A parallel SCRIP interpolation library for OASIS

A. Piacentini, E. Maisonnave, G. Jonville, L. Coquart, S. Valcke

WN/CMGC/18/34

The SCRIP interpolation library defines the multiplication matrix used by OASIS to interpolate coupling fields between different model grids. This document summarizes the recent developments introduced in both OASIS PSMILE & SCRIP libraries to enhance computing performances in sequential and parallel mode (hybrid OpenMP/MPI). It must be emphasized that the implementation tried to preserve the results, then renounce to rationalize several algorithms (in particular, neighbors search restrictions that were found wrong in several cases) that necessarily need to be investigated in a second step. Performances (speedup of the calculations by 2 to 3 orders of magnitude) are presented for nearest neighbors, bicubic, bilinear and conservative interpolations.

Rationale

Since its first implementation in OASIS [1], the SCRIP [2] calculation of weights and addresses (W&A, interpolation multiplication matrix) is performed only by the MPI master process of the coupled models. In addition, for each interpolation, the operation is done by only one of the models involved in the OASIS coupling, a sender of one of the coupling fields related to the interpolation.

OASIS based coupled systems are usually exploited on supercomputers and their components are often parallelized with MPI or even MPI+OpenMP libraries. We took benefit of the existing MPI internal parallelization of the OASIS/MCT communication routines, and extended it to the SCRIP library. In a standard use, parallel models which include the OASIS/SCRIP library can now perform W&A calculation in parallel without any extra operation of the user. Notice that the SCRIP library implement a hybrid MPI+OpenMP parallelisation. It relies on the MPI parallel layout of the calling model but only enrols one MPI process per node[1]. The number of OpenMP threads per node is set by the dedicated environment variable `OASIS_OMP_NUM_THREADS`[2]. For optimum performances, it is recommended to set this variable to the number of cores of the node. Notice that, for most of the OpenMP implementations, the number of threads activated at run time is limited by the overall value set with the `OMP_NUM_THREADS` environment variable. If `OASIS_OMP_NUM_THREADS` is not set, it defaults to `OMP_NUM_THREADS`. On the contrary, if some of the coupled MPI+OpenMP hybrid models have to be run on less than the total number of cores per node (e.g. the number of core per sockets), then they should rely on dedicated environment variables and not on `OMP_NUM_THREADS`, which has to be large enough to grant resources to the finest grain parallelisation (notice that this consideration is not specific to the SCRIP library, but applies already to the case of two coupled models with different OpenMP layouts). If the `OMP_NUM_THREADS` environment variable is not set or it is set to 0, then all the logical cores are activated. If hyperthreading is activated this can lead to a serious deterioration of performances. In such a case it is mandatory to set the `OASIS_OMP_NUM_THREADS` environment

1 On some machine, the `I_MPI_WAIT_MODE` environment variable must be set to "`enable`" to save CPU time on the other MPI processes of the node

2 Of course, the OASIS libraries must also be compiled and linked with the appropriate OpenMP option

variable to the number of physical cores of the node. Users may want to distribute SCRIP calculations on more than `OMP_NUM_THREADS` threads. In this case, one would define this number in the environment variable `OASIS_OMP_NUM_THREADS`. For optimum performances, it is recommended to set this variable to the number of cores of the node.

The SCRIP interpolations can be filed in two groups: A) conservative (1st and 2nd order) and B) all others: bilinear, bicubic, distance weighted and Gaussian weighted nearest neighbor. B-type interpolations mainly follow the same procedure: for each unmasked target grid point, a distance is calculated with all (or a subset of) source grid points to determine which one are the closest and could participate to the weight calculation. It means that N independent calculations (with $N =$ target grid point number) can be scattered to different nodes of the machine without major communication bottleneck. On this outer loop, a MPI parallelisation is done on every first core of the nodes. In addition, to avoid memory bound duplication of source grid point arrays, OpenMP threads also parallelise the outer loop on target grid points and share the source grid point related variables [3,4]. After W&A calculation, results are copied on shared variables (OpenMP) and gathered on the master process of the model (`MPI_IRECV`).

W&A procedure is slightly different with A-type interpolations. Mesh contour intersections are calculated on both source and target grids. Consequently, the OpenMP/MPI hybrid parallelization is done on two outer loops (over source and target grid points). The search of neighbors potentially intersected can be restricted using the so-called "bin" technique. In a second step, the complementary nearest neighbor search (launched for target grid points without intersection with unmasked source grid points) is also parallelised with OpenMP. Preliminary rewriting were necessary to start the parallelisation of this interpolation. Details are given in Appendix 1.

