

Transformations in the PRISM coupler

1 Interpolations

The fields considered in the PRISM coupling context will be 3D fields (a 2D field could be seen as a 3D field with a dimension equal to one). Thus for any interpolation operation the user will have to precise in the SCC what kind of interpolation he wants to apply:

- **3DFULL** (a full 3D interpolation),
- **2D+VERT** (a 2D interpolation on each horizontal level of a 3D grid then a 1D interpolation in the 3rd dimension),
- **1D+1D+1D** (three 1D interpolation for a 3D grid)
- **2D+NONE** (a 2D interpolation on the horizontal grid for a 2D grid),
- **1D+1D+NONE** (two 1D interpolation for a 2D grid),
- **1D+NONE+NONE** (a 1D interpolation for a 1D grid).

2 2D Nearest Neighbor(s) interpolation function

2.1 Description:

For each target grid point, the N nearest neighbors on the source grid, weighted by their distance, are averaged.

In case of nearest neighbor(s) gaussian weighted function, the neighbors, weighted by the value of a gaussian function at their distance from the target point, are averaged.

2.2 Arguments/options:

- The type of the field: **SCALAR or VECTOR**, (given through the PSMILe)
- If the partition covers the poles: **POLE or NOPOLE** (this argument can be given through the PSMILe partition per partition)
- If the mask should be considered or not in the interpolation (this could involve the use of extrapolation or not): **MASK or NOMASK**. (SCC)
- **N**, the number of neighbors, (given in the SCC)
- **S**, the variance of the gaussian function, in case of gaussian weighted interpolation function (SCC).

2.3 Remarks

In case of distributed field, the halo needed for the nearest neighbor(s) search cannot really be dimensioned. Two approaches are thus possible:

- The halo given by the developer is considered as sufficient to see the local nearest neighbors as the global ones. This method is certainly the cheapest one.
- The user wants to perform a precise interpolation. Thus, as it is not possible to rely on the halo to find the nearest neighbor(s), it is necessary to search locally (on each partition) the nearest neighbor(s) and then compare the local results or gather the partitions to perform a global search.

2.4 Steps and Locations

1- *Search of the nearest neighbor(s)*. This search can be implemented following two different goals:

- Considering the halo given by the developers as sufficient for the search leads to implement a cheap but qualitatively deficient method (Most of the interpolation would be efficient but...).
- The second strategy insures to give the right result but implies a high cost. In fact, gathering the partitions on a single process or comparing the local results implies many extra communications between the processes.

Location: PSMILe

2- *Computation of the weights*. This step is done considering the distance from the source points to the target points. In case of gaussian interpolation, the variance is used. In case of vector fields, if the partition covers a pole, projections of the source vector field in a fixed Cartesian coordinate system needs to be performed first.

Location: Transformer

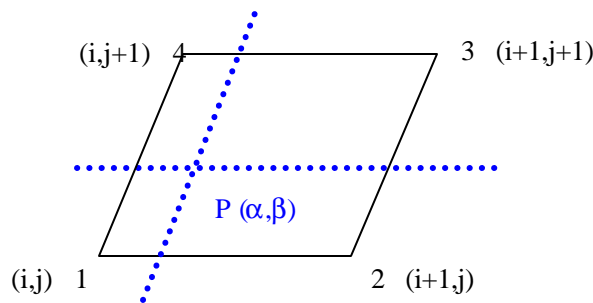
3- *Computation of the interpolated field*. Using the weights computed as described previously, and the source field sent by the PSMILe routines, the interpolated field is calculated.

Location: Transformer

3 First order interpolation function (bilinear)

3.1 Description

Standard bilinear interpolation schemes can be found in many textbooks. Here, following the SCRIP user's guide, we present a more general scheme that uses a local bilinear



approximation to interpolate a point in a quadrilateral grid in a spherical geometry.

Consider the grid points shown in Figure 1 labeled with logically rectangular indices (e.g. (i,j)).

Let the latitude-longitude coordinates of point 1 be $(\theta(i,j), \varphi(i,j))$, the coordinates of point 2 be $(\theta(i+1,j), \varphi(i+1,j))$, etc.

