A generic grid interface for parallel and adaptive scientific computing.
Part I: abstract framework

P. Bastian[1] M. Blatt[1] A. Dedner[2] C. Engwer[1] R. Klöfkorn[2]
M. Ohlberger[3] O. Sander[4]
[1]Institut für Parallele und Verteilte Systeme, Universität Stuttgart, Germany
[2]Abteilung für Angewandte Mathematik, Universität Freiburg, Germany
[3]Institut für Numerische und Angewandte Mathematik, Universität Münster, Germany
[4]Institut für Mathematik, Freie Universität Berlin,
DFG Research Center Matheon, Berlin, Germany

Abstract

We give a mathematically rigorous definition of a grid for algorithms solving partial differential equations. Unlike previous approaches [2, 3], our grids have a hierarchical structure. This makes them suitable for geometric multigrid algorithms and hierarchical local grid refinement. The description is also general enough to include geometrically nonconforming grids. The definitions in this article serve as the basis for an implementation of an abstract grid interface as C++ classes in the DUNE framework [1].

AMS Subject Classifications: 65N30, 52C99, 65Y05
Key words: DUNE, hierarchical grids, interface, finite elements, finite volumes, entity complex, geometric realization, father relation, index maps, parallelization

1 Introduction

The vast majority of methods used for the solution of partial differential equations (PDEs) are based on some form of a grid. This has lead to the development of a large number of numerical analysis codes, each containing a data structure representing such a computational grid. However, the requirements from such data structures vary widely from application to application. The typical trade-off is efficiency vs. flexibility. Hence each code will be more suitable in some areas of applications, and less so in others.

To overcome the limitations of existing grid-based PDE solvers we present the "Distributed and Unified Numerics Environment" DUNE (see [5] for the current version of the software). DUNE has a component-based software architecture where each component possesses an abstract interface and exists in several implementations with different features. This concept is realized very efficiently using generic programming techniques in C++, which essentially removes the interface overhead at compile-time. Moreover, existing software can be used to implement the components.

Several authors have tried to formalize the notion of a grid mathematically [2, 3, 4], a task which has proven to be surprisingly difficult. A concept that has been missing so far is the notion of a hierarchical grid. It is nevertheless of prime importance today as it forms the basis of geometric multigrid methods and many locally adaptive algorithms.

In this article we present the mathematical description of hierarchical grids which underlies and motivates the design of the DUNE grid interface. Based on this mathematical description a companion paper [1] describes the interface of DUNE's grid component in the form of C++ classes and illustrates its functionality and efficiency with some examples. When designing the classes for the grid interface we felt that a mathematically rigorous description of the entities comprising a grid and their complicated relations is required. We attempt to formally describe grids with the following features:

- Elements of various shapes and dimensions

- Grids embedded in higher-dimensional spaces (e. g., grids on manifolds)
- Hierarchical local grid refinement
- Nonconformity in and between levels of refinement
- Separation of grids and data
- Overlapping and nonoverlapping decompositions for parallel processing

An implementation of the grid interface may cover only a subset of these features.

The first formal definition of a grid for numerical computations was given by Berti [3]. He noted that the productivity of people working in numerical analysis could be increased greatly by the employment of reusable components. In particular this meant separating the algorithm from the implementation of the grid by specifying a set of *kernel concepts* of a grid that algorithms should rely on exclusively. Berti used generic programming extensively to reduce the interface overhead and also treated the problem of data-parallel grid computations.

With applications in numerical relativity in mind, Benger [2] generalized the definition of a grid such as to be able to handle grids of four-dimensional curved space-time. Inspired by differential geometry, he introduced the *fiber bundle data model*. His grids include sets of local coordinate charts which allow the description of curved spaces without a surrounding Euclidean space. Here as well generic programming is used extensively.

Botta et al. [4] took abstraction into a different direction. They introduced the notion of *relation-based computations*. This very general concept subsumes many different grid operations but also algorithms from linear algebra such as matrix-vector multiplication. They propose a thin low-level software layer intended to structure algorithm-oriented parallel computations on this very general concept.

All three approaches differ from DUNE in that they do not contain the concept of a *grid hierarchy*, which is essential for many efficient algorithms such as adaptive local grid refinement or geometric multigrid solvers. Also, neither of the implementations was designed with the idea in mind that it should be possible to reuse legacy code and switch between grid implementations quickly. These, however, are central ideas of DUNE.

In our view, a hierarchical grid consists of three key concepts. An *entity complex* describes the topological information (connectivity) of a non-hierarchical grid (level grid). Such a complex is embedded into a Euclidean space by means of a *geometric realization*. Several level grids are connected in a hierarchy by a *father relation*.

In Sec. 2 we formalize these definitions. Sec. 3 will then show how the set of leafs in the hierarchical structure can be seen as a non-hierarchical grid in its own right. Sec. 4 introduces *intersections* as another relation between elements. Sec. 5 presents the necessary concepts for parallel computation on a general grid. Finally, in Sec. 6 we show how data (e. g., degrees of freedom in the finite element context) can be associated with entities of a grid in a general and efficient way.

2 Hierarchical grids

This section gives a precise meaning to the idea of a grid hierarchy. We separate the topological from the geometrical aspects of the grid and define the father relation.

We begin by giving a few preliminary concepts. A more in-depth treatment can be found, e. g., in [3].

Definition 1 (Convex polytope) *A subset $\theta \subset \mathbb{R}^w$, $w \geq 0$, is a convex polytope if it is the convex hull of a nonempty finite set of points $X = \{x_0, \ldots, x_n\}$. The dimension $\dim \theta$ of the convex polytope is the maximal number of linearly independent vectors in the set $\{x_1 - x_0, \ldots, x_n - x_0\}$ if $n \geq 1$ and 0 if $n = 0$. Obviously, $0 \leq \dim \theta \leq w$. Given a nonempty, finite set of points X we denote by $\theta(X)$ the convex polytope generated by X.*

Definition 2 (Faces of a polytope) *If, for fixed $c \in \mathbb{R}^w$, $\gamma \in \mathbb{R}$, the inequality $c^T x \leq \gamma$ is valid for all $x \in \theta$, then $f = \theta \cap \{x \in \mathbb{R}^w \mid c^T x = \gamma\}$ is a face of θ. With this definition, $\emptyset$ is an (improper) face, and we also add θ as an additional improper face. All other faces are called* proper. *The dimension of a face is the*

dimension of its convex hull; by convention, the dimension of $\emptyset$ is set to -1. We write $f \preceq \theta$ if f is a face of θ.

