Interfaces: The Key to Code Reuse and Interoperability

CCTTSS Tutorial Working Group

http://www.cca-forum.org/tutorial/

This work has been sponsored by the Mathematics, Information and Computational Sciences (MICS) Program of the U.S. Dept. of Energy, Office of Science.

Contributors

- *David Bernholdt, ORNL (Presenter)*
- Jim Kohl, ORNL
- Lori Freitag, ANL

Interfaces are Central

- In the CCA, components interact *only* through their interfaces
- Both syntax and semantics of interface important
  - CCA tools currently only express syntax
  - Semantics via interface documentation (i.e. SIDL comments)
- Components which provide the same interface are interchangeable
- Interfaces need not be “standardized”, but…
- Interfaces which are widely used promote reuse and interoperability of components
  - Standardization within a given scientific domain
    - i.e. CFD, computational chemistry, nuclear physics, …
  - Standards for cross-cutting functionality/services
    - i.e. solvers, scientific data objects, communications, …

Interfaces as Community Standards

- Communities of interest with the relevant background and expertise are the right people to develop and endorse domain-specific and cross-cutting interfaces
- The CCA Forum and CCTTSS…
  - Develop and endorse standards pertaining to the core CCA specification (i.e. framework services)
  - Domain-specific and cross-cutting interfaces are *not* part of the formal CCA specification
  - Promote and assist in the development of domain-specific and cross-cutting interfaces
  - CCTTSS R&D Thrust in Scientific Components includes interface development activities
- Design of “standard” interfaces requires time and effort; payoff is for community rather than individual

Current Interface Development Activities

CCA Forum Scientific Data Components Working Group

- **Basic Scientific Data Objects**
  - Lead: David Bernholdt, ORNL
- **Structures and Unstructured Meshes**
  - Lead: Lori Freitag, ANL
  - in collaboration with TSTT (SciDAC ISIC)
- **Structured Adaptive Mesh Refinement**
  - Lead: Phil Colella, LBNL
  - in collaboration with APDEC (SciDAC ISIC)

Other Groups

- **Equation Solver Interface (ESI)**
  - Lead: Robert Clay, Terascale
  - Predates CCA, moving towards Babel compliance, possibly CCA compatability
- **MxN Parallel Data Redistribution**
  - Lead: Jim Kohl, ORNL
  - Part of CCTTSS MxN Thrust
- **Quantum Chemistry**
  - Leads: Curt Janssen, SNL; Theresa Windus, PNNL
  - Part of CCTTSS Applications Integration Thrust

Scientific Data Components Working Group Approach

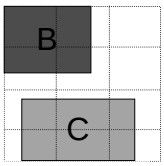
- Agreement on representations for basic scientific data types facilitates design of interoperable components
- Focus on uniform descriptions of existing data rather than forcing new data types on code
  - Facilitates use with existing codes
- Later, develop “native” CCA data types for use in from-scratch component development
- Focus on local & distributed arrays, meshes, etc.

Basic Scientific Data Objects

- Raw Data Interface (Data Working Group)
  - Tentative definition, no implementation
- Local Array Interface (Data Working Group)
  - Dense rectangular multi-dimensional arrays
  - Tentative definition, no implementation
- **Distributed Array Descriptor Interface**
  - Dense rectangular multi-dimensional arrays
  - Fairly mature definition (Bernholdt + Data Working Group)
    - Supports HPF 2.0 capabilities and more
    - Compatible w/ CUMULVS, Global Arrays, SCALAPACK, PETSc, ESI, …
  - Implemented for SC01 demo applications (Bernholdt)
  - Developing interface revisions based on experience, SIDLization (Elwasif)
- "Native" CCA Distributed Array Interface
  - Planned based on DAD

Distributed Array Descriptor

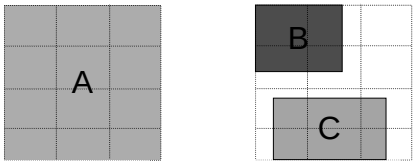
- Object-oriented, programmatic approach based on HPF-2 distributed array model, extended
- **Array template**
  - Template for distribution of array across processes
  - Virtual distributed array, no data type, no storage
  - Any number of actual arrays can be aligned to any part of a template
- **Array descriptor**
  - Associates actual array with distribution template
  - Specifies local data location, local data strides, type
- Interface follows factory design pattern
  - Factory ports allow you to create, clone, destroy templ./descr.
  - Templates, descriptors are standalone objects with interfaces defined by the spec
  - Descriptors passed among components as a way of exchanging distributed arrays



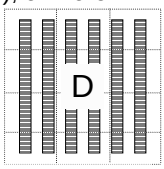
Templates

- Name, rank, global bounds, process topology
- Distribution type & parameters
  - Axis-wise:
    - Collapsed
    - Block (incl. cyclic, block-cyclic)
    - Generalized block (one per process)
    - Implicit (a la HPF)
  - Explicit (tile with arbitrary rectangular multi-dimensional regions)

Descriptors



- Name
- Template
- Process coordinates
- Data type
  - Arbitrary complex data types based on MPI types approach
- Alignment map
  - Actual array may cover all of template (A), a subset (B, C), or more arbitrary mappings (D)
- Local region info
  - Bounds (global coordinates)
  - Strides (compat. w/ C and Fortran native arrays)
  - Pointer to data
- Queries for owner of given region, region owned by given process



Dist. Array Descriptor Notes

- Interface somewhat "verbose" for simple distributed arrays, but…
- Generality requires current complexity
- Assumes local regions are in contiguous memory blocks
- Strides provide flexibility, language independence
- Accommodates, but does not explicitly support, ghost regions, etc.