Figure 1: Illustration of the bilinear interpolation method, once the concerned source grid points had been found. $P(\alpha,\beta)$ is being interpolated

Now let α and β be continuous local coordinates such that the coordinates (α,β) of point 1 are (0,0), point 2 are (1,0), point 3 are (1,1) and point 4 are (0,1). If point P lies inside the cell formed by the four points above, the function f at point P can be approximated by

$$f_p = (1-\alpha)(1-\beta)f(i,j) + \alpha(1-\beta)f(i+1,j) + \alpha\beta f(i+1,j+1) + (1-\alpha)\beta f(i,j+1) \\ = w_1 f(i,j) + w_2 f(i+1,j) + w_3 f(i+1,j+1) + w_4 f(i,j+1)$$

The remapping weights must therefore be computed by finding α and β at point P that corresponds to the distance between P and the origin of the local coordinates.

3.2 Arguments/options:

- The grid type: **LR** (logically rectangular grid) or **RD** (reduced grid), (PMIOD)
- The field type: **SCALAR** or **VECTOR**, (given through the PSMILe)
- If the partition covers the poles: **POLE** or **NOPOLE** (this argument can be given through the PSMILe partition per partition)
- If the mask should be considered or not in the interpolation (this could involve the use of extrapolation or not): **MASK** or **NOMASK**. (SCC)

3.3 Remarks

- 1- Because of the method, the size of the needed halo to compute the weights is one.
- 2- What could be the strategy if the developer does not give the halo? Should we provide the possibility to recompose the halo by searching the right neighbors through the different partitions of the grid? Should we just use an extrapolation for the bordering points?

3.4 Steps and Locations

1- *Search of the four Cartesian neighbor(s)*. This search is detailed above. If the dimension of the halo is one, the partition's domain is sufficient to find the four neighbors. If not, a search through the different partition or an extrapolation needs to be performed.

Location: PSMILe

2- *Computation of the weights*. This step is done following the method described in Section 2.1.

Location: Transformer

3- *Computation of the interpolated field*. Using the weights computed as described previously, and the source field sent by the PSMILe routines, the interpolated field is calculated.

Location: Transformer

4 Second order interpolation function (bi-cubic)

4.1 Description

The bi-cubic remapping corresponds to:

$$f_p = (1-\beta^2(3-2\beta))(1-\alpha^2(3-2\alpha))f(i,j) + \\ (1-\beta^2(3-2\beta))\alpha^2(3-2\alpha)f(i+1,j) +$$

$$\begin{aligned}
& \beta^2(3-2\beta)\alpha^2(3-2\alpha)f(i+1,j+1) + \\
& (1-\beta^2(3-2\beta))(1-\alpha^2(3-2\alpha))f(i,j+1) + \\
& (1-\beta^2(3-2\beta))\alpha(\alpha-1)^2 \frac{\partial f}{\partial i}(i,j) + \\
& (1-\beta^2(3-2\beta))\alpha^2(\alpha-1) \frac{\partial f}{\partial i}(i+1,j) + \\
& \beta^2(3-2\beta)\alpha^2(\alpha-1) \frac{\partial f}{\partial i}(i+1,j+1) + \\
& \beta(\beta-1)\alpha(\alpha-1)^2 \frac{\partial f}{\partial i}(i,j+1) + \\
& \beta^2(\beta-1)^2(1-\alpha^2(3-2\alpha)) \frac{\partial f}{\partial j}(i,j) + \\
& \beta^2(\beta-1)\alpha^2(3-2\alpha) \frac{\partial f}{\partial j}(i+1,j) + \\
& \beta^2(\beta-1)\alpha^2(3-2\alpha) \frac{\partial f}{\partial j}(i+1,j+1) + \\
& \beta^2(\beta-1)(1-\alpha^2(3-2\alpha)) \frac{\partial f}{\partial j}(i,j+1) + \\
& \alpha(\alpha-1)^2\beta(\beta-1)^2 \frac{\partial^2 f}{\partial i \partial j}(i,j) + \\
& \alpha^2(\alpha-1)\beta(\beta-1)^2 \frac{\partial^2 f}{\partial i \partial j}(i+1,j) + \\
& \alpha^2(\alpha-1)\beta^2(\beta-1) \frac{\partial^2 f}{\partial i \partial j}(i+1,j+1) + \\
& \alpha(\alpha-1)^2\beta^2(\beta-1) \frac{\partial^2 f}{\partial i \partial j}(i,j+1)
\end{aligned}$$

where α and β are identical to those found in the bilinear case. The four weights corresponding to each address pair correspond to the weight multiplying the field value at the point, the weight multiplying the gradient with respect to i , the weight multiplying the gradient with respect to j and the weight multiplying the cross gradient in that order. It is also possible to consider sixteen points, to compute the gradient rather than getting the gradient value through the reception of another field.