Let θ be a convex polytope and F the set of its faces. The is-face-of relation $\preceq$ on F is antisymmetric and transitive. Hence $(F, \preceq)$ has the structure of a partially ordered set (poset).

Definition 3 (Combinatorial isomorphism) *Let X and Y be sets with binary relations S and R, respectively. A (combinatorial) isomorphism is a bijective mapping $\varphi : X \to Y$ such that*

$$\varphi(u)\, R\, \varphi(v) \quad \text{if and only if} \quad u\, S\, v.$$

If such a mapping exists between two sets X and Y then these sets are called (combinatorially) isomorphic.

Intuitively, two sets are isomorphic if they exhibit the same combinatorial structure. We use this to define an equivalence relation $\simeq$ on the set of all convex polytopes by saying that two convex polytopes θ and θ' are isomorphic, $\theta \simeq \theta'$, iff there exists a mapping φ from the set of faces of θ to the set of faces of θ' such that $\varphi(u)$ is a face of $\varphi(v)$ if and only if u is a face of v.

Dividing the set of all convex polytopes by the equivalence relation $\simeq$ we obtain a set of 'prototype' polytopes which the numerical analyst calls *reference elements*.

Definition 4 (Reference element) *A d-dimensional reference element is an equivalence class of d-dimensional convex polytopes in d-dimensional Euclidean space with respect to combinatorial isomorphism $\simeq$. From each equivalence class we pick one representative. For any $m \in \mathbb{N}_0$ the set of representatives of m-dimensional reference elements is called $\mathcal{R}_m$, and we also define $\mathcal{R} = \bigcup_m \mathcal{R}_m$. In an abuse of notation we again speak of reference elements.*

Remark 1 *Reference elements used in numerical algorithms are, e.g., the unit square, the unit triangle, and the unit hexahedron.*

Based on these definitions we can define the codimension of a convex polytope relative to another convex polytope.

Definition 5 (Codimension) *Let $\theta, \sigma \subset \mathbb{R}^w$ be two convex polytopes with σ a face of θ. Then* $\operatorname{codim}(\sigma, \theta) = \dim\theta - \dim\sigma$ *is called the codimension of σ with respect to θ. Obviously, we have* $0 \leq \operatorname{codim}(\sigma, \theta) \leq \dim\theta + 1$.

The next definition introduces a central part of a hierarchical grid. An *entity complex* contains all topological information of a grid on a specific refinement level. The concept is a generalization of the simplicial complexes known from combinatorial geometry, the main difference being that it allows for non-simplex elements.

Definition 6 (General entity complex) *A general d-dimensional entity complex E is a system of nested subsets*

$$E = \{E^d, E^{d-1}, \ldots, E^0\}$$

such that E^d is a finite set (called set of vertices*) and for $0 \leq c < d$*

$$E^c \subseteq \mathcal{P}(E^{c+1}),$$

where $\mathcal{P}(E^{c+1})$ denotes the power set of E^{c+1}. Each element of E^c is called an entity of codimension c and of dimension $d - c$. We require that

$$\bigcup_{e \in E^c} e = E^{c+1},$$

which means that every codimension $c+1$ entity is contained in at least one codimension c entity. The union of all entities in a complex is called

$$\bar{E} = \bigcup_{0 \leq c \leq d} E^c.$$

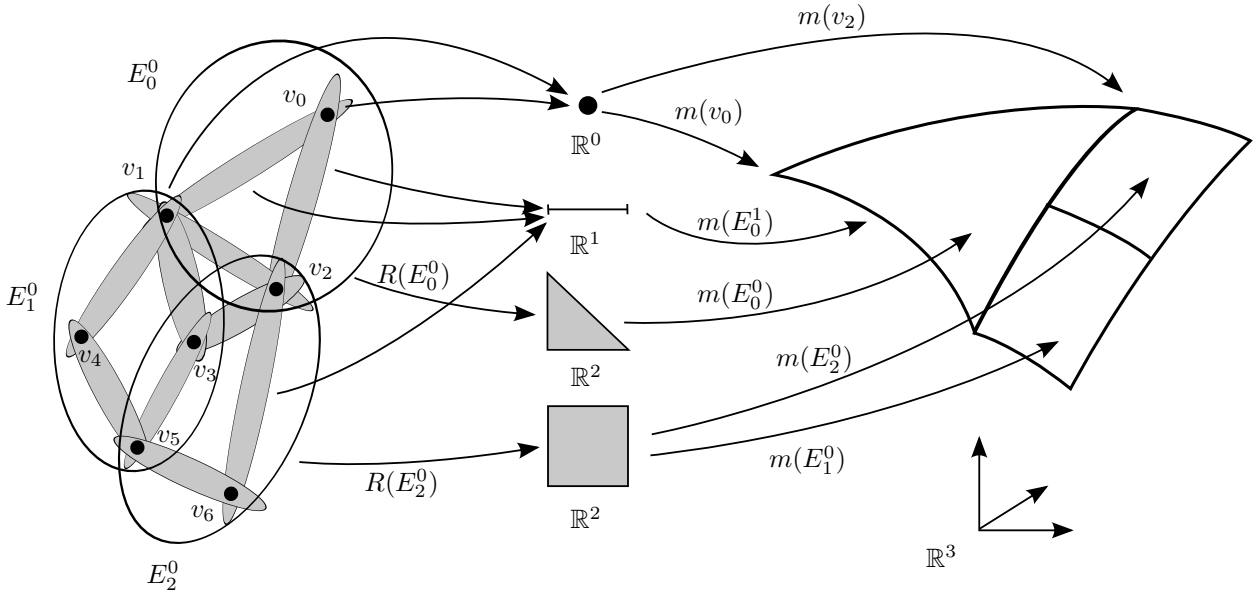

Figure 1: A 2-dimensional entity complex (left), the reference elements (center), and a geometric realization in $\mathbb{R}^3$ (right).