Performances

A dedicated simplified coupled model (MPI+OpenMP //) was developed to be able to launch the W&A calculations on different grids (see table 1)

Source grid type/Resolution	LowRes (200Km)	HighRes (50Km)	UltraHighRes (10Km)
LR grid (logically rectangular)	ORCA1 (362x294)	ORCA025 (1442x1050)	ORCA12 (4322x3147)
D grid (Gaussian reduced)	T127 (24572)	T359 (181724)	T799 (843490)

Table 1: Grid resolution of the 3 sets (LR,HR and UHR) of W&A calculations

4 interpolations are tested in both directions (LR to D and D to LR grids): distance weighted of the 4 nearest neighbors, bilinear, bicubic and conservative first order (completed by nearest neighbor if no unmasked intersection is found). Results are validated to ensure results reproducibility at machine precision (due to different operation order). Restriction of neighboring search (bin) is used with conservative interpolation only (the accuracy of the interpolation wouldn't be guaranteed otherwise, see appendix 2). In this case, a number of 500 bins is prescribed.

On Figure 1, scalability of the W&A calculation is plotted (LR to D-type grid only) for 1,2,4,8,20 and 40 OpenMP threads and 1,2,4,8,16,32,64,128 (possibly 256) MPI tasks, i.e. a total of

1,2,4,8,20,40,80,160,320,640,1280,2560 and 5120 parallel tasks. 40 threads corresponds to the number of physical cores per node on the targeted machine (BULL Météo-France "`beaufix`"). LowRes scalability is not shown, considering that the time restitution (< 1sec) is highly dependent on external disturbances (network, OS ...) and always smaller than other contributions of interpolation initialisation (e.g. input file reading or W&A file writing).

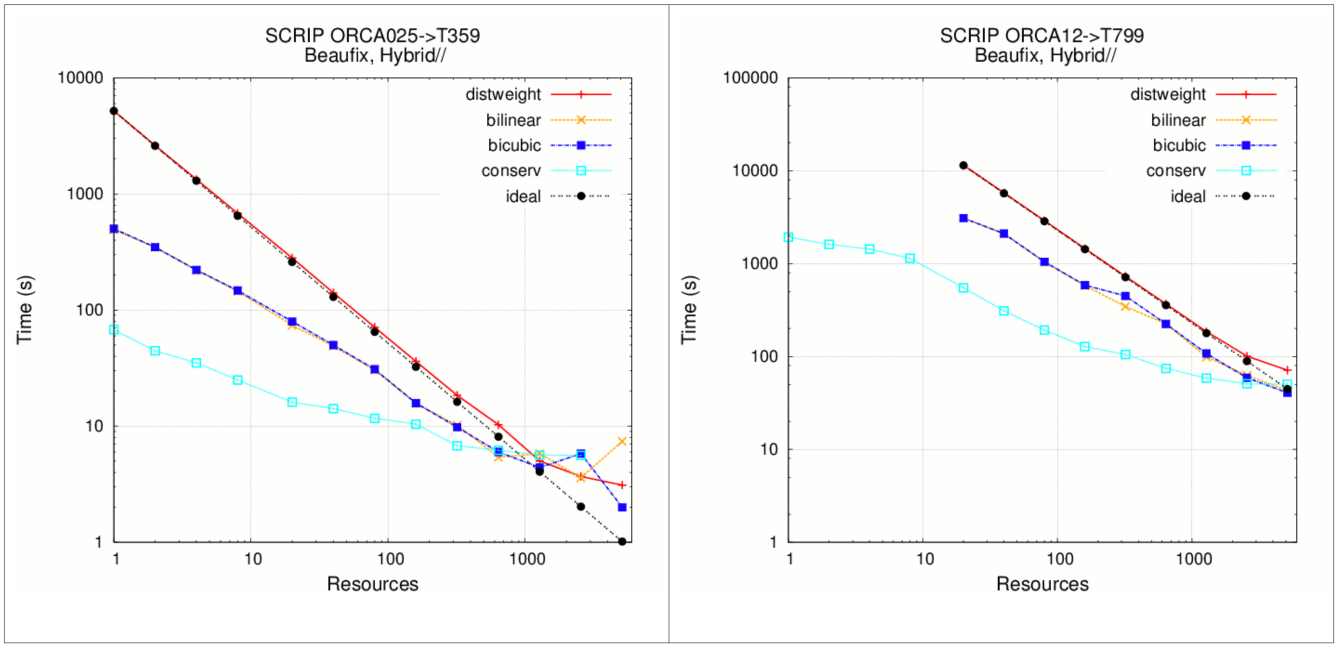


Figure 1: Scalability of W&A calculations for 4 SCRIP interpolations for HighRes (left) and UltraHighRes (right) configurations (logarithmic axes)

For B-type interpolations, the large number of possible neighbors treated (due to the lack of pre-selection by bin) slows down the calculation at low resources. This same constraint favors the good scaling until 1280 (2560) tasks at HR (UHR). A higher scalability would be achieved with a better load balancing, which is made difficult by the heterogeneity of the operations per target grid point (additional neighboring search if all source nearest neighbors masked, iterative loops ...)

