Modularity of Lowlevel Forest-of-octree Libraries

Johannes Holke (German Aerospace Center DLR, Cologne)
Carsten Burstedde (University of Bonn)
Outline

Adaptive Mesh Refinement

p4est and t8code

Arbitrary element types
Adaptive Meshes

Uniform

Adaptive
Adaptive Meshes

Adaptive Meshes

Adaptive Refinement

Only refine the mesh where needed.

- The same computational error with less elements
- Mesh management becomes more complicated
Dynamical AMR

The mesh changes (frequently) during the simulation (i.e. every $n$ time steps).

AMR needs to be fast and scalable!
Refinement tree and SFC
Forest of trees

From tree...

- Limitation: Cube-like geometric shapes
Forest of trees

...to forest

- Advantage: Geometric flexibility
- Challenge: Non-matching coordinate systems between octrees
Two meshes: Unstructured **coarse mesh** that describes the geometry. 
**Fine mesh** for the computation.
Two meshes: **Unstructured coarse mesh** that describes the geometry. **Fine mesh** for the computation.

Each coarse mesh cell is a refinement tree. Initial partition of coarse mesh (i.e. METIS) part of preprocessing.
AMR Algorithms

- Input: Coarse mesh
- New
- Adapt
- (Balance)
- Partition
- Ghost
- (Iterate)
- (Search)

Repeat

Coarse Mesh
AMR Algorithms

- Input: Coarse mesh
- New
- Adapt
- (Balance)
- Partition
- Ghost
- (Iterate)
- (Search)

Repeat

Coarse Mesh
AMR Algorithms

- Input: Coarse mesh
- New
- Adapt
  - (Balance)
- Partition
- Ghost
  - (Iterate)
  - (Search)

Coarse Mesh
AMR Algorithms

- Input: Coarse mesh
- New
- Adapt
- (Balance)
- Partition
- Ghost
- (Iterate)
- (Search)

Repeat

New
AMR Algorithms

- Input: Coarse mesh
- New
- Adapt
- (Balance)
- Partition
- Ghost
- (Iterate)
- (Search)

Repeat

Adapt + Balance
AMR Algorithms

- Input: Coarse mesh
- New
- Adapt
- (Balance)
- Partition
- Ghost
- (Iterate)
- (Search)

Partition

Repeat
AMR Algorithms

- Input: Coarse mesh
- New
- Adapt
- (Balance)
- Partition
- Ghost
- (Iterate)
- (Search)

Iterate

Repeat
AMR Algorithms

- Input: Coarse mesh
- New
- Adapt
- (Balance)
- Partition
- Ghost
- (Iterate)
- (Search)
p4est and t8code

p4est

- Developed by C. Burstedde\textsuperscript{a}, T. Isaac\textsuperscript{b}, L. C. Wilcox\textsuperscript{c} and others since \sim 2007
- Quads (2D) and Hexes (3D), Z-Curve
  - Face/Edge/Corner connections
  - Coarse mesh replicated on each rank
  - Scales to > $10^6$ MPI ranks, $10^{12}$ elements
  - Used in deal.ii, petsc, ForestClaw, Rhea, ParFlow, ...

\textsuperscript{a}University of Bonn, \textsuperscript{b} Georgia Tech, \textsuperscript{c} Naval Postgraduate School
**p4est and t8code**

**p4est**
- Developed by C. Burstedde\textsuperscript{a}, T. Isaac\textsuperscript{b}, L. C. Wilcox\textsuperscript{c} and others since \(\sim\) 2007
- Quads (2D) and Hexes (3D), Z-Curve
- Face/Edge/Corner connections
- Coarse mesh replicated on each rank
- Scales to \(> 10^6\) MPI ranks, \(10^{12}\) elements
- Used in deal.ii, petsc, ForestClaw, Rhea, ParFlow, ...