Hence, a codimension c entity is a set of codimension $c+1$ entities. The codimension 0 entities will sometimes be called *elements*.

To motivate this abstract definition consider the example in Fig. 1. On the left there is a two-dimensional entity complex containing seven vertices

$$E^2 = \{v_0, v_1, \ldots, v_6\}$$

represented by small black balls. The grey ellipses denote of entities of codimension 1 or 'edges', which are

$$\begin{aligned} E^1 &= \{E_0^1, E_1^1, \ldots, E_9^1\} \\ &= \big\{\{v_0, v_1\}, \{v_1, v_2\}, \{v_0, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \\ &\qquad \{v_1, v_4\}, \{v_3, v_5\}, \{v_2, v_6\}, \{v_4, v_5\}, \{v_5, v_6\}\big\}. \end{aligned}$$

Finally, the sets of edges encircled in black are the three elements, or entities of codimension 0

$$\begin{aligned} E^0 &= \{E_0^0, E_1^0, E_2^0\} \\ &= \Big\{\big\{\{v_0, v_1\}, \{v_1, v_2\}, \{v_0, v_2\}\big\}, \\ &\qquad \big\{\{v_1, v_3\}, \{v_3, v_5\}, \{v_4, v_5\}, \{v_1, v_4\}\big\}, \\ &\qquad \big\{\{v_2, v_3\}, \{v_2, v_6\}, \{v_5, v_6\}, \{v_3, v_5\}\big\}\Big\}. \end{aligned}$$

A d-dimensional general entity complex can contain entities which do not have a natural geometric realization as a convex polytope. For example, it would be possible to add the 'diagonal edge' $\{v_3, v_6\}$ to the 'quadrilateral' E_2^0. We therefore add on further restriction. Before we can state it we have to introduce the subentity relation. It is the natural equivalent of the is-face-of relation for entity complexes.

Definition 7 (Subentity relation) *Let E be a d-dimensional general entity complex. By $\mathcal{S} \subset \bar{E} \times \bar{E}$ we denote the reflexive and transitive closure of the inclusion relation of entities, i.e., we have $e \,\mathcal{S}\, e'$ if and only if $e = e'$ or there exists a sequence of entities $\{e_0, e_1, \ldots, e_k\}$ with*

$$e = e_0, \quad e_k = e', \quad e_i \in e_{i+1} \quad \forall 0 \le i < k.$$

By $s(e) = \{e' \in \bar{E} \,|\, e' \,\mathcal{S}\, e\}$ we denote the set of all subentities of a given entity $e \in \bar{E}$.

Now the last requirement for an entity complex states that for each entity there must be a corresponding reference element. In other words, we allow only those entities for which a convex polytope with identical combinatorial structure exists.

Definition 8 (Entity complex) *Let E be a general entity complex. E is called* entity complex *if there is a mapping $R : \bar{E} \to \mathcal{R}$ such that for each $e \in \bar{E}$ the entity e is combinatorially isomorphic to $R(e)$ in the sense that there exists a bijective mapping φ from the set $s(e)$ of subentities of e to the set f of faces of $R(e)$ such that for all $s_1, s_2 \in s(e)$ we have*

$$s_1 \mathcal{S} s_2 \quad \text{if and only if} \quad \varphi(s_1) \preceq \varphi(s_2).$$

From the set of isomorphisms from e to $R(e)$ we pick one and call it φ_e.

Note that this definition is consistent in the sense that

$$R(s) \simeq \varphi_e(s)$$

holds for all subentities s of an entity e.

The entity complex is a strictly topological object. We now give it a geometry by embedding it into a Euclidean space $\mathbb{R}^w, w \geq d$. We call this space the *world space* and w the *world dimension*. Before, we need the notion of *relative interior*.

Definition 9 (Relative interior) *Let $d, w \in \mathbb{N}$, $d \leq w$, and let $S \subset \mathbb{R}^w$ be such that there is a homeomorphism $f : \bar{S} \to \bar{B}_1$, where $\bar{S}$ denotes the closure of S and B_1 is the open unit ball in $\mathbb{R}^d$. Then we call*

$$\operatorname{int} S = f^{-1}(B_1)$$

the relative interior of S. Accordingly, we define the relative boundary

$$\partial S = \bar{S} \setminus \operatorname{int} S.$$

To see how this differs from the usual notion of interior in $\mathbb{R}^w$ consider a line segment l in $\mathbb{R}^2$. The interior $\mathring{l}$ is the set of all points $x \in l$ such that there exists a two-dimensional ball $B_x(\epsilon)$ with positive radius ϵ centered at x and which is entirely contained in l. Hence $\mathring{l}$ is empty. The relative interior $\operatorname{int} l$ on the other hand contains all of l except for the two endpoints. In particular, note that for any $x \in \mathbb{R}^w$ we have $\operatorname{int}\{x\} = \{x\}$.

Definition 10 (Geometric realization) *Let $w \in \mathbb{N}$, $w \geq d$. A geometric realization of an entity complex E is a family of homeomorphisms m such that for each $e \in \bar{E}$ there is an $m_e \in m$ with $m_e : R(e) \to \mathbb{R}^w$ such that the following conditions hold. Denote by $\theta_e = m_e(R(e)) \subset \mathbb{R}^w$ the geometric object associated with e.*

1. *For $e, e' \in E^0$, $e \neq e'$ we have*
$$\operatorname{int} \theta_e \cap \operatorname{int} \theta_{e'} = \emptyset.$$
2. *For all $e \in \bar{E}$, the set of all θ_s with $s \in e$ covers the boundary of θ_e*
$$\bigcup_{s \in e} \theta_s = \partial \theta_e.$$

The geometric realization provides an entity complex with a geometric shape. The interplay between entity complex, geometric realization, and the reference elements is illustrated in Fig. 1. Note that the triangle, as a point set, does not have to be connected to the two quadrilaterals except at the two common vertices.

