EBProto Design Document

Phillip Colella
Daniel Graves
Brian Van Straalen

November 21, 2019

1 Mathematical Description of Embedded Boundary Discretizations

1.1 Graph Description of Embedded Boundary Grids

In structured-grid discretizations of PDE, the topology of the grid is implicit in the definition of the index space $\mathbb{Z}^{\boldsymbol{D}}$. In the case of embedded boundaries, the index space is constructed from the boundary geometry, and can be thought of as a graph, similar to an unstructured-grid finite-volume method, but one for which the nodes and faces map onto the Cartesian grid graph. This more general graph description allows us to provide a uniform embedded boundary representation of a broad range of geometries.

We denote by $\mathcal{G} = \{\mathcal{V}, \mathcal{F}\}$ the embedded boundary graph with nodes $\mathcal{V}$ and arcs $\mathcal{F}$, with $\boldsymbol{f} \in \mathcal{F}$ connecting a pair of nodes $\mathrm{low}(\boldsymbol{f}), \mathrm{high}(\boldsymbol{f}) \in \mathcal{V}$. The connection to the Cartesian grid is given by an *index function* $\mathrm{ind} : \mathcal{V} \rightarrow \mathbb{Z}^{\boldsymbol{D}}$, and $\mathcal{F}$ is given as a disjoint union $\mathcal{F} = \mathcal{F}_0 \bigcup \cdots \bigcup \mathcal{F}_{\boldsymbol{D}-1}$, such that, if $\boldsymbol{f} \in \mathcal{F}_d$, then $\mathrm{ind}(\mathrm{low}(\boldsymbol{f})_{d'}) = \mathrm{ind}(\mathrm{high}(\boldsymbol{f})_{d'}) - \delta_{dd'}$. We also define $\mathrm{ind} : \mathcal{F} \rightarrow \mathbb{Z}^{\boldsymbol{D}}$ by $\mathrm{ind}(\boldsymbol{f}) \equiv \mathrm{ind}(\mathrm{high}(\boldsymbol{f}))$. Geometrically, $\boldsymbol{v}$ indexes a nonempty connected component of $\Omega \bigcap [\boldsymbol{i}h, (\boldsymbol{i} + \boldsymbol{u})h] \equiv \Upsilon_{\boldsymbol{v}}$, and $\boldsymbol{f}$ indexes the face given by $\partial V_{\mathrm{low}(\boldsymbol{f})} \bigcap \partial V_{\mathrm{high}(\boldsymbol{f})} \equiv \mathrm{A}_{\boldsymbol{f}}$. Thus $\mathcal{V}$ represents all the control volumes of the embedded boundary grid, and $\mathcal{F}_d$ represents all the faces between adjacent control volumes in the d-direction.

In addition to topological information, we also need geometric information regarding the intersection of the irregular domain with the Cartesian grid. The motivating example for our approach the finite-volume discretization of $\nabla \cdot \vec{F}$. Applying the divergence theorem over

$\Upsilon_{\boldsymbol{v}}$, we have

$$\frac{1}{|\Upsilon_{\boldsymbol{v}}|}\int\limits_{\Upsilon_{\boldsymbol{v}}} \nabla\cdot\vec{F}d\boldsymbol{x} = \frac{1}{|\Upsilon_{\boldsymbol{v}}|}\int\limits_{\mathrm{A}_{\boldsymbol{v}}^{EB}} \vec{F}\cdot\hat{n}^{EB}dA + \frac{1}{|\Upsilon_{\boldsymbol{v}}|}\sum_d\Big(\sum_{\boldsymbol{f}\in\mathcal{F}^d:\boldsymbol{v}=\mathrm{low}(\boldsymbol{f})}\int\limits_{\mathrm{A}_{\boldsymbol{f}}} F^d dA - \sum_{\boldsymbol{f}\in\mathcal{F}^d:\boldsymbol{v}=\mathrm{high}(\boldsymbol{f})}\int\limits_{\mathrm{A}_{\boldsymbol{f}}} F^d dA\Big) \quad (1)$$

To compute a numerical approximation to the average on the left-hand of (**??**), we replace the surface integrals of the fluxes by the midpoint rule, i.e. by the values of the fluxes at centroids multiplied by the areas.