The conservative (conserv) interpolation includes several operations: some loops are now parallel with OpenMP and MPI ("*sweeps*"). The additional neighboring search ("*`fracnnei`*") is OpenMP parallel only. Some sections are still sequential. These are the reasons why a diagnostic of the possible bottlenecks is more difficult. Some assumptions:

- at low resources, better performances are observed in comparison to B-type group, due to the bin functionality, and a new reduction of possible neighbors (improved bins)
- speed-up is slightly smaller: due to the reduced and variable number of possible neighbors to check (function of the position of the pole if non iso-area source grid), a large load imbalance affects performances since the beginning of the curve
- scalability limit is the same and again explained by the calculations heterogeneity per target grid point (additional neighboring search, pole projections at high latitudes, ...)

On figure 2, we compare the restitution time of the four W&A calculations before (version v3 of the code, in green) and after our optimization work (in red) with a parallel task number that maximizes the calculation speed. At any resolution, it leads to a reduction by 2 or 3 order of magnitude. This result makes possible the W&A computation at coupling time step frequency and open the door to further developments towards dynamical coupling, at least for global model resolution smaller than 50Km.

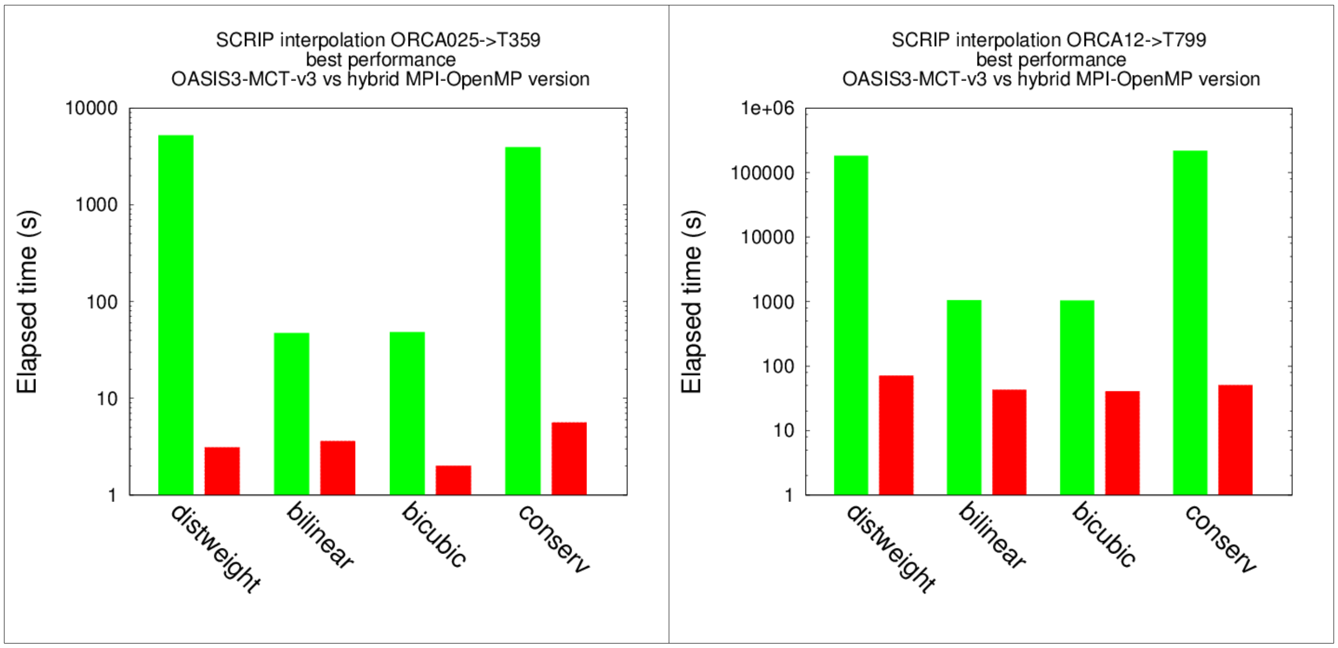


Figure 2: Restitution time of SCRIP W&A calculations on one core with the original OASIS3-MCT v3 library (green) and in parallel (red) on 2560 tasks (bilinear and conserv), 5120 tasks (bicubic and distweight) at HR (left) and 5120 tasks (all interpolations) at UHR (right). Configurations using 10240 tasks or more were not tested. (logarithmic elapsed time axis)

The best obtained performances of the parallel SCRIP library are far from the ideal scalability. Further improvements of the scalability require a thorough rewriting of the grid search algorithms which was beyond the scope of the present work and that would demand an extensive validation of the results. Nevertheless, it is important to remember that if the W&A calculations are performed from inside a coupled model running on N processing units, the current implementation allows, with no change of the results, to take advantage of all the allocated resources, while the sequential library, as implemented in OASIS-MCT3 v3 was wasting N-1 PU's.