Next we combine two level grids to a hierarchical structure. For this we first assign a father element to each element on the higher level and then extend this mapping to the entities of lower dimensions, where possible.

Definition 11 (Element father mapping) *Let E and E' be two entity complexes of equal dimension with corresponding geometric realizations m and m', respectively. An element father mapping is a mapping $F^0 : E'^{,0} \to E^0$ such that*

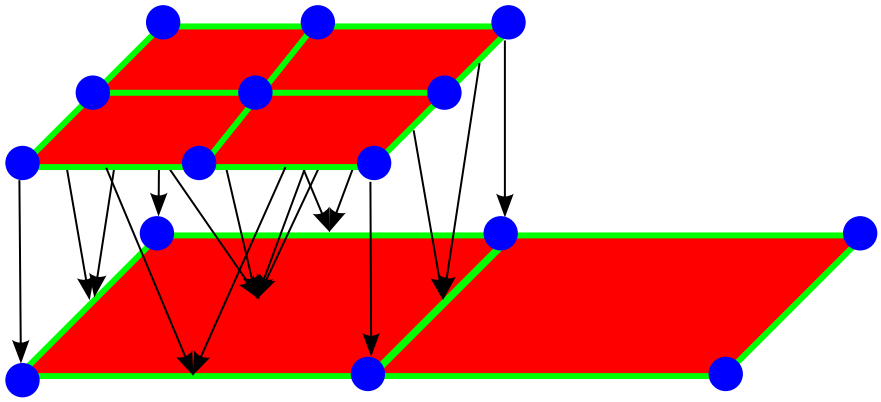

Figure 2: Father relation between two level grids.

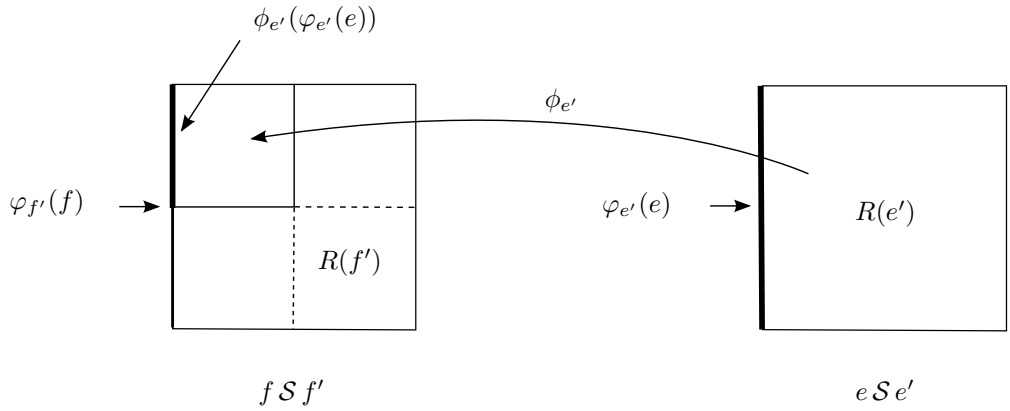

Figure 3: Father relation for positive codimension: f is the father of e because $\phi_{e'}(\varphi_{e'}(e))$ is a subset of $\varphi_{f'}(f)$.

1. *the children* $c(e) = \{e' \in E'^{,0} \mid e = F^0(e')\}$ *of* e *form a logical refinement of* e*, that is there are smooth injective mappings* $\phi_{e'} : R(e') \to R(F^0(e'))$ *such that the images of the* $\phi_{e'}$ *partition* $R(F^0(e'))$*,*
2. *if* $e_0, e_1 \in E'^{,0}$ *such that* $F^0(e_0) \neq F^0(e_1)$ *and* $\theta_{e_0} \cap \theta_{e_1}$ *has positive* $d-1$*-dimensional measure then* $\theta_{F^0(e_0)} \cap \theta_{F^0(e_1)}$ *has positive* $d-1$*-dimensional measure, which means that adjacent elements have adjacent fathers.*

The element father mapping is extended to a relation between entities of all codimensions. The extended structure is a relation rather than a mapping because not every entity of positive codimension has a father.

Definition 12 (Father relation) *Set* $F \subseteq \bar{E} \times \bar{E}'$ *with* $f\, F\, e$ *if and only if* $\dim e = \dim f$ *and one of the following conditions holds:*

1. $f = F^0(e)$
2. *Let* $f' \in E^0$*,* $e' \in E^{0\prime}$ *such that* $f' = F^0(e')$ *and* $f \mathcal{S} f'$ *and* $e \mathcal{S} e'$*. Then*

$$\phi_{e'}(\varphi_{e'}(e)) \subseteq \varphi_{f'}(f)$$

(see Fig. 3), with φ *and* ϕ *from Def. 8 and Def. 11.*

We now have all the ingredients to introduce hierarchical grids, which are the centerpiece of this section.

Definition 13 (Hierarchical grid) *A d-dimensional grid in a w-dimensional world is a triple $(\mathcal{E}, \mathcal{M}, \mathcal{F})$ consisting of a finite set of d-dimensional entity complexes*

$$\mathcal{E} = \{E_0, \ldots, E_k\},$$

a set of geometric realizations

$$\mathcal{M} = \{m_0, \ldots, m_k\},$$

where each m_i is a geometric realization of E_i into $\mathbb{R}^w$, and with a set of father relations

$$\mathcal{F} = \{F_0, \ldots, F_{k-1}\},$$

such that F_i connects E_i with E_{i+1} for all $0 \leq i < k$.