$$= \frac{1}{\kappa_{\boldsymbol{v}}h}\Big(\vec{F}(\boldsymbol{x}^{EB})\cdot\hat{\boldsymbol{n}}^{EB} + \sum_d\Big(\sum_{\boldsymbol{f}\in\mathcal{F}_d:\boldsymbol{v}=\mathrm{low}(\boldsymbol{f})} F^d(\boldsymbol{x}_{\boldsymbol{f}})\alpha_{\boldsymbol{f}} - \sum_{\boldsymbol{f}\in\mathcal{F}_d:\boldsymbol{v}=\mathrm{high}(\boldsymbol{f})} F^d(\boldsymbol{x}_{\boldsymbol{f}})\alpha_{\boldsymbol{f}}\Big)\Big) + O\Big(\frac{h}{\kappa_{\boldsymbol{v}}}\Big)$$

where we use the *volume fractions* $\kappa_{\boldsymbol{v}} = |\Upsilon_{\boldsymbol{v}}|/h^{\boldsymbol{D}}$; the *area fractions* $\alpha = |\mathrm{A}_{\boldsymbol{f}}|/h^{\boldsymbol{D}-1}$, $\alpha_{\boldsymbol{v}}^{EB} = |\mathrm{A}_{\boldsymbol{v}}^{EB}|/h^{\boldsymbol{D}-1}$; the *centroids* $\boldsymbol{x}_{\boldsymbol{v}}$ of $\Upsilon_{\boldsymbol{v}}$, $\boldsymbol{x}_{\boldsymbol{f}}$ of $\mathrm{A}_{\boldsymbol{f}}$ and $\boldsymbol{x}_{\boldsymbol{v}}^{EB}$ of $\mathrm{A}_{\boldsymbol{v}}^{EB}$; and $\hat{\boldsymbol{n}}_{\boldsymbol{v}}^{EB}$, the unit normal to the boundary evaluated at the centroid . We can obtain higher order methods by using higher order quadrature rules. Given other state variables, fluxes evaluated at centroids are computed as a composition of the stencil operations and pointwise functions that are described below.

The index mapping allows us to define subsets of $\mathcal{V}$ and $\mathcal{F}_d$ corresponding to control volumes and faces contained in a Box $B \subset \mathbb{Z}^{\boldsymbol{D}}$. We denote by

$$\mathcal{V}(B) = \mathrm{ind}^{-1}(B)$$
$$\mathcal{F}_d(B) = \{\boldsymbol{f}\in\mathcal{F}_d : \mathrm{high}(\boldsymbol{f})\in\mathcal{V}(B)\}$$

We also define a Box B to be *regular* if it is in a part of the domain that does not intersect $\partial\Omega$. We define this this in terms of the topological and geometric properties of $\mathcal{G}$ as follows.

- For each $\boldsymbol{i}\in B$, there is exactly one $\boldsymbol{v}\in\mathcal{V}(B)$.
- For each $\boldsymbol{v}\in\mathcal{V}(B)$ and for each d, there are exactly two faces $\boldsymbol{f}_{low}, \boldsymbol{f}_{high}\in\mathcal{F}_d$, with $\mathrm{high}(\boldsymbol{f}_{low}) = \boldsymbol{v}$, $\mathrm{low}(\boldsymbol{f}_{high}) = \boldsymbol{v}$, and $\alpha_{\boldsymbol{f}_{\{low,high\}}} = 1$.
- For each $\boldsymbol{v}\in\mathcal{V}(B)$, $\kappa_{\boldsymbol{v}} = 1$, and $\alpha_{\boldsymbol{v}}^{EB} = 0$.

This definition leads to a decomposition of $\mathcal{V} = \mathcal{R}\bigcup\mathcal{B}$, where $\mathcal{R}$ consists of all the regular control volumes, and $\mathcal{B} = \mathcal{V} - \mathcal{R}$ are the control volumes that intersect $\partial\Omega$. We also define the regular faces $\mathcal{R}^{\mathcal{F},d} = \{\boldsymbol{f}\in\mathcal{F}_d : \mathrm{low}(\boldsymbol{f}), \mathrm{high}(\boldsymbol{f})\in\mathcal{R}\}$

1.2 Rectangular Array Data and Operations on Embedded Boundary Data

Our fundamental EBArray objects are defined in terms of the rectangular subsets of the graph.