Possible improvements

During this work, some problems were reported (see Appendix 3). In particular, we do not forget that D-type grids lead to erroneous results when a Lambert polar projection is used in conservative method calculations. This will have to be corrected, by, for instance, adding corners to mesh segment intersections at the high latitudes of the Gaussian grid, if the Lambert polar projection proves to be beneficial, which still needs to be shown.

Generally speaking, two kinds of limitations were found during this parallel implementation:

1- the heterogeneity of number of possible neighbors (neighboring search), sensible at the beginning of scaling curve for conservative interpolation
2- the heterogeneity of calculations themselves (additional neighboring search, iterative loop, polar projection ...) visible at the end of the curves

The reduction of neighboring search of distweight, bicubic and bilinear interpolation by bin seems inherited from the conservative method but could lead to erroneous results (see Appendix 2 for details). A re-thinking of the relevance and precision of this pre-selection for the other methods is mandatory before any new try of algorithm enhancement. This bin method should be compared with more popular algorithms like the k-nearest neighbors methods[3]. In any case, after this pre-selection of possible neighbors, the load of computations per target grid point would have to be evaluated in order to optimally scatter them on the MPI-OpenMP tasks.

Other improvements can be conjectured:

- An acceleration of the distance evaluation (replacing expensive trigonometric functions by simpler multiplications[4]).
- A simpler additional level of parallelism could be added by letting all models performing their calculations at the same time (preprocessing of the namcouple information)
- The inclusion in OASIS of the new MONTE-CARLO method, recently added in the SCRIP library, that could take benefit of the new GPU capabilities to become the fastest solution at high resolution

References

[1] Craig A., Valcke S., Coquart L., 2017: "Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3.0", Geoscientific Model Development, 10, pp. 3297-3308, doi:10.5194/gmd-10-3297-2017
[2] Jones, P., 1999: Conservative remapping: First-and second-order conservative remapping, Mon. Weather Rev., 127, 2204–2210
[3] Maisonnave, E., 2015: Interpolation weights calculation distributed with OpenMP® Working Note, **WN/CMGC/15/5**, SUC au CERFACS, URA CERFACS/CNRS No1875, France
[4] Coquart, L., Maisonnave, E. and Valcke, S., 2018: Using Open MP in OASIS3-MCT for the N-nearest-neighbor remapping, Technical report, TR-CMGC-18-19, CECI, Université de Toulouse, CNRS, CERFACS, France

3 i.e. https://github.com/darkskyapp/sphere-knn
4 i.e. http://jonisalonen.com/2014/computing-distance-between-coordinates-can-be-simple-and-fast/

Appendix 1
Recent changes in remap_conserv.F90, fracnnei.F90 and grids.f90 in view of their parallelisation
(A-type interpolations)

Preliminary analysis

The conservative remapping computation is the result of 7 algorithm steps

- The detection and correct masking of the repeated points on each grid. This treatment is optional and activated by the preprocessing TREAT_OVERLAY key (this was added in the OASIS version of the SCRIP).
- The first sweep on the destination grid cells detecting for each of them all the intersecting source grid cells, computing the integral contributions and the contribution to the surface fractions
- The second sweep on the source grid cells detecting for each of them all the intersecting destination grid cells, completing the integral contributions and the contribution to the surface fractions
- A specific treatment for the computation of the areas and centroids of polar cells
- The centroids computation
- Some sanity checks

If the fracnnei option (added in the OASIS version of the SCRIP) is activated

- The detection of valid destination points entirely surrounded by masked cells for which the nearest neighbour source cell is sought and its value used at the destination point.

With the initial configuration, on an Intel(R) Core(TM) i7-4930MX with 3.0GHz clock, the generation of the remapping from the orca 1/4 deg grid (orca025) to the T359 reduced gaussian grid takes more than 2700 seconds, 731 of which are in the detection of the overlapping points, 36 in the nearest neighbour final correction, a few fractions of second in the poles treatment, the centroids computations and the check, and almost 2000 seconds in the sweeps, with the second sweep being slightly faster than the first one.
The remapping in the other direction from T359 to orca025 takes 3800 seconds, 516 spent in the detection of the overlapping points, 293 in the nearest neighbour correction and almost 3000 in the sweeps, with the second sweeps taking almost twice as the first one.

The difference in the timing of the two remappings, especially in the ratio between the time spent in the first and second sweep can be explained by the fact that the detection of the cell side intersections and the computation of line integrals is restrained to destination and source cells with partly overlapping bounding boxes. The definition of the bounding boxes is quite rough and makes the restriction quite inefficient in the sweeps spanning the orca025 grid.

General considerations on the memory usage in OpenMP

In many passages, the original implementation of remap_conserv uses some local work array to store preselected values to be used for successive computations. This technique was most probably adopted for vector computers: storing operands in contiguous arrays was fundamental to obtain an effective

vectorisation. Memory was not at stake at the times.
When OpenMP is used for the parallelisation of the grid sweep loops, all work memory whose content depends on the index of the loop has to be replicated on each thread (private arrays).
The memory occupation on a node will result from the product of the number of OpenMP threads and the size of private work arrays. Foreseeing an important increase in grids resolution and in the number of cores per node of future processors, the product could easily saturate the processor memory. It is therefore of the highest importance to reduce at a minimum the private work arrays inside the sweep loops.