We will use $\mathcal{E}^c = \bigcup_{i=0}^{k} E_i^c$ for the union of all entities of codimension c in a hierarchical grid. In an abuse of notation we will also sometimes write $\mathcal{E} = \bigcup_c \mathcal{E}^c$ for the union of all entities. Finally we write

$$\mathcal{F} = \bigcup_i F_i \subset \mathcal{E} \times \mathcal{E}$$

for the set of all father relations in a hierarchical grid.

We call each pair (E_i, m_i) a *level grid*. It is natural to assign a level number to each entity in the hierarchical grid.

Definition 14 (Level of an entity) *Let $\mathcal{E} = \{E_0, \ldots, E_k\}$ be the entity complexes of a grid. The function*

$$\begin{aligned} l &: \bigcup_i \bar{E}_i \to \mathbb{N}_0 \\ l(e) &= i \quad \text{if and only if } e \in E_i \end{aligned}$$

is called the entity level function.

3 Leaf grids

This section introduces the concept of a leaf grid. While each pair (E_i, m_i) constitutes a non-hierarchical grid suitable for, say, finite element computations, so does the set of all leaf elements together with an entity complex structure it induces and its corresponding geometric realization. In fact, it is the natural grid to choose for non-hierarchical numerical methods on locally adaptive grids. We begin by introducing a new relation, this time on all entities of a hierarchical grid.

Definition 15 (Copies) *If an entity has exactly one child we call this child a copy of its father. We define the reflexive and symmetric binary relation $C \subseteq \mathcal{E} \times \mathcal{E}$ by*

$$e \, C \, e' \quad \Leftrightarrow \quad e = e' \text{ or } c(e) = \{e'\} \text{ or } c(e') = \{e\}.$$

The transitive closure $\sim$ of C is an equivalence relation.

Note that the set of elements on the lowest level together with the father mapping forms a forest. Next we define the leaf entities, which are the elements without children and their subentities.

Definition 16 (Leaf entities) *The set of codimension c leaf entities $L^c \subset E^c$ is defined for $0 \leq c \leq d$ by*

1. *$e \in L^0$ if and only if $e \in E^0$ and $c(e) = \emptyset$.*
2. *$e \in L^c$ if and only if there exists an $e' \in L^0$ and $e \, \mathcal{S} \, e'$.*

Further we set $L = \bigcup_{0 \leq c \leq d} L^c$ the set of all leaf entities.

The sets L^c contain copies. Factoring those out we obtain the leaf entity complex.

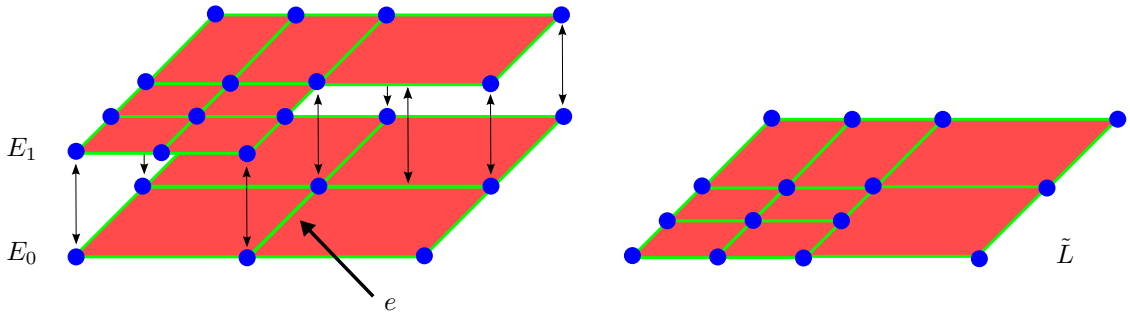

Figure 4: Copy relation between two entity complexes E_0 and E_1 (left), and corresponding leaf entity complex (right). Note that the edge e in E_0 is contained in $\tilde{L}$.

Definition 17 (Leaf entity complex) *By $[e] = \{e' \in L \,|\, e' \sim e\}$ we denote the equivalence class of e in L and $\tilde{L} = L/\sim = \{[e] \,|\, e \in L\}$ denotes the quotient set. The subentity relation $\mathcal{S}$ can be defined naturally on the quotient set as $\tilde{\mathcal{S}} \subseteq \tilde{L} \times \tilde{L}$ with $[e]\,\tilde{\mathcal{S}}\,[e']$ if and only if there exist class representatives $e \in [e]$ and $e' \in [e']$ with $e\,\mathcal{S}\,e'$.*

For a complete leaf grid we need a geometric realization which corresponds to the leaf entity complex. This geometric realization can be inherited from the level grid realizations.

Definition 18 (Leaf geometric realization) *Assume that the geometric realizations of the E_i are such that $\theta_e = \theta_{e'}$ whenever $e \sim e'$. Then the geometric realizations of the E_i induce a geometric realization $\tilde{m}$ of the leaf entity complex $\tilde{L}$ by setting $\tilde{m}_{[e]} = m_e$ for all $[e] \in \tilde{L}$.*

In the following we will assume the assumption stated in this definition to hold. The construction of the leaf grid is illustrated with an example in Fig. 4.

4 Intersections

Many numerical methods require information about intersections of neighboring elements. For example finite volume methods need to evaluate integrals over such intersections. As grids may be nonconforming an intersection does not necessarily correspond to a subentity.

Definition 19 (Level intersections) *Let (E, m) be an entity complex with a geometric realization, and denote by $\Theta = \bigcup_{e \in E^0} \theta_e$ the domain it covers. For any $e \in E^0$ we define its set $\mathcal{I}_e$ of intersections as a set of subsets of $\mathbb{R}^w$,*

$$\mathcal{I}_e = \{I_e^0, \ldots, I_e^k\}.$$

Each set I_e^i must have positive $d-1$-dimensional measure, and there must either exist an $e' \in E^0$ such that $e' \neq e$ and

$$I_e^i = \theta_e \cap \theta_{e'}$$

(intersection with another element), or there is a $d-1$-dimensional subentity $s \in e$ such that

$$I_e^i = \theta_s \cap \partial\Theta$$

(intersection with the grid boundary).

