Adding Interpolation and Restriction Operators to Pooma Fields

Jeffrey D. Oldham

2001 Apr 23

Abstract

We propose adding interpolation and restriction operators to map between finer and coarser granularity fields. Implemented as field stencils, these operators would ease use of fields with different granularities as demonstrated in a Pooma 2.4 hydrodynamics kernel and in multigrid methods.

1 Interpolation and Restriction Operators

Currently Pooma field operations support only operands with the same geometries. To support a Pooma 2.4 hydrodynamics kernel and multigrid techniques, we propose adding interpolation and restriction operators to map between fields with finer and coarser granularities. Implemented as stencils, these operators would permit clean coding of the hydrodynamics kernel using Pooma 2.4. It would also expand the set of models that Pooma supports including, e.g., multigrid methods.

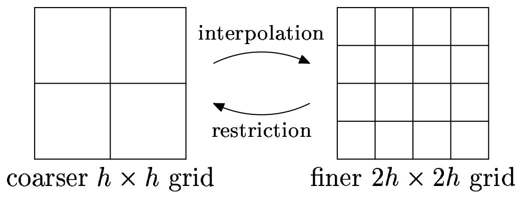

Figure 1: Interpolation and restriction operators convert between coarser and finer grained grids.

A hydrodynamics kernel implementing Caramana et al. [CBSW98] uses two staggered grids, one shifted half a cell up and over. Although all physical values are stored in these two grids, both with the same granularity, e.g., $h \times h$, intermediate computations are presented in and most easily written using grids with twice the granularity, e.g., $2h \times 2h$. Information from the $n \times n$ fields is interpolated to $2h \times 2h$ fields, computations are performed at the finer granularity, and then information is summarized, usually by summing, to restrict back to $h \times h$ fields.

In another use of interpolation and restriction operators, multigrid methods solve differential problems by solving problems by first solving problems on coarse grids and then repeatedly refining the solution on finer grids [BHM00, BM87, ST82]. For example, to solve a differential equation $Au = f$ for u, the algorithm is

1. Construct an initial guess u^h.
2. Compute remainder $r^h \leftarrow f^h - A^h u^h$.
3. Interpolate to a finer grid: $f^{2h} \leftarrow I_h^{2h} r^h$.
4. Approximately solve $A^{2h} u^{2h} = f^{2h}$ for u^{2h}.
5. Modify the guess $u^h \leftarrow u^h + I_{2h}^h u^{2h}$.
6. Goto Step 2.

The superscripts h and $2h$ indicate the grid granularities with $2h$ finer than h. The algorithm works by first computing the remainder, interpolating to a $2h \times 2h$ finer grid using the interpolation operator I_h^{2h}. After solving the resulting equation, the result on the fine grid is restricted back to the $h \times h$ coarse grid using a restriction operator I_{2h}^h.

2 Operator Use in a Hydrodynamics Kernel

To demonstrate the use of interpolation and restriction operators in Pooma code, we present relevant portions of the hydrodynamics kernel.

1. Given a cell-centered pressure density field and a vertex-centered coordinate field both with $h \times h$ grids, the masses of a $2h \times 2h$ finer-grain cell-centered field are computed.

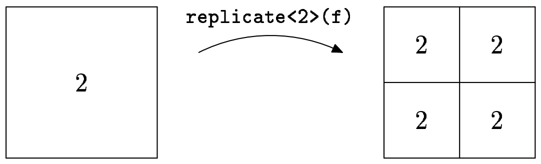

Figure 2: Replicating values just copies them.