- *Partial implementation for SC01 demo apps*
  - *Simple types only, identity alignment maps only, minimal query capability, minimal error checking*
- *Currently working towards SIDL-based version*
- *For more info contact David Bernholdt, ORNL*

Meshes: Structured and Unstructured

- Goal: To provide common interfaces for mesh and geometry query and modification
  - **Hybrid solution strategies**
    - **Different mesh and element types in different parts of the computational domain**
    - **Different meshes for different physics in the same domain**
  - **Plug and play experimentation**
    - **Mesh type**
    - **Discretization scheme**
    - **Adaptation strategy**
- Collaboration of CCA and TSTT SciDAC centers
  - **TSTT brings together mesh and discretization expertise from ANL, BNL, LLNL, ORNL, PNL, SNL, RPI, SUNY Stony Brook, and Terascale**

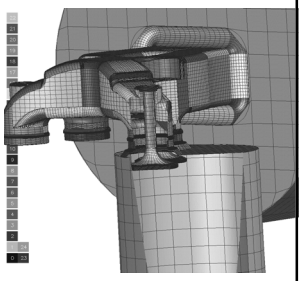
Existing TSTT Tools

A wide variety of tools exist for the generation of …

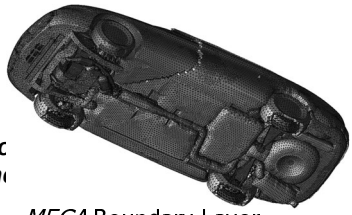
- … structured meshes
  - **Overture** - high quality predominantly structured meshes on complex CAD geometries (LLNL)
  - Variational and Elliptic Grid Generators (ORNL, SNL)
- … unstructured meshes
  - **MEGA** (RPI) - primarily tetrahedral meshes, boundary layer mesh generation, curved elements, AMR
  - **CUBIT** (SNL) - primarily hexahedral meshes, automatic decomposition tools, common geometry module
  - **NWGrid** (PNNL) - hybrid meshes using combined Delaunay, AMR and block structured algorithms

These tools all meet particular needs, but they do not interoperate to form hybrid, composite mesh

Need Geometry and Mesh Hierarchies



Overture Diesel Engine Mesh (LLNL)



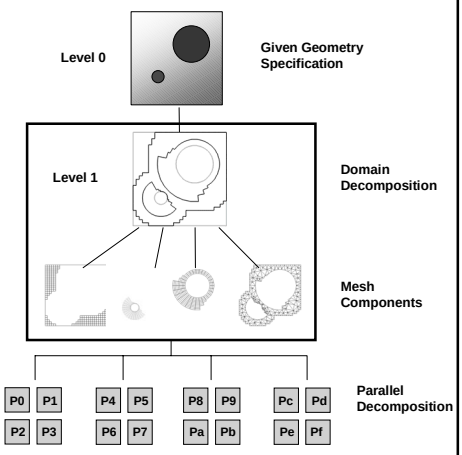
MEGA Boundary Layer Mesh (RPI)

Geometric Hierarchy

Required to

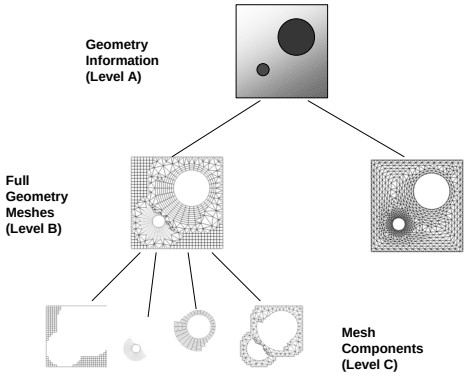
- provide a common frame of reference for all tools
- facilitate multilevel solvers
- facilitate transfer of information in discretizations

- Level 0: Original problem specification via high level geometric description
- Level 1/2: Decomposition into subdomains and mesh components that refer back to Level 0
- Level 3: Partitioning



Mesh Data Hierarchy

- **Level A: Geometric description of the domain**
  - Accessed via tools such as CGM (SNL) or functional interfaces to solid modeling kernels (RPI)
- **Level B: Full geometry hybrid meshes**
  - communication mechanisms that link mesh components (key new research area)
  - allows structured and unstructured meshes to be combined in a single computation
- **Level C: Mesh Components**