Let $(\tilde{L}, \tilde{m})$ be the leaf entity complex of a hierarchical grid with its induced geometric realization $\tilde{m}$. In complete analogy to Def. 19 we also define the set of *leaf intersections* $\tilde{\mathcal{I}}_{[e]}$ for each $[e] \in \tilde{L}^0$.

Remark 2 *Many numerical algorithms need to integrate over intersections of elements. This may be difficult as intersections in nonconforming grids may have shapes for which quadrature rules are not available. The DUNE system therefore allows grid implementations to return triangulations or similar partitions of a single intersection.*

5 Parallelization

This section introduces grid features needed for parallel computations. We assume that parallel computations on a hierarchical grid follow the 'single program multiple data' (SPMD) programming paradigm based on a suitable decomposition of the grid entities.

The decomposition is carried out in a two-step process. First, elements are assigned to processes (master decomposition). In a second step the decomposition for the remaining entities is determined from this master decomposition. We assume that $n \geq 1$ processes are available for the parallel computation and that each process is identified by a number $p \in P = \{0, \ldots, n-1\}$.

The master decomposition is defined as follows:

Definition 20 (Master decomposition) *The mapping of entities to processes is described by the master decomposition relation*

$$\mathcal{D}^0 \subseteq \mathcal{E}^0 \times P.$$

If $e\,\mathcal{D}^0\,p$ then entity e is known to process p.

The elements of a process are assigned to different classes.

Definition 21 (Partition type) *The map*

$$t^0 : \mathcal{D}^0 \to \{\mathtt{i}, \mathtt{o}, \mathtt{g}\}$$

assigns a partition type $t^0(e,p)$ to entity e in process p. The three partition types are called ***interior*** *(*i*),* ***overlap*** *(*o*), and* ***ghost*** *(*g*).*

Remark 3 *Each entity has the partition type* ***interior*** *in exactly one process, thus providing a nonoverlapping decomposition of the entity set. In contrast,* ***overlap*** *elements exist in several processes because the numerical algorithm demands it explicitly (for example, overlapping Schwarz methods). Additional* ***ghost*** *elements may be necessary to ensure data consistency (see* (1) *below).*

Using these definitions we introduce the following notation:

$$\mathcal{E}^0|_p = \{e \in \mathcal{E}^0 \mid e\,\mathcal{D}^0\,p\}, \qquad \mathcal{E}^0|_p^\pi = \big\{e \in \mathcal{E}^0|_p \,\big|\, t^0(e,p) = \pi,\ \pi \in \{\mathtt{i}, \mathtt{o}, \mathtt{g}\}\big\},$$

and require that the interior elements $\mathcal{E}^0|_p^{\mathtt{i}}$ form a partition of the entity set:

$$\bigcup_{p \in P} \mathcal{E}^0|_p^{\mathtt{i}} = \mathcal{E}^0, \qquad \mathcal{E}^0|_p^{\mathtt{i}} \cap \mathcal{E}^0|_q^{\mathtt{i}} = \emptyset \ \text{ for all } \ p \neq q.$$

Analogously, we define the level-wise element partitions

$$E_j^0|_p = \{e \in E_j^0 \mid e\,\mathcal{D}^0\,p\}, \qquad E_j^0|_p^\pi = \big\{e \in E_j^0|_p \,\big|\, t^0(e,p) = \pi,\ \pi \in \{\mathtt{i}, \mathtt{o}, \mathtt{g}\}\big\},$$

for each grid level j.

Set $\mathcal{F}^0|_p = \mathcal{F} \cap (\mathcal{E}^0|_p \times \mathcal{E}^0|_p)$. We require that the father relation $\mathcal{F}$ can be represented in a distributed way. This means that $\bigcup_{p \in P} \mathcal{F}^0|_p = \mathcal{F}^0$. In order to ensure this we demand that if father and child are in different processes, the father must be introduced as a `ghost` entity in the child's process. In formulas

$$e\,\mathcal{D}^0\,p \wedge t^0(e,p) \in \{\mathtt{i}, \mathtt{o}\} \wedge f\,\mathcal{F}^0\,e \ \Rightarrow\ f\,\mathcal{D}^0\,p. \tag{1}$$

A similar requirement is needed in order to represent each intersection in at least one process. Therefore we require for each element e which is `interior` on process p that any other element e' is also known to process p if there is a set $I_e = \theta_e \cap \theta_{e'}$ in either $\mathcal{I}_e$ or $\tilde{\mathcal{I}}_{[e]}$. This means that the `interior` entities are surrounded by at least one layer of entities which are either `overlap` or `ghost`.

The purpose of the partition type is to define subdomains for each process p and refinement level j via

$$I_{j,p} = \bigcup_{e \in E_j^0|_p^{\mathtt{i}}} \theta_e, \qquad O_{j,p} = \bigcup_{e \in E_j^0|_p^{\mathtt{o}}} \theta_e, \qquad G_{j,p} = \bigcup_{e \in E_j^0|_p^{\mathtt{g}}} \theta_e.$$