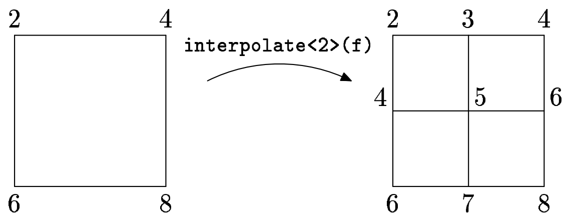

Figure 3: Interpolating values uses averages.

```
masses = replicate<2>(pressureDensity) *
         computeVolumes(interpolate<2>(coordinates));
```

Here `replicate<2>(pressureDensity)` produces a two-dimensional field with twice the granularity for both dimensions by copying each cell-centered `pressureDensity` value into its four associated cells in the finer grid. Since the operators do not specify any centerings, the input and output fields are assumed to have the same centerings.

`interpolate<2>(coordinates)` also produces a two-dimensional field with twice the granularity for both dimensions but, instead of copying each `coordinates` value, it produces values in the finer grid by interpolating between the two closest `coordinates` entries. The `computeVolumes` field stencil (not an interpolation or restriction operator) computes the volume of each cell. Finally, the fields returned by the two operations are multiplied.

2. The `total` restriction operator eases computing the mass of each cell and each vertex.

```
cellMass = total<2>(masses);
vertexMass = total<2,Vertex,Cell>(masses);
```

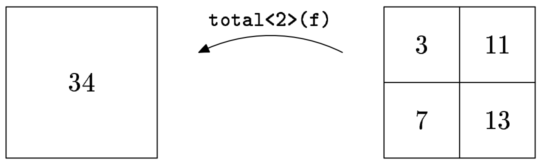

Figure 4: Restricting by summing cells.

These restriction operators map from a two-dimensional field to one with half the granularity for both dimensions by assigning a value equal to its four associated values in the finer grid. See Figure 4. Computing vertex masses converts from a cell-centered field to a vertex-centered field so these centerings are specified in the operator. Computing cell masses does not convert between centerings so these are not specified.

3. Computing the four corner forces in each cell requires a moderately complicated interpolation formula.

   ```
   normals = computeNormals(coordinates);
   forces = replicate<2>(pressure)
     * addNormals<2,Cell,AllFace>(normals);
   ```

   `computeNormals` uses the cell-centered `coordinates` field to compute the faces' normals. This operator, not an interpolation nor restriction operator, demonstrates the need for operator implementations to be able to refer to values in fields with different centerings.

   The second line illustrates the need for programmers to be able to add their own program-specific interpolation operators such as `addNormals`. Given a face-centered field of normals, this operator forms a finer grained field with each cell-centered value with the sum of incident normals.

4. The corner forces are used in two different computations.

   ```
   velocityChange = constant1 * total<2,Vert,Cell>(forces);
   internalEnergy += constant2 *
     total<2>(dot(forces,
       replicate<2,Cell,Vert>(velocity + velocityChange)));
   ```

The first computation sums the four forces around each vertex, illustrating that cell-centered values can be restricted to a vertex-centered value if specified as template parameters. The second computation replicates the sum of two vertex-centered fields into a cell-centered field. Then it sums the dot product of these values using the `total` restriction operator.

This sample code illustrates the need for

- an ability to easily add custom interpolation and restriction operators,
- support for only integral changes to dimensions, and
- operators that deal with different centerings.

Not illustrated is the requirement that the user, not the compiler, ensures that fields assigned values have the proper size, an assumption that Pooma already makes.

To support dimension-independent code and to ease implementation, interpolation and restriction operators modify all dimensions by the same amount, an amount specified by a template parameter. Mathematically, different multipliers for different dimensions are possible, but supporting this might require an exponential number of operator definitions. Specifying by a template parameter permits Pooma to produce data-parallel execution.

2.1 Semantics of Operators with Different Centerings

The input fields of several operators have different centerings than the resulting output field. We specify these operators' semantics. For example, `total<2,Vertex,Cell>(masses)` operates on the cell-centered `masses` to form a vertex-centered field by summing the values of incident cells To use the operator, the programmer must ensure that there is at least one layer of guard cells around the input field. This guard layer ensures that all vertices obtaining a value are interior vertices with 2^d surrounding cells.