4.2 Arguments/options:

- The grid type: **LR** (logically rectangular grid) or **RD** (reduced grid), (SMIOC)
- The field type: **SCALAR** or **VECTOR**, (given through the PSMILe)
- If the partition covers the poles: **POLE** or **NOPOLE** (this argument can be given through the PSMILe partition per partition)
- If the mask should be considered or not in the interpolation (this could involve the use of extrapolation or not): **MASK** or **NOMASK**. (SCC)
- The method for the bi-cubic interpolation (SCC):
 - o **GRAD**, for using 4 points and the value of the gradient at these points given through another coupling field,
 - o **NOGRAD**, for using 4 points and compute the value of the gradient at these points (i.e. for this method we need 16 points),
 - o **SIXTEEN**, for using 16 points (this method differs from the previous one because of the use of hybrid method if the needed points are not given).

4.3 Remarks

- 1- The size of the needed halo to compute the weights is one, if the gradient is given through another field, or two if there is a need to compute this gradient or use sixteen points.

- 2- What could be the strategy if the developer does not give the halo? Should we provide the possibility to recompose the halo by searching the right neighbors through the different partitions of the grid? Should we just use an extrapolation for the bordering points?

4.4 Steps and Locations

1- *Search of the four Cartesian neighbor(s)*. This search is detailed above. If the gradient is given through another field and if the dimension of the halo is one, the partition's domain is sufficient to find the four neighbors. If the gradient is computed or if the computation is done with 16 points and if the dimension of the halo is 2, the partition's domain is sufficient to find the four or sixteen neighbors. If not, a global search through the different partition or an extrapolation needs to be performed.

Location: PSMILe

2- *Computation of the weights*. This step is done following the method described in Section 3.1.

Location: Transformer

3- *Computation of the interpolated field*. Using the weights computed as described previously, and the source field sent by the PSMILe routines, the interpolated field is calculated.

Location: Transformer

5 Conservative Remapping

5.1 Description

This scheme guarantees that the area-integrated field is conserved between the source and the target grid. Indeed, for each target mesh, the weight assigned to each underlying source mesh is:

- proportional to the overlapping surface, not taken into account for the ratio the source masked cells: *DESTAREA*,
- proportional to the overlapping surface, taken into account locally for the ratio the source masked cells (the flux is conserved but just redistributed locally along the intersected cells): *FRACAREA*.

The option *TRUEAREA* insures that the areas, as considered by the model, are used to normalize the results.

The option *CONSERV* added to one of the previous method insures the global conservation of the flux, re-dispatching the extra-flux over the globe (the extra flux is averaged and globally redistributed over the whole grid).

The option *NEARNEI* insures that nearest neighbor value is used for target grid and meshes having no intersection with any unmasked source meshes.

5.2 Arguments/options:

- If the partition covers the poles: *POLE* or *NOPOLE* (this argument can be given through the PSMILe partition per partition)

- If the mask should be considered or not in the interpolation (this could involve the use of extrapolation or not): *MASK* or *NOMASK*. (SCC)
- The normalization option: *DESTAREA* / *FRACAREA* / *TRUEAREA* / *CONSERV* / *NEARNEI* / *NONE*, (SCC)
- The order of the remapping: *FIRST* or *SECOND*, (SCC)
- For *SECOND*, the longitudinal, latitudinal and cross gradients are needed. They are given as auxiliary fields or approximately computed.

5.3 Steps and Locations

1- For each side of the target cells, *find the different intersections* with the source cells, and *compute local intersected areas*.

Location: PSMILe

2- *Compute the intersected areas* between the target cell and the intersected source cells, Location: it could be necessary to gather the results of different source partitions if, for example, a target cell intersects source cells from two source partitions.

3- *Compute the weights* according to the intersected areas, and *normalization*.

Location: Transformer.

4- *Computation of the interpolated field*. Using the weights computed as described previously, and the source field sent by the PSMILe routines, the interpolated field is calculated.