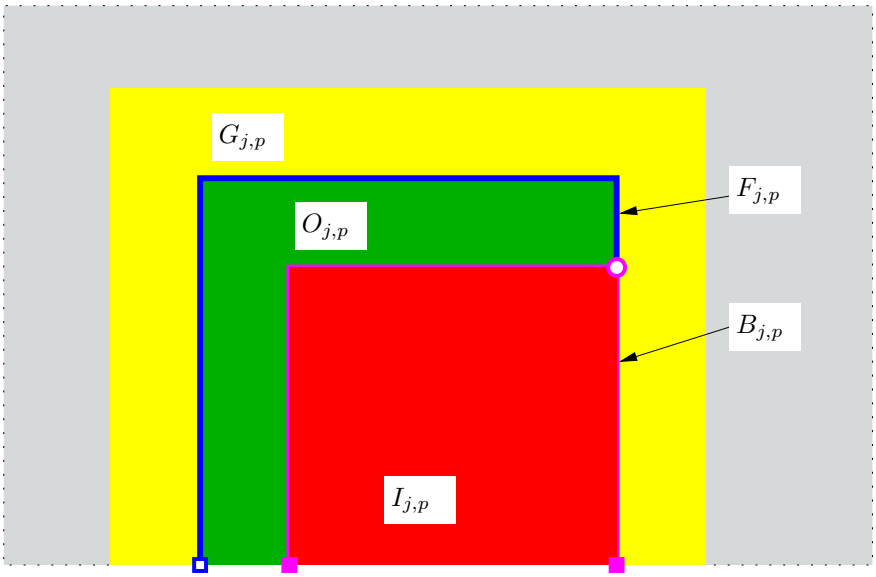

Figure 5: Data decomposition based on domain decomposition ideas.

We also define

$$\Theta_j = \bigcup_{e \in E_j^0} \theta_e.$$

These subdomains are illustrated in Figure 5. The $I_{j,p}$ are nonoverlapping by definition. Then $O_{j,p}$ is intended to "surround" $I_{j,p}$, and $G_{j,p}$ in turn "surrounds" $O_{j,p}$. Note here that `overlap` entities may not exist at all. In this case the corresponding subdomains are empty.

Up to now, only the decomposition of elements has been handled. The next definition extends the decomposition relation to positive codimensions. We begin by introducing the `border` and `front` boundaries. We define the `border` boundary as a part of the boundary of the domain of `interior` elements $\partial I_{j,p}$

$$B_{j,p} = \overline{\partial I_{j,p} \setminus \partial \Theta_j}.$$

Here $\overline{\cdot}$ denotes the closure of a point set. The `front` boundary is defined as the boundary between `overlap` and `ghost` elements.

$$F_{j,p} = \left(\overline{\partial O_{j,p} \setminus \partial \Theta_j}\right) \setminus B_{j,p}.$$

These boundaries are also illustrated in Figure 5. Note that due to this construction the points "■" and "○" are part of `border` $B_{j,p}$ and point "□" is part of `front` $F_{j,p}$. Note also that these definitions work when `overlap` entities are missing (then the `front` boundary is empty) or `ghost` entities are missing.

Definition 22 (Extended decomposition) *The extended decomposition relation $\mathcal{D} \subseteq \mathcal{E} \times P$ is defined using the subentity relation. For all $e \in \mathcal{E}^0$, $s \in \mathcal{E}$, such that $s\,\mathcal{S}\,e$ the equivalence*

$$s\,\mathcal{D}\,p \;\Leftrightarrow\; e\,\mathcal{D}^0\,p \tag{2}$$

must hold for all $p \in P$, i.e., an entity and all its subentities are always present together in a process.

Due to the reflexivity of $\mathcal{S}$ we have $\mathcal{D}^0 \subseteq \mathcal{D}$. Another consequence of (2) is that the subentity relation can be represented in the distributed model, i.e.,

$$\mathcal{S}|_p := \mathcal{S} \cap (\mathcal{E}|_p \times \mathcal{E}|_p), \qquad \bigcup_{p \in P} \mathcal{S}|_p = \mathcal{S}.$$

Partition types are generalized to higher codimensions with the map

$$t : \mathcal{D} \to \{\mathtt{i}, \mathtt{b}, \mathtt{o}, \mathtt{f}, \mathtt{g}\}.$$

Two more partition types, `border` (`b`) and `front` (`f`), are introduced which correspond to the boundaries $B_{j,p}$ and $F_{j,p}$ of the subdomains, respectively. For codimension 0 entities we have $t(e,p) = t^0(e,p)$. For any $(e,p) \in \mathcal{D}, \dim e < d$ the partition type is defined geometrically using the subdomains and boundaries defined above:

$$t(e,p) = \begin{cases} \mathtt{i} & \text{if } \operatorname{int}\theta_e \subseteq I_{L(e),p} \setminus B_{L(e),p} \\ \mathtt{b} & \text{if } \theta_e \subseteq B_{L(e),p} \\ \mathtt{o} & \text{if } \operatorname{int}\theta_e \subseteq O_{L(e),p} \setminus (B_{L(e),p} \cup F_{L(e),p}) \\ \mathtt{f} & \text{if } \theta_e \subseteq F_{L(e),p} \\ \mathtt{g} & \text{if } \operatorname{int}\theta_e \subseteq G_{L(e),p} \setminus (B_{L(e),p} \cup F_{L(e),p}). \end{cases}$$

Note that in case θ_e is a single point $\theta_e = \{x\}$ we use the definition $\operatorname{int}\{x\} = \{x\}$.

Remark 4 (Data exchange) *Besides the decomposition the data exchange is an important part of any parallel algorithm. In our concept this means that data associated with the same entity in different processes is to be exchanged, and any parallel grid implementation has to support this. Formally this can be described as follows. In each process $p \in P$ select a set of source entities*

$$\Sigma_p \subseteq \mathcal{E}|_p$$

and a set of destination entities

$$\Delta_p \subseteq \mathcal{E}|_p.$$

Then a general communication operation moves data for each $e \in \Sigma_p \cap \Delta_q$ from process p to process $q \neq p$ or vice versa. In the implementation predefined subsets Σ_p, Δ_p are available, see [1].

6 Index maps

An important part of our concept is that grids are completely separated from any numerical data associated with them. However, in order to access the data of a grid some link is needed between grid entities and the data. This link is provided by the index maps introduced in this section.

Since we like to consider adaptive grid refinement a computation actually uses a sequence of different grids rather than a single grid. We assume that a computation consists of alternating phases of work to be done on a fixed grid and grid modification (refinement, coarsening, or load-balancing). This is reflected by the following definition.