Treatment of the repeated points on a grid (TREAT_OVERLAY)

If oasis has been compiled with the preprocess key -DTREAT_OVERLAY, before entering the conservative remapping both grids are processed in order to detect overlapping points due to periodical or polar closures; this was added in the OASIS version of the SCRIP. On the destination grid we enforce that only the point with lowest index is active and the replicas are masked out. For the source grid we need to build a mapping associating the index of the replicated points to the point with lowest index.

The original version used to scan the whole grid for every point to be checked. The coincidence check has a tolerance of the order of the precision (fortran primitive epsilon(1.). The complexity is obviously O(n^2).
The new version sorts the grid coordinates calling a modified version of the standard heapsort (lines 242 and 281 of remap_conserv.f90). The code has been adapted from the Quantum Espresso fortran 90 implementation in order to compute only an index permutation vector, to use the “latitude first, then longitude” sorting criterion and to evaluate equality with a tolerance of epsilon(1.). It has a complexity of O(n log(n)).

Once the coordinates are sorted, equality has only to be checked amongst consecutive entries.

This approach cuts the time for the overlap check in the orca025 to t359 remapping down from 731 seconds to 0.4.

Since the parallelisation would require extra storage to avoid conflicts while modifying the grid mask, we decided to keep this treatment sequential.

Indirect addressing based searches

The original implementation of the routine finding the intersections between the sides of a grid cell and the sides of the preselected cells (by bins and bounding boxes intersection criteria) from the other grid worked on contiguous arrays containing the addresses and the corner coordinates of the cells.
For each cell of one grid, in a first loop on the other grid (restricted by bins), the selected (by bounding box intersection) cells are counted (num_srch_cells) then the address and corner coordinates work storages are allocated and, in a second loop, the addresses and the corner coordinates are copied into the work storage (gather1: and gather2: loops).
Since num_srch_cells can be high if the resolutions of the two grids are at least locally very different, the size of the working storage is not negligible. For the OpenMP parallelisation of the sweep loops, the storage should be duplicated (private) on each thread and this could easily lead to memory saturation.

In the new implementation, only the cell addresses are stored and the coordinates of the corners are

directly taken from the original grid storage that will be shared (only one instance in memory) by all the OpenMP threads.
In the intersection subroutine, instead of working on the srch_corner_lon and srch_corner_lat contiguous arrays, a global address is associated to the local search cell number (line 2154 of remap_conserv.f90) and the coordinates from grid_corner_lon and grid_corner_lat at the global address are used.

Optimised strategy for storing the search cell addresses

In order to preselect all the cells of one grid whose bounding box crosses the bounding box of a given cell on the other grid, we need a double loop on the grid: the first one to count the eligible cells and allocate the work arrays and the second one to actually store the values.

In the original implementation, the costly test on the coordinates of the bounding boxes was performed only once and the result stored in a logical array (srch_mask) of the size of the scanned grid to be reused in the test of the second loop. Once again, this array should be duplicated (private) for the OpenMP parallelisation.

In the new implementation, we perform the test on the coordinates in the second loop as well (pick1: and pick2: loops at lines 504 and 1126 of remap_conserv.f90), but we restrain the loop range to the minimal range of addresses that encompasses the selected cells in the first loop (loops gather1: and gather2: in particular at lines 486-487 and 1108-1109 of remap_conserv.f90).
Another slight optimisation, at least for the intel compiler, comes from the use of nested IF statements instead of a single condition with .AND. Since the fortran standard does not enforce the left to right evaluation of the elements of a combined logical expression, at high optimisation levels (-O2 and -O3), all the tests are performed even if one is failing (cf. the classical situation IF (x>0 .AND. sqrt(x)<a) leading to floating point exception). Nesting the IF statements grants that if the first test fails, the following are not evaluated. This optimisation is useful if one of the test has a larger probability to fail (better if in the outermost IF) or if one of the test is costlier (better if in the innermost IF). Since grids with regular resolutions have more longitude that latitude subdivisions, the test on the longitude should be in the outer IF's (because it has a larger probability to fail), yet, the introduction of a more complex test on longitude taking periodicity into account, suggested to take it to the innermost position (lines 480, 506, 1102, 1128 of remap_conserv.f90).

OpenMP compliant link storage

Since more than one side of a grid cell can cross the same cell on the other grid, more than one line integral can provide a contribution to the same remapping link (same destination and source address pair).
In the original version, a new contribution is stored as a new link only if a scan on the already stored links finds no corresponding address pair. Conversely, if the address pair is found, the new contribution is added to the weight of the link.