$$\begin{aligned} \phi &: \mathcal{V}(B) \to \mathcal{T}, \\ \phi &: \mathcal{F}_d(B) \to \mathcal{T}, \\ \phi &: \mathcal{B}(B) \to \mathcal{T}, \\ \mathcal{T} = \mathcal{T}(\mathbb{T}, C, D, E) &= \mathbb{T}^{C \times D \times E}. \end{aligned} \tag{2}$$

This is analogous to the Proto `BoxData`. However, here we have to distinguish between cell-centered arrays and face-centered arrays, since `Box`es of control volumes and of faces are topologically different. We also provide a specialized array for representing quantities defined on the boundary. We refer to the types of Boxes that are domains of the arrays, i.e., $\mathcal{V}$, $\mathcal{B}$, $\mathcal{F}_d$ as *domain types*, denoted generally as $\boldsymbol{d} \in \mathcal{D}$.

Because of the multiple different kinds of arrays, there are multiple different stencil operations corresponding to different array inputs and outputs.

$$S^{\mathcal{D}\mathcal{D}'}(\phi)_{\boldsymbol{d}} = \sum_{\boldsymbol{d}'} a_{\boldsymbol{d},\boldsymbol{d}'} \phi_{\boldsymbol{d}'} \ , \ \boldsymbol{d} \in \mathcal{D}(B) \ , \ B \equiv \mathrm{Box}(S),$$

where $\mathcal{D}\mathcal{D}'$ are any pair of $\mathcal{V}, \mathcal{F}_d$. Unlike in the `Proto` case, the largest Box over which the output array is defined, denoted above by B, is part of the definition of the stencil. This is because the stencil cannot be completely characterized without knowing where it might be applied relative to the boundary of the domain. However, a stencil can be applied to obtain an output defined on a smaller Box $B' \subset B$.

We also define Stencils that take as inputs or outputs arrays defined over rectangular subsets of $\mathcal{B}$.

$$\begin{aligned} S^{\mathcal{V}\mathcal{B}}(\phi)_{\boldsymbol{v}} &= \sum_{\boldsymbol{v}'} a_{\boldsymbol{v},\boldsymbol{v}'} \phi_{\boldsymbol{v}'} \ , \ \boldsymbol{v} \in \mathcal{B}(B), \\ S^{\mathcal{B}\mathcal{V}}(\phi)_{\boldsymbol{v}} &= \sum_{\boldsymbol{v}'} a_{\boldsymbol{v},\boldsymbol{v}'} \phi_{\boldsymbol{v}'} \ , \ \boldsymbol{v} \in \mathcal{V}(B). \end{aligned}$$

For any Stencil, we can specify a spanning Box $\mathrm{span}(S)$ to be the smallest Box such that

$$\mathrm{span}(S) \supset \{\mathrm{ind}(\boldsymbol{d}') - \mathrm{ind}(\boldsymbol{d}) : a_{\boldsymbol{d},\boldsymbol{d}'} \neq 0, \boldsymbol{d} \in \mathcal{D}(\mathrm{Box}(S)\}.$$

The low and high corners of the spanning Box of a Stencil can be used to define the Box over which the output array is defined given the input array and the Stencil:

$$\mathrm{Box}(S(\phi)) = \Big(\bigcap_{\boldsymbol{i} \in \mathrm{span}(S)} \mathrm{Box}(\phi) - \boldsymbol{i} \Big) \cap \mathrm{Box}(S).$$

Thus a Stencil is defined with any EBArray as an argument, although the output can be an array with an empty Box. We also assume that there is a set regular(S) such that, for any $\boldsymbol{d} \in$ regular(S), S is defined in terms of a `Proto` Stencil:

$$S(\phi)_{\boldsymbol{d}} = \sum_{\boldsymbol{s}\in\mathcal{S}} a_{\boldsymbol{s}} \phi_{\mathrm{ind}^{-1}(\mathrm{ind}(\boldsymbol{d})+\boldsymbol{s})}.$$

where $\mathcal{S} \subset \mathrm{span}(S^{\mathcal{V}})$ and the coefficients $\{a_{\boldsymbol{s}}\}_{\boldsymbol{s}\in\mathcal{S}}$ are independent of $\boldsymbol{d}$. A necessary condition for $\boldsymbol{d} \in$ regular(S) is that the smallest Box covering $\mathcal{S} + \mathrm{ind}(\boldsymbol{d})$ be regular; in particular, this insures that $\mathrm{ind}^{-1}(\mathrm{ind}(\boldsymbol{d}) + \boldsymbol{s})$ is well-defined for each $\boldsymbol{s} \in \mathcal{S}$.