To compute the normals using `computeNormals(coordinates)` requires computing face-centered values using vertex-centered values. Thus, we need a notation for a face's incident vertices. The notation must permit easy differentiation of faces on opposite sides of the polytope so the normals can point in the proper direction. I do not know how to implement it.

Summing incident normals in `addNormals<2,Cell,AllFace>(normals)` indicates the need to determine faces incident to a cell. Given a coarse-grained face-centered field, a finer-grained cell-centered field is created. Each 2^d cells form one coarse-grained cell with the associated faces.

To compute the internal energy, we use

```
internalEnergy += constant2 * total<2>(dot(cornerForce,
  replicate<2,Cell,Vert>(velocity + velocityChange)));
```

The `cornerForce` field is cell-centered so the `replicate`d field must also be cell-centered. The velocity sum field is vertex-centered, as indicated by the third template parameter. For each value in a coarse vertex-centered field, the vertices in the corresponding polytope in a vertex-centered replicated field all have the same value. This polytope's "smallest" vertex matches the vertex in the coarse field. To form a cell-centered field, shift all values half a unit in negative directions. Since there are vertices surrounding all cells, there is no need for guard layers.

3 Implementing the Operators

To implement interpolation and restriction operators, we propose extending field stencils. The heart of a field stencil is its function call operator `()`. A declaration for a two-dimensional `replicate` operator could be

```
template <unsigned xOffset, unsigned yOffset>
operator()(const F & f, int fx, int fy) const;
```

The operator's arguments are a field reference `f` and a point $(\texttt{fx}, \texttt{fy})$ in the field. If the operator is to produce a grid with twice the granularity, the four arguments for the template parameters will be $< \texttt{0}, \texttt{0} >$, $< \texttt{0}, \texttt{1} >$, $< \texttt{1}, \texttt{0} >$, and $< \texttt{1}, \texttt{1} >$. Using template arguments permits compile-time specialization, which is necessary for `interpolate`.

Restriction operators would also be implemented using field stencils. Instead of applying a field stencil, e.g., `total<2>`, at each point in a fine-grained grid, the stencil would be applied to every, e.g., `2`, locations in each direction. For example, it would be applied to positions $(0,0), (2,0), (4,0), \ldots, (0,2), (2,2), \ldots$. This stride would be stored in the stencil.

4 Default Operators

Default restriction operators that Pooma should support include

`sum`	total values in the section corresponding to one coarse grid point
`prod`	multiple of all values in the section
`max`	maximum of all values in the section
`min`	minimum of all values in the section
`all`	boolean and of all (boolean) values in the section
`any`	boolean or of all (boolean) values in the section
`bitAnd`	bitwise and of all values in the section
`bitOr`	bitwise or of all values in the section
`average`	average of all values in the section

Default interpolation operators that Pooma should support should include

`replicate`	copy value from coarse grid point to all corresponding points
`interpolate`	each value is the average of its two closest coarse-grid points

References

[BHM00] William L. Briggs, Van Emden Hensen, and Steve F. McCormick. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[BM87] W. Briggs and S. McCormick. Introduction. In Stephen F. McCormick, editor, *Multigrid Methods*, Frontier in Applied Mathematics, pages 1–30. Society for Industrial and Applied Mathematics, Philadelphia, 1987.

[CBSW98] E. J. Caramana, D. E. Burton, M. J. Shashkov, and P. P. Whalen. The construction of compatible hydrodynamics algorithms utilizing conservation of total energy. *Journal of Computational Physics*, 146:227–262, 1998.

[ST82] Klaus Stüben and Ulrich Trottenberg. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In W. Hackenbusch and U. Trottenberg, editors, *Multigrid Methods: Proceedings of the Conference Held at Köln-Porz*, volume 960 of *Lecture Notes in Mathematics*, pages 1–176, Berlin, 1982. Springer-Verlag.