This approach is not viable for parallelisation since the order of the computations is unpredictable and rush conditions are easily encountered.

In the new version, every line integral contribution is stored as a new link. This work can be parallelised and at the end of the parallel section, the single thread storages can be gathered into a storage per MPI process and these storages finally gathered on the master process.

The remap weight storage is then sorted (as usual) by the sort_add routine that uses a heapsort on the criterion “destination address first, the source address”. After the sorting all the contributions to the same address pair are stored in contiguous positions.

We introduce the uniq_add (lines 576 and following of scrip.F90) that cumulates by sum the duplicated links and reduces the size of the remapping storage. Instead of explicitly copying and shifting the links we use the much more efficient pack fortran primitive.

Optimised search of dangling links (known as Valid-masked-masked points)

If all the grid cells intersected by the sides of a non masked destination grid cell are masked, the interpolation will not provide a valid result for the destination point. This situation is indicated as a valid-masked-masked situation. The value of the nearest neighbour non-masked cell centre on the source grid will be assigned to the destination grid cell, if the user decides to activate this option added in OASIS.

The first step is to detect the cells on the destination grid that are not listed as destinations of any link in the conservative remapping.

In the original version, the fracnnei_vmm routine implemented exactly this logic: a loop on the destination grid and for every cell a scan of the link storage searching for at least a matching destination address. If a link is found, the cell is flagged out by the logical array lnnei selecting the points for which the next neighbour search is needed. The complexity of the search increases as the product of the grid size and of the number of active links.

In the new version a single scan of the link storage is needed to flag out the destination links addresses from the lnnei array.

The computing time (for the whole next neighbour treatment, including the neighbour searches) for the T359 to the orca025 grid falls from 293 to 5.9 seconds.

Optimised version of the nearest neighbour search fracnnei.F90

To find the nearest neighbour, the source grid is scanned and the angular distance (greater circle) between the non masked cell centres and the destination cell centre is computed to find the minimum distance.
In the original version, the additional links are stored in temporary arrays that are added to the link storage only afterwards.
In the new version, the link storage size is increased beforehand (line 156 and following of fracnnei.F90) and the additional links are immediately stored there with no need of extra work memory.

In the original version, the destination grid is swept with a loop on the whole grid, skipping the points for which the nearest neighbour computation is not necessary (lnnei array acting as a mask), while in the new version the loop is on the number of additional links and indirect addressing is used (line 241 of fracnnei.F90). In this way this loop can be easily parallelised by OpenMP and will be naturally load balanced.

As a further optimization, the angular coordinates (using trigonometric functions) on the source grid are computed only once, and not recomputed for every additional link as in the original version (line

163 and following of fracnnei.F90).

Conclusions on the parallelisation strategy

After the rewriting of remap_conserv and fracnnei, the time spent for the detection of the repeated points, for the centroid computation, the checks, the sorting of the links and the condensation of the repeated links is low enough to keep these steps sequential.

In the fracnnei routine the loop on the additional links for the search of the non masked nearest neighbour can be easily and effectively parallelised by OpenMP. Since the number of additional links is of several orders of magnitude less than the size of the grids (especially if the resolution of the two grids is close), the overhead of communications for the hybrid parallelisation will not be paid back.

Most of the computing time is spent in the sweep loops and as they have been rewritten they can easily be parallelised with the hybrid MPI+OpenMP approach used for the other remappings (explicit splitting of the outermost loop among the MPI processes and, on each MPI process, among the OpenMP threads; no need, in this case, to precompute the distribution of the masked cells;partial private link storage on each thread, gathered afterwards in a shared storage on each process and finally gathered on the master MPI process).

The parallel load balancing depends on now many intersection and line integral computations are necessary for each of the assigned destination grid cells.
The number of the tentative cell intersections to be checked depends on the effectiveness of the binning and bounding box overlapping restrictions. In the original version, the bounding boxes turn out to be excessively large in many cases and seriously reduce the effect of the restrictions. This point deserves a thorough analysis (see next section).
The number of line integral computations depends on the difference between the local resolution of the two grids. In the case of the mapping between an oceanic grid with polar (north) refinement and a reduced atmospheric grid with low resolution at high latitudes, the cells at high latitudes will require more computations. For this reason, it could be interesting to check the impact on performances of the choice of a round-robin distribution of the loops against a contiguous one, knowing that the latter is much simpler and requires one less level of indirect addressing.

Analysis of the bounding box definitions in grids.f90

The efficiency of both restrictions of the computations in the grid sweeps entirely relies on a correct and coherent definition of the bounding boxes of the source and destination grid cells.

