts: Why an Aura?

- Pervasive need for "ghosting"
  - Node-patch computations
  - Upwinding computations
  - Mesh adaptation propagation (refinement, death-boundary)
- Every member of $\Omega_{USES}(p)$ has a full neighborhood
- *Opportunities* for parallel-insensitive algorithms
  - Push summation is parallel-sensitive
    - Element computations sum into connected nodes' field value
    - Parallel swap-add at the end
    - Sensitive to element decomposition and local ordering
  - Pull summation is parallel-insensitive
    - Save partial sums in elements' field value (scratch variable)
    - Parallel copy of element field value to aura elements
    - For each node sum in a prescribed order (e.g. element id)

Parallel Connections

- Connect $(m_D , p) \rightarrow (m_R , q)$
  - Clarify which processor
  - $m_D \in \Omega_{RESIDE}(p)$
  - $m_R \in \Omega_{RESIDE}(q)$
- Sharing : $m_D = m_R$ and $m_D \in \Omega_{SHARES}(p,q)$
  - Coordinate "uses" actions on shared entities
- Aura : $m_D = m_R$, $m_D \in \Omega_{OWNS}(p)$, $m_R \in \Omega_{AURA}(q)$
  - Update aura entities from their owners

Parallel Support Tools (functions)

- **Discover sharing tool**
  - Discover sharing by matching entities' global identifiers
  - No a-priori knowledge of the decomposition required
  - Uses a generalized *parallel indexing* operation
- **Generate / regenerate aura tool**
  - Push patches of shared entities to sharing processors
- **Load balance tool**
  - Load balance elements, throwing away sharing & aura
  - then rediscover sharing
  - then regenerate aura
  - Simple and robust
  - Performance vs complicated update of sharing and aura?

Conclusion

- A "lean & clean" (a.k.a. minimalist) approach to the challenge of a parallel, heterogeneous, dynamic unstructured mesh in-memory database
- Support distributed-memory parallelism
  - Replication of mesh schema
  - Distribution of mesh bulk data
- Address local performance: *kernels* of field values
  - Field value arrays grouped into memory "chunks"
  - Cache utilization
  - Anticipate manycore task-based parallelism
- Soon to be available at trilinos.sandia.gov