We can also define a full algebra on Stencils. If S_1, S_2 are Stencils with the same domain types, and $\alpha \in \mathbb{T}$, then $\alpha S_1 + S_2 = S$ defines a Stencil with $\mathrm{Box}(S) = \mathrm{Box}(S_1) \cap \mathrm{Box}(S_2)$. Similarly, we can compose two stencils: $S^{\mathcal{D}\mathcal{D}'} = S = S_1 \circ S_2$, $S(\phi) = S_1(S_2(\phi))$, for $S_1 = S_1^{\mathcal{D}\mathcal{D}''}$, $S_2 = S_2^{\mathcal{D}''\mathcal{D}'}$, with

$$a_{\boldsymbol{d},\boldsymbol{d}'} = \sum_{\boldsymbol{d}''} a^1_{\boldsymbol{d},\boldsymbol{d}''} a^2_{\boldsymbol{d}'',\boldsymbol{d}'}$$

and

$$\mathrm{Box}(S) = \Big(\bigcap_{\boldsymbol{i}\in\mathrm{span}(S_1)} \mathrm{Box}(S_2) - \boldsymbol{i} \Big) \cap \mathrm{Box}(S_1).$$

In the present set of applications, a Stencil S is defined by defining regular(S) and a corresponding `Proto` Stencil in that region; then the Stencil coefficients at the remaining locations in Box(S) are computed using a polynomial interpolant whose coefficients are computed to satisfy a weighted least-squares fit to the input data. In principle, we could eliminate completely the regular Stencil definition, and use Stencils to define general variable-coefficient operators. However, we do not do this here. Stencils are designed to implement the extension to EB geometries of linear discretizations of constant-coefficient operators.

Constructing Stencils. Stencils representing discretizations of constant-coefficient partial differential operators are typically derived by constructing a local polynomial approximation from the input EBArray values, and then computing the approximation to the operator by applying the operator to that polynomial. This process can be expressed in terms of a stencil that is independent of the input array values. Typically, stencils are constructed by initially specifying the span of the stencil $\boldsymbol{S}$ and a collection of weights $(w_{\boldsymbol{d}'_0}, w_{\boldsymbol{d}'_1}, \dots) = \boldsymbol{w}$ that are a function of the local topology of $\mathcal{D}'(\boldsymbol{S} + \mathrm{ind}(\boldsymbol{d}))$, and of the centroids of the $\boldsymbol{d}'$s but otherwise independent of the geometry. Then $\boldsymbol{a} = (a_{\boldsymbol{d},\boldsymbol{d}'_0}, a_{\boldsymbol{d},\boldsymbol{d}'_1}, \dots)$ is computed using the Moore-Penrose pseudo-inverse

$$\boldsymbol{a} = \boldsymbol{T}(\boldsymbol{D}\boldsymbol{M})^{+}\boldsymbol{D}$$

where $\boldsymbol{D}$ is the diagonal matrix whose entries are the elements of $\boldsymbol{w}$, $\boldsymbol{M}$ is a matrix whose rows are P moments with each row of moments evaluated at a $\boldsymbol{d}'$, with the choice of moments

determined by the desired order of accuracy of the approximation; and $\boldsymbol{T}$ is the vector of moments corresponding to the value of the solution, of a derivative, or averages of the same, being evaluated at $\boldsymbol{d}$ by the stencil. With this approach, a regular stencil falls out naturally, since at the locations where a regular stencil is valid, the stencil constructed above is actually a `Proto` Stencil. However, we will provide an option of specifying a regular stencil independently. We expect that we will build up a library of such Stencil specifications (i.e. combinations of span, weights, and order of accuracy), that can be used to automatically construct stencils given the input geometry at run time as required by the user.