Access to Mesh Data Hierarchy

- As a single object (high-level common interfaces)
  - Develop functions that provide, e.g.,
    - PDE discretization operators
    - adaptive mesh refinement
    - multilevel data transfer
  - Prototype provided by Overture framework
  - Enables rapid development of new mesh-based applications
- Through the mesh components (low-level common interfaces)
  - Provide, e.g.,
    - element-by-element access to mesh components
    - fortran-callable routines that return interpolation coefficients at a single point (or array of points)
  - Facilitates incorporation into existing applications

Common Interface Specification

- **Initially focusing on low level access to static mesh components (Level C)**
  - Data: mesh geometry, topology, field data
  - Efficiency though
    - Access patterns appropriate for each mesh type
    - Caching strategies and agglomerated access
  - Appropriateness through working with
    - Application scientists
    - TOPS and CCA SciDAC ISICs
  - Application scientists program to the common interface and can than use any conforming tool without changing their code
- **High level interfaces**
  - to entire grid hierarchy which allows interoperable meshing by creating a common view of geometry
  - mesh adaptation including error estimators and curved elements
- **All TSTT tools will be interface compliant**

Challenges

- Basic data representation differs dramatically among existing tools
  - Array-based access
  - Iterator-based access
- The separation of meshes and discretization
  - Higher order node representation; via global array access or entity association
- Language interoperability
  - Most tools in C or C++; most applications in Fortran
  - C-style interface, SIDL, language specific options
- Flexible memory management system
  - User to pass in preallocated arrays
  - Have the underlying tool create memory
- Tradeoffs of efficiency, flexibility, generality

Current Interface Definition

- Defined mesh entities and topologies
  - Vertex, Edge, Face and Region
  - Polygon, Triangle, Quadrilateral, Polyhedral, Tetrahedron, Hexahedron, Prism, Pyramid, Septahedron
- Opaque Objects
  - Mesh_Handle, Entity_Handle, Tag_Handle, Mesh_Error
- Functions
  - Mesh Create, Load, Services, Entity Arrays, Adjacency Arrays, Entity and Adjacency Iterators, Coordinates and Indices, Destroy
  - Entity Type, Topology, Dimension, Adjacencies, Vertex Coords
- Proposed Functions
  - Tag add/get/set/delete
  - Mesh Sets, Submeshes, Higher Order Node Access, Parallel Queries

Status and Information

- Status
  - Most TSTT sites have at least a partial implementation of the interface
  - Prototype interface demonstrated as a CCA-compliant component at SC in time-dependent PDE simulation
  - Biweekly teleconferences to continue specification definition
  - Opportunistic all-hands meetings
  - 10 IMR, Grid2002
  - Internal use and exchange of tools
- Information
  - www.tstt-scidac.org
  - Contact: Lori Freitag Diachin, ANL

MxN Parallel Data Redistribution

- Low-level interface to allow exchange of data between different parallel distributions
  - Support for unit conversion, grid interpolation, etc. to be built on top
- Designed for minimal intrusion on existing code
- Allows “third-party” control of transfer between components
- Extension of CUMULVS & PAWS packages w/ input from various load balancing/partitioning packages
- Approach involves separate MxNDataPort and MxNCtrlPort, and standalone MxNDataHandle object

MxN Port Model -- Setup

1. Participating components register specific data objects for MxN access
   - Currently uses Dist. Array Descriptor to specify data
   - Each object can be read-only, write-only, read-write
2. Controlling component requests comm. schedule linking objects on M and N sides
   - Ultimately, user must know which objects they want to connect
3. Controlling component makes connection between objects on M and N sides
   - Specify synchronization, frequency of transfer

MxN Port Model -- Use

- Controlling component posts requestTransfer for those connections it is interested in
  - Thereafter, that data transferred when ready, according to specified frequency
- Participating components call dataReady for data object when it is in a consistent state, suitable for transfer
  - Each dataReady increments object's "iteration" count – used when periodic transfer requested
  - Data transferred if required
  - dataReady falls through if no transfer required
- Blocking and non-blocking versions available

MxN Port Notes

- Participating components must be "instrumented" with registerData and dataReady calls
  - Minimal overhead if no transfers requested
- Controller chooses which data to connect, how to connect it, and what to actually transfer
- Controller may or may not be a participant
- *Two implementations (earlier version of spec) for SC01 demo apps*
  - *CUMULVS-based, ORNL*
  - *PAWS-based, LANL*
- *Contact Jim Kohl, ORNL for more information*

What Can You Do?

- When designing software, think about interfaces that maximize flexibility and might be useful to others
- Look for “standard” interfaces before creating a new one
- Find and participate in existing interface development efforts relevant to your work
- Where you see an opportunity, start a new community-based interface effort
  - CCA Forum and CCTTSS will help
  - Contact: Lois McInnes, ANL, Scientific Components Focus Lead
- *The best software is code you don’t have to write*
  -- Steve Jobs