**t8code**
- Developed by C. Burstedde and J. Holke since 2014
- Points/Lines/Quads/Tris/Hexes/Tets/Prisms hybrid, extendable in a modular fashion
- Currently Face connections only
- Coarse mesh partitioned
- Scales to \(> 10^6\) MPI ranks, \(10^{12}\) elements
- Performance comparable to p4est

\textsuperscript{a}University of Bonn, \textsuperscript{b}Georgia Tech, \textsuperscript{c}Naval Postgraduate School
Our philosophy

"Do one thing and do it well."
We do

- Fast and scalable AMR algorithms
- Exchange of communication buffers (packed by application)
- Mesh elements as black box for anything that the application wants
- Topology information
- Iterators over mesh elements and element-to-element interfaces
- Fast local and global search algorithms
- Full flexibility regarding refinement criteria
We do not

- PDE solvers
- Tell you what data to put on mesh elements
- packing/unpacking of data
- Geometry information
- Interpolating data between meshes (but we give you the meshes and corresponding elements)
- Refinement criteria
- Time stepping
Arbitrary element types

Core algorithms (high-level)

- New
- Adapt
- Partition
- Ghost
- Balance
- Iterate
Arbitrary element types

Core algorithms (high-level)

- New
- Adapt
- Partition
- Ghost
- Balance
- Iterate

low-level

- element_refine
- element_parent
- element_id
- element_face_neighbor

...
Arbitrary element types

Core algorithms (high-level)

- New
- Adapt
- Partition
- Ghost
- Balance
- Iterate

Decouple high-level and low-level algorithms
Define API for element-local (low-level) functions
Low-level functions can be exchanged arbitrarily without affecting high-level logic
Realized via C++ pure virtual base class

low-level

- element_refine
- element_parent
- element_id
- element_face_neighbor

...
**Virtual base class**

```cpp
/** This struct holds virtual functions for a particular element class. */
struct t8_class_scheme {

  /** This schema defines the operations for a particular element class. */
  protected:
  t8_class_t t_class;  /// The element class

  /** Return the maximum allowed level for any element of a given class. */
  virtual int t8_element_maxlevel(void) const = 0;

  /** Return the level of a particular element. */
  virtual int t8_element_level(const t8_element_t * elem) const = 0;

  /** Copy all entries of \verb|\b| source to \verb|\b| dest. \verb|\b| dest must be an existing
   * element. No memory is allocated by this function.
   * \verb|\b| elem and \verb|\b| dest must point to the same element.
   */
  virtual void t8_element_copy(const t8_element_t * source, t8_element_t * dest) const = 0;

  /** Compute the parent of a given element \verb|\b| elem and store it in \verb|\b| parent.
   * \verb|\b| parent needs to be an existing element. No memory is allocated by this function.
   * \verb|\b| elem and \verb|\b| parent can point to the same element, then the entries of
   * \verb|\b| elem are overwriten by the ones of its parent.
   */
  virtual void t8_element_parent(const t8_element_t * elem, t8_element_t * parent) = 0;

  virtual void t8_element_print(const t8_element_t * elem) = 0;

  /** A virtual function to define the specific operations for this class. */
  virtual int t8_element_maxlevel(void) const = 0;

  /** Return the level of a particular element. */
  virtual int t8_element_level(const t8_element_t * elem) const = 0;

  /** Copy all entries of \verb|\b| source to \verb|\b| dest. \verb|\b| dest must be an existing
   * element. No memory is allocated by this function.
   * \verb|\b| elem and \verb|\b| dest must point to the same element.
   */
  virtual void t8_element_copy(const t8_element_t * source, t8_element_t * dest) const = 0;

  /** Compute the parent of a given element \verb|\b| elem and store it in \verb|\b| parent.
   * \verb|\b| parent needs to be an existing element. No memory is allocated by this function.
   * \verb|\b| elem and \verb|\b| parent can point to the same element, then the entries of
   * \verb|\b| elem are overwriten by the ones of its parent.
   */
  virtual void t8_element_parent(const t8_element_t * elem, t8_element_t * parent) = 0;

  virtual void t8_element_print(const t8_element_t * elem) = 0;
}
```

**Tetrahedra**

```cpp
/** This data type stores a tetrahedron. */
typedef struct t8_tet {
  /** The refinement level of the tetrahedron relative to the root at level 0. */
  int t_level;

  /** Type of the tetrahedron in 0, ..., 5. */
  t8_tet_type_t type;

  t8_tet_coord_t x;  /// The x integer coordinate of the anchor node.
  t8_tet_coord_t y;  /// The y integer coordinate of the anchor node.
  t8_tet_coord_t z;  /// The z integer coordinate of the anchor node.

  t8_tet_t t;

  void t8_default_scheme_tet_c::t8_element_parent(const t8_element_t * elem, t8_element_t * parent) {
    t8_default_tet_t * t = (const t8_default_tet_t *) elem;
    t8_default_tet_t * p = (t8_default_tet_t *) parent;

    t8_ASSERT(t8_element_is_valid(elem));
    t8_ASSERT(t8_element_is_valid(parent));

    t8_dtri_cube_id_t cid;
    t8_dtri_coord_t h;

    t8_ASSERT(t->level > 0);
    h = t8_DTRI_LEN(t->level);
    /* Compute type of parent */
    cid = compute_cubeid(t, t->level);
    parent->type = t8_dtri_cid_type_to_parenttype(cid)[t->type];
    /* Set coordinates of parent */
    parent->x = t->x + h;
    parent->y = t->y + h;
    parent->z = t->z + h;
    parent->level = t->level - 1;
  }
```
Low-level API