Location: Transformer

6 Algebraic operations, Combinations or Merge

6.1 Description

This local and point-wise operation combines different parts of different coupling fields or of other predefined external data. This operation may involve the smoothing of the fields near the different domain borders, the addition, subtraction, multiplication, etc., with possibly different coupling fields and numbers as operands (+, -, x, ^2, SIN, LOG, ...).

6.2 Arguments/options:

- Symbolic name of the operation. In the SCC, all the fields involved in the same algebraic operation will refer to the same operation symbolic name. (SCC)
- For each field involved in the operation:
 - o The operand: **PLUS**, **MINUS**, **TIME**, **SIN**, **LOG**, etc... (SCC)
 - o The value of the scalar to apply, (SCC)
 - o If the mask should be considered or not in the operation: **MASK** or **NOMASK**. (SCC)

6.3 Remarks

1- If an operation involves several fields, the composition of the operation is done retrieving the concerned field in the SCC using the operation symbolic name.

6.4 Steps and Locations

- 1- If necessary, *interpolate the field* involved in the operation on the destination grid,
- 2- *Apply any scalar product or function to the different fields*,
- 3- *Assemble the different fields* (addition, subtraction, etc...).

Location: The location depends on the need of interpolation:

- 1- If all involved fields come from the same source models, the operations can be performed in the source model PSMILe before sending the result of the operations to the transformer or to the target model PSMILe.
- 2- If the involved fields come from different source models which have the same grid than the target model, each source model manages its own part of the algebraic operation and sends the result to the target model PSMILe which assembles the results.
- 3- If all the fields need an interpolation operation, the algebraic operations and the assembling is done in the transformer. The result is then sent to the target model PSMILe.
- 4- In the other cases, some fields need to be interpolated, the interpolation is insured in the transformer. The other fields are sent directly to the target model PSMILe, where the assembling.

7 Scattering

7.1 Description

This local operation scatters the model data onto the points listed in an index.

8 Gathering

8.1 Description:

This local operation gathers from the input data all the points listed in an index.

9 Collapse

9.1 Description

This local operation results in the collapse of any dimension or combination of dimensions by various, possibly weighted, statistical operations, such as mean, max, min, etc. (possibly relative to a threshold, e.g. maximum of positive values).

9.2 Arguments/options:

- The dimension to collapse: *X, Y, Z*,
- The option of collapsing: *MEAN, MAX, MIN*, etc...

9.3 Remarks

1- the collapsing can be performed vertically or horizontally

9.4 Steps and Locations

1- *Apply the collapsing method to the field*

Location: The location depends on the correlation between the distribution and the collapsing dimension. If the field to collapse vertically is distributed horizontally over the source processes, the collapsing can be performed into the PSMILe layer. If the distribution is organized over the vertical dimension, one needs to gather the whole field in the transformer to collapse it. Etc...

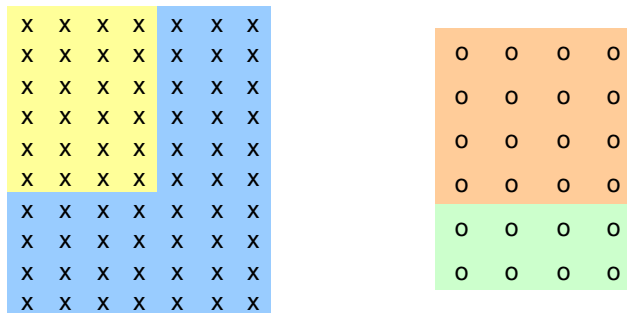
10 Implementation of the interpolation operations

In the general sense, the **grid** represents the computational domain of a model. Thus a grid could be distributed on several processes. The **partition** is the part of the grid distributed on a process.

10.1 Neighborhood search

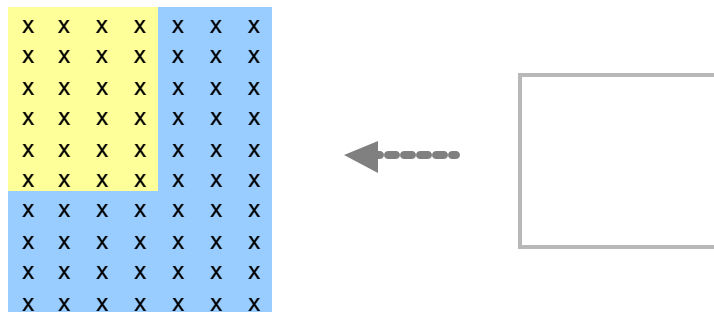
The previous step of any interpolation between two distributed grids consists in determining for any point of a target grid partition (a partition is the part of the grid that belongs to one process), what will be the source points of a source partition involved in the computation of its interpolation weights.