For the conservative remapping, the grid cell corners are a mandatory input, therefore the bounding box are defined by the coordinates of the most external corners. In principle, a lon/lat rectangle encompassing all the corners is defined.
In practice, however, scrip moves the longitude of all the corners, one by one, in the $[0,2\pi]$ interval, before computing the bounding boxes. For this reason, cells that span across the longitude periodicity threshold will be stored with some corners close to 0 and some close to 2π.
Since the east boundary of the cell is assigned to the corner with the smallest longitude (and vice-versa for the west boundary), in such cases, the corners are swapped and therefore the bounding box meaningless (they take the complement of the real longitude bounding segment).
Moreover, since the tests on bounding box boundary longitudes are implemented simply as “less than” or “greater then”, they cannot account for periodicity.

Also, bounding boxes spanning more than π rad in longitude (officially not supported) are automatically extended to [0,2π] and they do not intervene in the sweep restrictions.

A specific attention is paid to polar cells. Scrip tries to detect them by the consideration that the north and south boundaries are on the opposite sides of the pole. If the centre of the cell is close enough to the pole both boundaries will have a smaller latitude than the centre.
In such a case, on a quite arbitrary base, scrip extends up to π (90N) the boxes whose larger latitude is smaller than the latitude of the centre and down to -π (90S) the boxes whose smaller latitude is larger than the latitude of the centre. Not only this test does not detect all the possible pole crossing cases, but it acts on some specific orca grid twisted boxes (as stored in old grids.nc files) for which the centre does not fall inside the corners, irrespectively of the position of the cell. Some cells close to the Antarctic coast are extended up to 90N. Notice that these cells spanning almost the whole latitude range, seriously impact on the efficiency of the binning.

New strategy for the bounding boxes

We suggest a new strategy for a better definition of the bounding boxes.
It relies on the hypotheses that all the corners in grids.nc are entered with the same periodicity module than the corresponding centre.
As an example, if the centre of a grid with periodicity at 180 is at -179 and the corners are at two degrees on each side, they have to be entered as -181 and -177, and not as 179 and -177.
Similarly, if the centre of a grid with periodicity at 0 is at 1 and the corners are at two degrees on each side, they have to be entered as -1 and 3, and not as 359 and 3.

This is required in the OASIS documentation, but some legacy files do not respect the convention, therefore an initial check is performed: if the corners longitude range extends beyond π, it means that they have not been entered following the convention (the test holds because real cells extending on more than 180 deg. are not allowed in OASIS). In such a case, we'll impose that all the corners will lay in the same range as the centre (lines 370 and following of grids.f90).

Once we know that all the corners have been entered in the same range as the centres, we can move all the centres inside the [0,2π] interval, moving all the corners of each cell together with the centre (lines 412 and following of grids.f90).
In the example of a centre in -179 and corners at 179 and -177, the first check corrects the left corner as -181, then the whole cell is moved by +360 to [179,183].

The bounding box can now be safely computed using the minimum and maximum coordinates of the corners (lines 635 and following of grids.f90)

By construction, since all the centres lay in the [0,2π] interval, at least one boundary of the bounding box falls inside the [0,2π] interval.
In order to specialize the longitudes comparison to different periodicity configurations, we class the grid cells in three classes:
class 0 contains the cells with bounding box fitting entirely in [0,2π].
class -1 contains the cells with the west boundary smaller than 0 and the east boundary in [0,2π].
class 1 contains the cells with the west boundary in [0,2π] and east boundary larger than 2π
(lines 794 and following of grids.f90).

We introduced a partial fix for the excessive latitude extension of the bounding boxes (lines 800 and

following of grids.f90).
If the centre of the grid is to the north of the north boundary of the bounding box, then the box is extended north up to the centre if the centre is south of 60N and up to the north pole if the centre is at high latitudes.
Conversely, if the centre of the grid is to the south of the south boundary of the bounding box, then the box is extended south up to the centre if the centre is north of 60S and up to the south pole if the centre is at high southern latitudes.
This correction fixes the problem of artificially large bounding boxes, yet it does not address in a satisfactory way the detection of cells encompassing the poles.
Since some of such cells result in large longitude extensions (the corners being on the opposite side of the poles) we preserve the correction extending to [0,2π] the longitude of bounding box, moreover, for high latitudes we extend the box up to the corresponding pole. The test on "the longitude size being larger than π has been relaxed to take into account the rounding errors while converting degrees to radians and while translating the corners across periodicity (lines 702 and following of grids.f90).

Once the bounding boxes have been built we are free to move the corners longitudes inside the [0,2π] interval to stay closer to the rounding errors of the old strategy, yet this transformation is absolutely not necessary (lines 846 and following of grids.f90).

New bounding box intersection test in remap_conserv.F90

With the adoption of the new storage for the bounding boxes we need a new test strategy on longitude intersection, taking the periodicity into account.
The old strategy was to compare the longitude of the west boundary (stored in position 3) and the east one (stored in position 4) of the bounding boxes for pairs of destination and source grid cells.

The four cases for intersection are represented in Figure 1 and they can all be detected by the logical check that "the west boundary of the source cell is west of the east boundary of the destination cell AND the east boundary of the source cell is east of the west boundary of the destination cell".