Pointwise Operations. Similarly to the case of `Proto`, a pointwise operation takes as input one or more EBArrays of the same domain type, evaluates a function on the values of those arrays on a rectangular subset of that domain type, and returns an EBArray containing those values.

$$\psi = F@(\phi^1, \phi^2, ...) \Leftrightarrow \psi_{\boldsymbol{d}} = F(\phi^1_{\boldsymbol{d}}, \phi^2_{\boldsymbol{d}}, ...),$$

where $\mathrm{Box}(\psi)$ is defined to be the intersection of the Boxes of the arguments. We can relax slightly the need for the domain types to be the same by allowing face-centered EBArrays to be aliased to cell-centered EBArrays by composing them with the high and low functions. If $\phi : \mathcal{V}(B) \to \mathcal{T}$, then

$$\begin{aligned}\mathrm{low}(\phi, d) =& \psi : \mathcal{F}^d(\mathrm{Box}(\phi)) \to \mathcal{T} \ , \ \psi_{\boldsymbol{f}} = \phi_{\mathrm{high}(\boldsymbol{f})},\\ \mathrm{high}(\phi, d) =& \psi : \mathcal{F}^d(\mathrm{Box}(\phi) - \boldsymbol{e}^d) \to \mathcal{T} \ , \ \psi_{\boldsymbol{f}} = \phi_{\mathrm{low}(\boldsymbol{f})}.\end{aligned}$$

Such aliases can appear as arguments of a pointwise operation without copying.

1.3 Nested Refinement

Given an embedded boundary discretization of space, we can define $\mathcal{C}_r(\mathcal{G}) = \{\mathcal{C}_r(\mathcal{V}), \mathcal{C}_r(\mathcal{F})\}$, the coarsening of that discretization by some integer factor $r \geq 1$. Each maximal connected component of the subgraph consisting of nodes $\mathrm{ind}^{-1}([\boldsymbol{i}r, (\boldsymbol{i} + \boldsymbol{u})r - \boldsymbol{u}])$ defines an element $\boldsymbol{v}_c \in \mathcal{C}_r(\mathcal{V})$, with $\mathrm{ind}(\boldsymbol{v}_c) = \boldsymbol{i}$. A face $\boldsymbol{f}_c = (\boldsymbol{v}_c, \boldsymbol{v}'_c)$ is defined to be the set $\boldsymbol{f}_c = \{\boldsymbol{f} : \mathrm{low}(\boldsymbol{f}) \in \boldsymbol{v}_c, \mathrm{high}(\boldsymbol{f}) \in \boldsymbol{v}'_c\}$, for all pairs $\boldsymbol{v}_c \neq \boldsymbol{v}'_c$ for which $\boldsymbol{f}_c$ is not empty. In that case, $\boldsymbol{f}_c \subset \mathcal{F}_d$ for some d, and $\boldsymbol{f}_c \in \mathcal{C}_r(\mathcal{F})_d$.

The geometric quantities such as volumes, areas, and centroids, are obtained by averaging, which is exact in the sense of truncation error.

$$\begin{aligned}\kappa_{\boldsymbol{v}_c} =& \frac{1}{r^{\boldsymbol{D}}} \sum_{\boldsymbol{v} \in \boldsymbol{v}_c} \kappa_{\boldsymbol{v}} \ , & \boldsymbol{x}_{\boldsymbol{v}_c} =& \frac{1}{\kappa_{\boldsymbol{v}_c}} \sum_{\boldsymbol{v} \in \boldsymbol{v}_c} \kappa_{\boldsymbol{v}} \boldsymbol{x}_{\boldsymbol{v}},\\ \alpha_{\boldsymbol{f}_c} =& \frac{1}{r^{\boldsymbol{D}-1}} \sum_{\boldsymbol{f} \in \boldsymbol{f}_c} \alpha_{\boldsymbol{f}} \ , & \boldsymbol{x}_{\boldsymbol{f}_c} =& \frac{1}{\alpha_{\boldsymbol{f}_c}} \sum_{\boldsymbol{f} \in \boldsymbol{f}_c} \alpha_{\boldsymbol{f}} \boldsymbol{x}_{\boldsymbol{f}},\\ \alpha^{EB}_{\boldsymbol{v}_c} =& \frac{1}{r^{\boldsymbol{D}-1}} \sum_{\boldsymbol{v} \in \boldsymbol{v}_c} \alpha^{EB}_{\boldsymbol{v}} \ , & \boldsymbol{x}^{EB}_{\boldsymbol{v}_c} =& \frac{1}{\alpha^{EB}_{\boldsymbol{v}_c}} \sum_{\boldsymbol{v} \in \boldsymbol{v}_c} \alpha^{EB}_{\boldsymbol{v}} \boldsymbol{x}^{EB}_{\boldsymbol{v}}.\end{aligned}$$