Virtual base class

```c
/** This struct holds virtual functions for a particular element class. **/
struct T8_element_scheme {

  /** This class defines the operations for a particular element class. **/
  protected:

  /** The element class */
  t8_element_t *element;

  /** Return the maximum allowed level for any element of a given class. **/
  * return: The maximum allowed level for elements of class T8.
  */
  virtual int T8_element_maxlevel (void) = 0;

  /** Return the level of a particular element. **/
  * pre: [in] elem The element whose level should be returned.
  * return: The level of T8 elem.
  */
  virtual int T8_element_level (T8_element_t * elem) = 0;

  /** Copy all entries of T8 source to T8 dest. T8 dest must be an existing
  * element. No memory is allocated by this function.
  * pre: [in] source The element whose entries will be copied to T8 dest.
  * pre: [in,out] dest This element's entries will be overwritten with the
  * entries of T8 source.
  * post: T8 dest is the same object as T8 source.
  */
  virtual void T8_element_copy (const T8_element_t * source,
                               T8_element_t * dest) = 0;

  /** Compute the parent of a given element T8 elem and store it in T8 parent.**/
  * pre: T8 elem and T8 parent can point to the same element, then the entries of
  * T8 elem are overwritten by the ones of its parent.
  * pre: elem The element whose parent will be computed.
  * pre: elem parent This element's entries will be overwritten by those
  * of T8 elem's parent.
  * post: The storage for this element must exist
  * and each element class of the parent.
  * post: For a pyramid, for example, it may be either a
  * tetrahedron or a pyramid depending on T8 elem's childid.
  */
  virtual void T8_element_parent (const T8_element_t * elem,
                                  T8_element_t * parent) = 0;
};
```

Hexahedra

```c
/* The structure holding a hexahedral element in the default scheme. */

* We make this definition public for interoperability of element classes.
* We might want to put this into a private, scheme-specific header file.
*/

typedef T8_element_t T8_element_t;

void T8_default_scheme_hex_c::T8_element_parent (const T8_element_t * elem,
                                                  T8_element_t * parent)
{
  T8ASSERT (T8_element_is_valid (elem));
  T8ASSERT (T8_element_is_valid (parent));
  *parent = (T8_element_t *) elem,
  (T8_element_t *) parent);
```
Arbitrary element types

Do whatever you want

The decoupling of high- and low-level functions allows the user to implement their own refinement patterns and SFCs.

Element type independency introduces some challenges

Example: Face-neighbors across tree boundaries
Element type independency introduces some challenges

Example: Face-neighbors across tree boundaries

Avoid couplings
Hex↔Prism
Prism↔Tet
Quad↔Tri
...
Example: Face-neighbors across tree boundaries
Example: Face-neighbors across tree boundaries

1: Build 2D boundary element
Example: Face-neighbors across tree boundaries

1: Build 2D boundary element

2: Transform coordinates
Example: Face-neighbors across tree boundaries

1: Build 2D boundary element

2: Transform coordinates

3: Extrude boundary element
Thank you for your attention.

- p4est.org
- github.com/cburstedd/p4est
- github.com/holke/t8code
- My Thesis *Scalable Algorithms for Parallel Tree-based Adaptive Mesh Refinement with General Element Types* on Arxiv
Do not be afraid of hanging nodes

**Hanging nodes?**

To resolve hanging nodes one could either

1. Interpolate in the solver routines.
2. Resolve them with one green refinement step after Adapt and Balance.
Do not be afraid of hanging nodes

Hanging nodes?

To resolve hanging nodes one could either

1. Interpolate in the solver routines.
2. Resolve them with one green refinement step after Adapt and Balance.
Do not be afraid of hanging nodes

Hanging nodes?

To resolve hanging nodes one could either

1. Interpolate in the solver routines.
2. Resolve them with one green refinement step after Adapt and Balance.