Let's consider the following example:

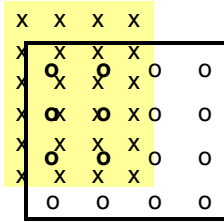


The source points (crosses) belong to a grid distributed on two processes (process 1 is in yellow whereas process 2 is in blue). The target points (circles) belong also to a grid distributed on two processes (process 1 is in orange whereas process 2 is in green). In the following, we will detail the tasks needed to perform the interpolation between the target process 1 and the source process 1.

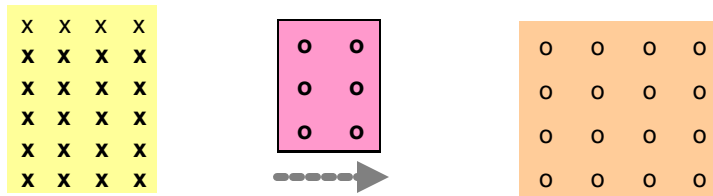
- 1- The target process 1 sends to source process 1 the envelop of its partition.



- 2- The source process 1 evaluates, considering the interpolation method (the neighbors to find for a bilinear and a nearest neighbor interpolation won't be the same) and the halo of each partition, what target points can find their neighbors in its source points. In this example, the six target points of the left upper corner can find for example their neighbors in the source process 1 for a bilinear interpolation.



3- The source process 1 informs the target process 1 what target points will be computed with its source points.



The ensembles of points involved in the interpolation are now ready to be sent to the transformer. Let's call EPIOS(1,1) – Ensemble of Points for the Interpolation Operation Source – the ensemble of the twenty bottom points and EPIOT(1,1) – Ensemble of Points for the Interpolation Operation Target – the ensemble of the six left upper corner points.

For an interpolation between two components A and B distributed on P_a and P_b processes, the EPIOS(i,j) will gather all the source points of process i ($0 < i < P_a - 1$) that are considered as neighbors of target points of the process j ($0 < j < P_b - 1$) for the interpolation operation. The EPIOT(i,j) will gather all the target points of process j which have their corresponding neighbors in the partition of process i .

For an interpolation between component A distributed on 2 processes and component B distributed on 2 processes, the EPIOS will be the intersections between:

- B1 and A1 (EPIOS(1,1)),
- B1 and A2 (EPIOS(2,1)),
- B2 and A1 (EPIOS(1,2)),
- B2 and A2 (EPIOS(2,2)).

The target EPIOT will be:

- The points belonging to B1 that have neighboring points in A1 (EPIOT(1,1)),
- The points belonging to B1 that have neighboring points in A2 (EPIOT(2,1)),
- The points belonging to B2 that have neighboring points in A1 (EPIOT(1,2)),
- The points belonging to B2 that have neighboring points in A2 (EPIOT(2,2)).

Therefore, the interpolation between A and B maybe performed in four operations: between EPIOS(1,1) and EPIOT(1,1), between EPIOS(2,1) and EPIOT(2,1), between EPIOS(1,2) and EPIOT(1,2), and between EPIOS(2,2) and EPIOT(2,2),.

10.2 Transfer of information between the PSMILe and the Transformer

The transfer of these informations (EPIOS and EPIOT) can be managed following different strategies:

- 1st strategy:
At the end of the definition phase (inside the PRISM_Enddef), each source and target process transfers informations of the local partition (latitudes, longitudes, z coordinates, mask, area, etc...). Each source and target process transfers the indices of the EPIOS and EPIOT in their partition local space, as well as the neighboring addresses.
During the coupling phase, the source field is sent partition per partition to the transformer.
- 2nd strategy:
At the end of the definition phase, each source and target process transfers informations of the points that belong to EPIOS or EPIOT domains (latitudes, longitudes, z coordinates), recreating a pseudo-local space for each EPIOS and for each EPIOT. There is no optimization concerning point's redundancies. Thus the information of a point that belongs to two EPIOS for example will be duplicated twice in the transformer memory.
Each source and target process transfers the other informations (masks, areas, etc...) following this new space format.
During the coupling phase, the source field is also pre-formatted to be sent to the transformer.
- 3rd strategy:
At the end of the definition phase, each source and target process transfers informations of the points that belong to the unions of the EPIOS, or the unions of EPIOT domains (latitudes, longitudes, z coordinates), recreating a pseudo-local space for the union of the local EPIOS and a pseudo-local space for the union of the local EPIOT.
Each source and target process transfers the other informations (masks, areas, etc...) following this new space format.
During the coupling phase, the source field is also pre-formatted to be sent to the transformer.

Let's reconsider an example to illustrate this purpose.

01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49
50	51	52	53	54	55	56
57	58	59	60	61	62	63
64	65	66	67	68	69	70

01	02	03	04
05	06	07	08
09	10	11	12
13	14	15	16
17	18	19	20

In the global space of source model represented by the numbers 01 to 70:

- EPIOS(1,1) = (8, 9, 10, 11, 15, 16, 17, 18, 22, 23, 24, 25, 29, 30, 31, 32)
- EPIOS(2,1) = (12, 13, 14, 19, 20, 21, 26, 27, 28, 33, 34, 35, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56)
- EPIOS(2,2) = (57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70)

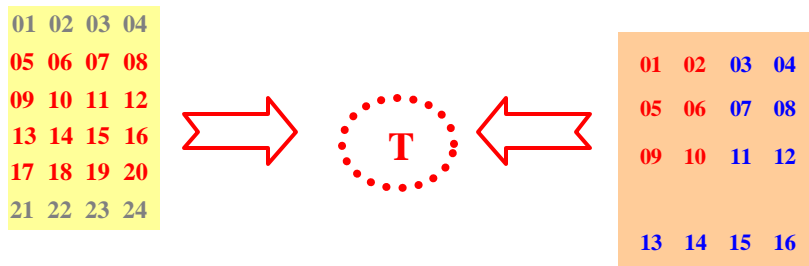
In the global space of target model represented by the numbers 01 to 20:

- EPIOT(1,1) = (1, 2, 5, 6, 9, 10)
- EPIOT(2,1) = (3, 4, 7, 8, 11, 12, 13, 14, 15, 16)
- EPIOT(2,2) = (17, 18, 19, 20)

Let's illustrate the different strategy for this example:

- 1st strategy:

The source and target components, through the PSMILe, send to the transformer, for each partition, the latitudes, longitudes, mask and areas arrays which dimensions are the partitions' dimensions: 24 (4x6) for source process 1, 46 for source process 2, 16 for target process 1, 4 for target process 2.



Each process then sends its EPIOS or EPIOT, in its local space i.e. the source process 1 sends EPIOS(1,1) as (5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20).

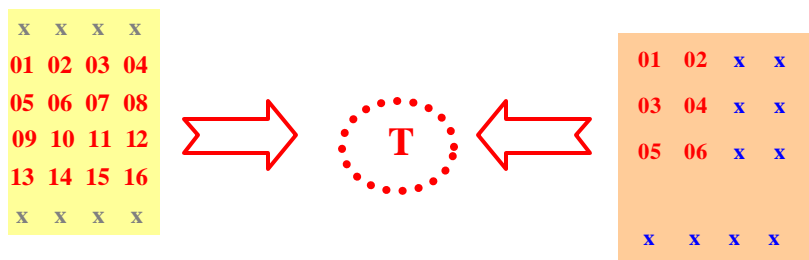
Finally, each source process sends, for each EPIOS, the neighboring addresses in the partition's space, for each corresponding EPIOT.

The neighboring addresses for the interpolation between source process 1 and target process 1 for each of the 6 points in EPIOT(1,1) are for a bilinear interpolation (4 source neighbors for each target point): (/5, 6, 9, 10/, /7, 8, 11, 12/, /9, 10, 13, 14/, /11, 12, 15, 16/, /13, 14, 17, 18/, /15, 16, 19, 20/)

- 2nd strategy:

The source and target components, through the PSMILe, sends to the transformer, for each EPIOS or EPIOT, the latitudes and longitudes arrays which dimension are the EPIOS/EPIOT dimensions: 16 for EPIOS(1,1), 6 for EPIOT(1,1), etc...