Definition 23 (Grid sequence) *An adaptive computation produces a sequence of entity sets*

$${}^0\mathcal{E}, {}^1\mathcal{E}, \ldots, {}^k\mathcal{E}, \ldots.$$

Given a grid ${}^k\mathcal{E}$, the next grid in the sequence is created by the following two half-steps:

- *Choose a set $\mathcal{E}^{0,-} \subseteq {}^k\mathcal{E}^0$. Remove all elements in $\mathcal{E}^{0,-}$ from ${}^k\mathcal{E}^0$. Also remove all subentities of $\mathcal{E}^{0,-}$ which are not also a subentity of ${}^k\mathcal{E}^0 \setminus \mathcal{E}^{0,-}$. Call the result ${}^{k+\frac{1}{2}}\mathcal{E}$. The set $\mathcal{E}^{0,-}$ contains elements which have been explicitly marked for removal, as well as elements which are removed automatically, such as green closure elements.*
- *Add new elements to ${}^{k+\frac{1}{2}}\mathcal{E}$ along with their subentities, as long as they are not already present. The result is ${}^{k+1}\mathcal{E}$. We expect the new elements to be chosen such that ${}^{k+1}\mathcal{E}$ is a valid grid in the sense of our definition.*

Note that by definition the dimension of the grid is the same for all k. For each $k \in \mathbb{N}_0$ the corresponding relations ${}^k\mathcal{S}, {}^k\mathcal{F}, {}^k\mathcal{D}$, and the map ${}^kt$ are defined accordingly.

The connection of entities to arbitrary user data is achieved by associating several indices with each entity. These indices can then be used to locate the data in an appropriate data structure, e.g., an array or a map (associative array). We now introduce three different index maps.

Definition 24 (Level index maps) *For a given sequence index k, codimension c, level j, reference element r, and process number p the level index is a map*

$$ {}^k\kappa_j^{c,r}|_p : {}^kE_j^{c,r}|_p \to \mathbb{N}_0. $$

The level index map is consecutive, which means that it is injective and

$$ 0 \leq {}^k\kappa_j^{c,r}|_p(e) < \left|{}^kE_j^{c,r}|_p\right| \quad \text{for all } e \in {}^kE_j^{c,r}|_p. $$

The level index maps are used to store data in arrays for efficient processing. From the individual maps ${}^k\kappa_j^{c,r}|_p$ one can easily construct consecutive maps for more general sets of entities. Entities are distinguished by reference element because two entities with the same reference element will usually carry the same number of degrees of freedom and hence array offsets are easier to compute.

Definition 25 (Leaf index map) *For a given sequence index k, codimension c, reference element r, and process number p the leaf index map*

$$ {}^k\lambda^{c,r}|_p : {}^kL^{c,r}|_p \to \mathbb{N}_0 $$

maps leaf entities to consecutive numbers, i. e., for all $k \in \mathbb{N}_0$ we require:

1. *(range restricted to number of equivalence classes)*

$$ \forall e \in {}^kL^{c,r}|_p : 0 \leq {}^k\lambda^{c,r}|_p(e) < \left|{}^k\tilde{L}^{c,r}|_p\right|, $$

2. *(equality only on entity copies)*

$$ \forall e, e' \in {}^kL^{c,r}|_p : {}^k\lambda^{c,r}|_p(e) = {}^k\lambda^{c,r}|_p(e') \Leftrightarrow e \sim e'. $$

Level and leaf indices are consecutive on their domains of definition. Hence a fortiori they have to change during grid modifications. In order not to lose data when going from one grid in the sequence to the next we need a form of indexing that does not change. This is provided by the persistent index map.

Definition 26 (Persistent index map) *For any sequence index k and any codimension c we define the persistent index map*

$$ {}^k\mu^c : {}^k\mathcal{E}^c \to \mathbb{I}, $$

where $\mathbb{I}$ is a set of indices. The persistent index maps are required to fulfill the following two conditions.

1. *(persistence)*

$$ e \in {}^k\mathcal{E}^c, e \in {}^{k+1}\mathcal{E}^c \Leftrightarrow {}^k\mu^c(e) = {}^{k+1}\mu^c(e), $$

2. *(equality only on entity copies)*

$$ \forall e, e' \in {}^k\mathcal{E}^c : \ {}^k\mu^c(e) = {}^k\mu^c(e') \Leftrightarrow e \sim e'. $$

These properties imply that if an entity is contained in two subsequent grids the persistent index does not change. If, however, $e \in {}^k\mathcal{E}$ and $e \in {}^{k+2}\mathcal{E}$ but not $e \in {}^{k+1}\mathcal{E}$ then the entity may have a different persistent index in ${}^k\mathcal{E}$ and ${}^{k+2}\mathcal{E}$.

Remark 5 (Use of the index maps) *Computations on grid sequences are assumed to be structured into alternating phases of work on a fixed grid and modifications of the grid to obtain the next grid in the sequence. While working on a fixed grid the level and leaf indices are used to access data in arrays with constant time complexity for each access. In the modification phase the persistent index is used to transfer data from one grid to the next grid. During that phase data is typically stored in an intermediate data structure with logarithmic time complexity for each access.*

References

[1] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, and O. Sander. A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in DUNE. *in preparation*, 2007.

[2] W. Benger. *Visualization of General Relativistic Tensor Fields via a Fiber Bundle Data Model*. PhD thesis, Freie Universität Berlin, 2005.

[3] G. Berti. *Generic Software Components for Scientific Computing*. PhD thesis, BTU Cottbus, 2000.

[4] N. Botta, C. Ionescu, C. Linstead, and R. Klein. Structuring distributed relation-based computations with SCDRC. Technical report, PIK Report No. 103, Potsdam Institute for Climate Impact Research, 2006.

[5] DUNE – Distributed and Unified Numerics Environment. `http://dune-project.org/`.