We extend the definition of Stencils to allow for Stencils that act between EBArrays defined over different levels of refinement, analogous to the functionality provided using strided stencils in `Proto`.

$$S^{\mathcal{D}\mathcal{C}_r(\mathcal{D})}(\phi)_{\boldsymbol{d}} = \sum_{\boldsymbol{d}'_c} a_{\boldsymbol{d},\boldsymbol{d}_c}\phi_{\boldsymbol{d}_c} \ , \ \boldsymbol{d} \in \mathcal{D}(\mathrm{Box}(S)),$$

$$S^{\mathcal{C}_r(\mathcal{D})\mathcal{D}}(\phi)_{\boldsymbol{d}_c} = \sum_{\boldsymbol{d}} a_{\boldsymbol{d}_c,\boldsymbol{d}}\phi_{\boldsymbol{d}} \ , \ \boldsymbol{d}_c \in \mathcal{C}_r(\mathcal{D})(\mathrm{Box}(S)).$$

2 EBProto

As is the case in `Proto`, discretizations of partial differential equations will be represented as composition of Stencil operations and pointwise operations on EBArrays. In this section, we will describe the C++ classes that implement these data types and operations. These classes assume the existence of `Proto`'s `Point` and `Box` classes for representing rectangular regions on a Cartesian grid. Another difference between `Proto` and `EBProto` is that the underlying spatial independent variables, i.e. the EB graph, are a computed object, as opposed to $\mathbb{Z}^{\boldsymbol{D}}$, which is implicitly defined. Consequently, the definition of the geometric information with use distributed-memory objects from `Chombo`, although the definition of EBArrays and stencil operations will be on a `Box`-by-`Box` basis.

2.1 GeometryService

`GeometryService<Order>` creates geometry from an implicit function representation of the boundary. The template parameter `Order` is the integer order of accuracy of the representation (at least 2).

- ```
  GeometryService(const shared_ptr<BaseIF>& a_baseIF,
                  const RealVect& a_origin,
                  const double& a_dx,
                  const Box& a_domain,
                  const DisjointBoxLayout& a_finestGrids,
                  const Point& a_ghost,
                  int a_maxCoarsen)
  ```

  builds the EB geometric description on a hierarchy of rectangular domains, specified by the finest domain `a_domain` with mesh spacing `a_dx`, down to a coarsened level specified by `a_maxCoarsen`. The finest level is decomposed into a disjoint union of rectangles distributed over processors given by a `Chombo DisjointBoxLayout`. The implicit function is specified in terms of a pointer to a class derived from `BaseIF`.

- `EBGraph` contains the geometry description on a `Box`, used for construction of `EBBoxData`. For a given level, these are accessed through the `GeometryService` member function
  ```

```
EBGraph getGraph(const Box &a_domain, int a_boxid)
```

2.2 EBBoxData

An `EBBoxData<CENTERING, T, C>` represents a EBArray of the form (**??**) defined over a `Box`. This is the embedded boundary extension of the Proto `BoxData` class. Unlike `Proto BoxData`, `EBBoxData` include centering in their type definition via the enumeration type `enum CENTERING {CELL=0,BOUNDARY,XFACE,YFACE,ZFACE}` corresponding to EBArrays defined over $\mathcal{V}(B)$, $\mathcal{B}(B)$, $\mathcal{F}_d(B)$, $d = 0, \ldots \boldsymbol{D} - 1$, respectively. `T` is the type of the value taken on by the array, and `C` is an unsigned integer specifying the number of components (we do not yet support higher-rank tensors). The constructor

```
EBBoxData(const Box& a_box, const EBGraph& a_graph)
```

defines an `EBBoxData` over a `Box a_box`, using the geometric information in `a_graph`.

Pointwise Operations and `forall`

We currently have a single `forall` capability implemented in `EBProto`, that implements operations of the form (**??**). For example, if we want to compute

$$A' = F@(A) \text{ on } \boldsymbol{B}, A : \mathcal{V}(\boldsymbol{B}_A) \rightarrow \mathcal{T} \ , \ F; \mathcal{T} \rightarrow \mathcal{T}'$$

to compute

$$A' : \boldsymbol{B}_{A'} \rightarrow \mathcal{T}' \tag{3}$$

on $\mathcal{V}(\boldsymbol{B}_A \cap \boldsymbol{B}_{A'} \cap \boldsymbol{B})$, we define the function

```
void(F(Var<Tprime,CPrime>& AprimePoint, Var<T,C> APoint)
{
AprimePoint(0) = APoint(1);
AprimePoint(1) = APoint(2) * APoint(3);
...
}
```

Then this is applied to compute (**??**) as follows.