The other informations are then sent following this format. Thus the mask and areas arrays sent relatively to the EPIOS(1,1) have the size 16 and are indexed following the EPIOS(1,1).

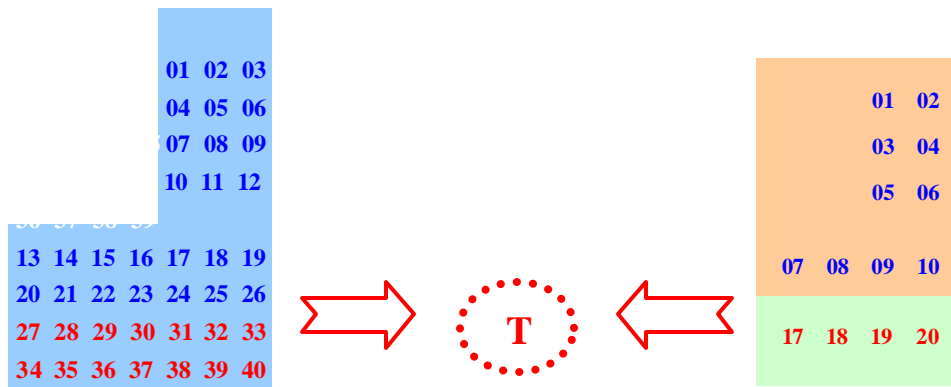


The neighboring addresses are also formatted in this local space. This gives for each of the 6 points in EPIOT(1,1): (/1, 2, 5, 6/, /3, 4, 7, 8/, /5, 6, 9, 10/, /7, 8, 11, 12/, /9, 10, 13, 14/, /11, 12, 15, 16/). These addresses are sent to the transformer.

- 3rd strategy:

The source and target components through the PSMILE sends to the transformer, for each process, the latitudes and longitudes arrays, representing the union of the different EPIOS/EPIOT of each process, which dimension, for example, is 40 for the union of the EPIOS on source process 2.

The other informations are then sent following this format. Thus the mask and areas arrays sent relatively to the source process 2 have the size 40 and are indexed following the union of EPIOS(2,1) and EPIOS(2,2).



The neighboring addresses are also formatted in this local space – they are for each of the 10 points in EPIOT(2,1), (/1, 2, 4, 5/, /2, 3, 5, 6/, /4, 5, 7, 8/, /5, 6, 8, 9/, /7, 8, 10, 11/, /8, 9, 11, 12/, /13, 14, 20, 21/, /15, 16, 22, 23/, /16, 17, 23, 24/, /18, 19, 25, 26/) – and sent to the transformer.

11 What information for the PSMILe and the Transformer?

This table details all the useful information for the different transformations.

Class of information	Information	PSMILe	Transformer
Grids			
	type	yes	yes
	pole/nopole	no	yes
	size of the halo	yes	no
Fields			
	Scalar/Vector	no	yes
	Mask/NoMask	yes	yes
Interpolations			
<i>(nearest neighbor(s))</i>	Number of neighbors	yes	yes
<i>(gaussian interpolation)</i>	variance	no	yes
<i>(bi-cubic)</i>	method (i.e. number of points to consider: 4 or 16)	yes	yes
<i>(conserv)</i>	normalization option	no	yes
Algebraic operations			
<i>(for each field)</i>	Symbolic name	yes	yes
<i>(for each field)</i>	source component	yes	yes
<i>(for each field)</i>	associated operand	yes	yes
<i>(for each field)</i>	associated scalar factor	yes	yes
<i>(for each field)</i>	associated mask	yes	yes
Collapse			
	collapsing dimension	yes	yes
	collapsing option	yes	yes

12 A first guess on the implementation of the interface PSMILe/Transformer on the Transformer side

In the transformer, the routine *PRISMTrs_loop()* insures the interface with the PSMILe. In this routine, an infinite loop receives the set of integers sent by the PSMILe layer of the different models and analyses it. Considering a type of transformations, the transformer performs a predefined sequence of actions that match with the PSMILe side.

13 A first guess on the implementation of the interface PSMILe/Transformer on the PSMILe side

This part details the first routines implemented to insure the exchange of information between the PSMILe and the transformer on the PSMILe side.