```
Box B = ...;
EBBoxData<CENTERING::CELL,Real,C > A = ...;
EBBoxData<CENTERING::CELL,Real,Cprime > A(B_Aprime);
forallInPlace(F,B,APrime,A);
```

More generally, the signature of a `forallInPlace` is given as follows.

```
void ebforallInPlace(const& Func a_F,
                     Box& a_bx,
                     Srcs& ... a_src)
```

where F is the function that is to be applied pointwise, a_bx is the box over which the pointwise application is to be restricted, and Srcs& ... is a variadic argument list that consists either of scalar arguments, or of EBBoxDatas with the same centering. The matching signature for F is

```
void F(Type1& a_src1,
       Type2& a_src2, ...
       TypeN& a_srcN)
```

where Typej is a type corresponding to the j^{th} argument in the foralInPlace Src& ... list. If the j^{th} entry in the list is a scalar type Tj, then Typej=Tj. If the j^{th} entry in the list is of type EBBoxData<cen,Tj,Cj>, then Typej = Var<Tj,Cj>. Any of the non-const EBBoxData arguments can be updated in place using this version of forall.

2.3 EBStencil, EBDictionary

An EBStencil<int,T,CENTERING,CENTERING> is an object used to define and implement operations of the form (??). Mostly, users with use EBStencils whose rules for construction given a GeometryService have been archived. What we will describe he is the process by which an application accesses EBStencils, and applies them to EBBoxDatas. This is done via an EBDictionary<order,T,srcCenter,destCenter>, where the template parameter order is the integer order of accuracy of the class of stencils (for the current family of algorithms, order = 2; T is the type of the stencil coefficients; and srcCenter, destCenter are the centerings of the source and destination of the stencil operation.

```
EBDictionary(const shared_ptr<GeometryService<order> >& a_geoserv,
             const DisjointBoxLayout& a_grids,
             const Box& a_domain,
             T a_dxPoint,
             Point a_srcGhost,
             Point a_dstGhost)
```

is the constructor for the dictionary. To use a stencil from the archive, one must first register a stencil, using

```
void registerStencil(string a_stencilName,
                     string a_domainBCName,
                     string a_ebbcName)
```

where a_stencilName is the name of the stencil in the archive, and a_domainBCName is the name of the domain boundary condition, and a_ebbcname is the name of the boundary condition on the embedded boundary. To access an EBStencil one uses the EBDictionary member function

```
shared_ptr<EBStencil<order,T,srcCenter,destCenter> >
getEBStencil(const string & a_stencilName,
             const string & a_ebbcName,
             const int    & a_boxid)
```

where a_stencilName is the name of the stencil, a_ebbcName is the name of the EB boundary condition, and a_boxid is the index into the DisjointBoxLayout over which the problem is defined.

We apply EBStencils in a similar fashion to Stencils in Proto.

```
GeometryService<2> gs;
auto sten = getEBStencil("Divergence","DivergenceBC",firstBox);
auto graph = gs.getGraph(thisDomain,firstBox);
EBBoxData<CENTERING::XFACE,Real,1> xflux(B,graph);
xflux = ...;
auto divF = sten(xflux,1.0);
```

List of Archived EB Stencils

These are defined in the objects of type EBStencilArchive<srcCenter,destCenter,order,T>.

- "Second_Order_Poisson"
- "AverageCellToFace"
- "Cell_To_Face_Low"
- "Cell_To_Face_High"
- "Divergence"
- "InterpolateToFaceCentroid"
- "Volume_Weighted_Averaging_rad_n"
- "Restriction"
- "Multigrid"
- "PWC_Prolongation_n"

- "Volume_Weighted_Redistribution_rad_n"
- "Slope_Low_d"
- "Slope_High_d"