Five routines have already been implemented to simply interface the PSMILe and the transformer:

- 1- *PSMILe_trs_inform()* allows the PSMILe to send a set of integers (called control packet) to the transformer to inform this last entity that a sequence of tasks is required. Depending on the following routines, the content of this set of integers will differ.
- 2- *PSMILe_trs_set_grid()* sends the information relative to an intersected partition (EPIOS or EPIOT). The format of this information and the number of times *PSMILe_trs_set_grid* is called, will depend on the strategy chosen (see section 10.2).
- 3- *PSMILe_trs_put()* sends a field to the transformer if this field needs any non-local transformation.
- 4- *PSMILe_trs_interp()* sends the neighbors address to the transformer in order for him to be able to perform the interpolation. The format of this information and the number of times *PSMILe_trs_interp* is called, will depend on the strategy chosen (see section 10.2).
- 5- *PSMILe_trs_get()* requires the reception of a transformed field.

14 Description of the sequences of interactions between the PSMILe and the Transformer

This part gives a first guess on how the interaction between the PSMILe layer and the transformer. It is highly influence form the already implemented routines. Therefore, this proposal should not be considered as an optimized proposal. Some exchange like the exchange of neighbors before each interpolation would be made just once after optimization.

14.1 Set the transformer grids information

14.1.1 Description

This phase corresponds to set the ‘transformer grids’ information that will be used by the transformer to compute the weights and perform the interpolation.

14.1.2 Starting signal

This action needs to be requested once the intersections between the pairs of grids (more precisely the pairs of partitions) have been computed. Thus, it is required at the end of the definition phase through the PSMILe routine **PRISM_Enddef()**.

14.1.3 Sequence of tasks

PRISM_Enddef()

```

+ ...
+ PSMILe_Trns_set_grid( )
| + set the control packet
| + PSMILeTrns_inform( )    ? ? ?    PRISMTrs_loop( )
| |
| | + analyze the control packet
| | + select case PRISM_Set_grid_info
| | + PRISMTrs_Set_gnd_info( )
| | + MPI_Recv (latitudes)
| | + MPI_Recv (longitudes)
| | + set the grid information in
| | | the transformer structure
| | |
| | +
| | |
| + PSMILe_Trns_interp( )
| | + MPI_Send (neighbors)*    ? ? ?
| | +
| +
+ ...
+

```

* The number of MPI_Send depends on the strategy described in section 10.2

14.2 Send a field to the transformer

14.2.1 Description

This action corresponds to send the part of the field that belongs to the transformer grid. Therefore, before sending this part of field the internal routines of the PSMILe need to pre-condition this field.

If an interpolation is associated to this field, the PSMILe has then to send the neighboring points addresses to the transformer. Once the transformer has received the field and the neighboring address, he is able to perform the interpolation.

14.2.2 Starting signal

This action is activated when the user primitive **PRISM_Put()** is called.

14.2.3 Sequence of tasks (if an interpolation is required)

PRISM_Put()

```
+ ...
+ PSMILe_Trns_put( )
| + set the control packet
| + PSMILe_Trns_inform( )    ? ? ?    PRISMTrs_loop( )
| |
| | + analyze the control packet
| | + select case PRISM_Put
| | + PRISMTrs_Mind ( )
| + MPI_Send (field)*      ? ? ?    | | + MPI_Recv (field)
| +                          | | + check what transformation
|                          | | | has to be performed
|                          + + +
+ ...
+
```

* The number of MPI_Send depends on the strategy described in section 10.2

14.3 Request a field form the transformer

14.3.1 Description

A target model process requires a part of its field. The PSMILe manages the assembling of the field.

14.3.2 Starting signal

This action is required through the call of the user primitive PRISM_Get().

14.3.3 Sequence of tasks

PRISM_Get()

```
+ ...
+ PSMILe_Trns_get( )
| + set the control packet
| + PSMILe_Trns_inform( )    ? ? ?    PRISMTrs_loop( )
| |
| | + analyze the control packet
| | + select case PRISM_Get
| | + PRISMTrs_Target ( )
| | + Look for the field
| + MPI_Recv (field)*      ? ? ?    | | + MPI_Send (field)
| +                               + + +
+ ...
+
```

* The number of MPI_Recv depends on the strategy described in section 10.2