When both the destination cell (red in the figure) and the source cell (green in the figure) fall inside the [0,2π] interval (both are of kind 0), the longitudes can be directly compared and the test will read "the longitude of the west boundary (stored in position 3) of the source cell is smaller or equal than the longitude of the east boundary (stored in position 4) of the destination cell AND the longitude of the east boundary (stored in position 4) of the source cell is greater or equal than the longitude of the west boundary (stored in position 3) of the destination cell"

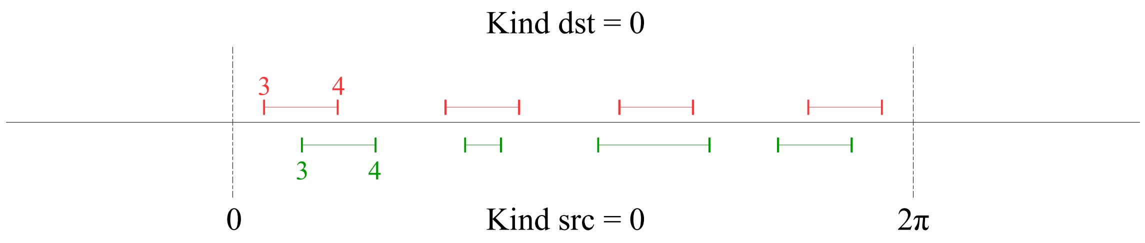


Figure 1

Cells crossing the periodicity threshold necessarily cross each other since they have at least the threshold value in common. Therefore, if the destination cell crosses the periodicity threshold,(kind= –

1 or kind = 1) a specific test has to be performed only with source boxes falling inside [0,2π] (kind=0)

In the case of a destination cell with the west corner stored with a negative longitude, the four cases of intersection are represented on the left side of Figure 2. Crossing with cells close to 0 is checked with a single standard test in [0,2π] on the east side of the destination cell whose longitude has to be larger than the longitude of the west side of the source cell. Crossing with cells close to 2π, represented by their modulo image in Figure 2, is checked with a single periodicity test on the west side of the destination cell whose longitude, incremented by 2π has to be smaller than the longitude of the east side of the source cell. Conversely it is easy to devise the tests for the three remaining configurations: destination cell kind = 1 and source cell kind = 0 (right side of Figure 2), destination cell kind = 0 and source cell kind = –1 (left side of Figure 3), destination cell kind = 0 and source cell kind = 1 (right side of Figure 3).

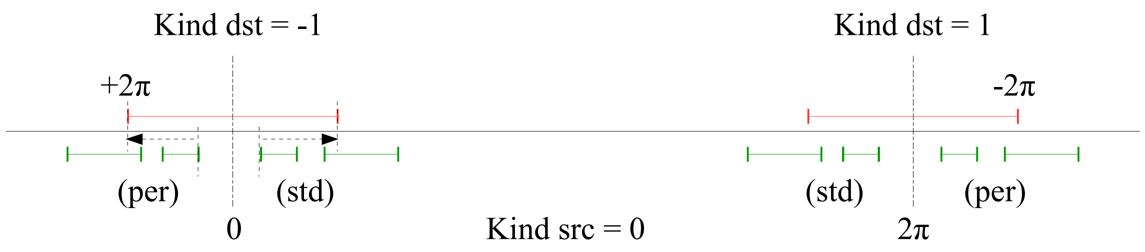


Figure 2

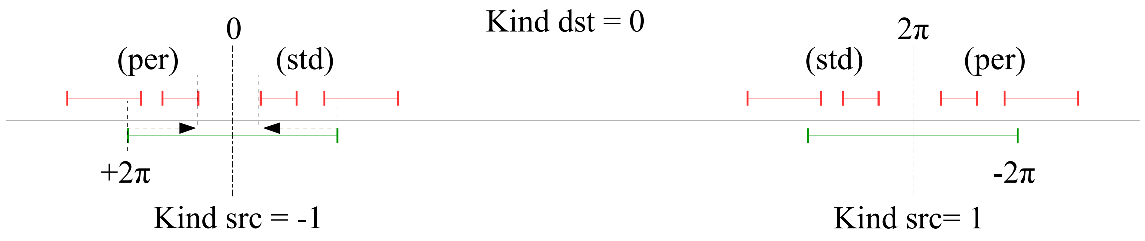


Figure 3

Impact on the performances

The adoption of the new bounding box definition has a strong impact on the restriction of the computations in the remap_conserv grid sweeps.
With the old strategy in many cases, the number of preselected cells for the computation of side intersections (variable num_srch_cells) is excessively high. This has three negative consequences: excessive computing time, excessive work memory storage, unbalanced computational load while parallelising the sweeps, since the high values of num_srch_cells are not equally distributed and cannot be computed beforehand.
As shown on Figure 4, where we compare the number of preselected search cells (num_srch_cells) for every destination grid cell in the two sweeps (linear y scale in the top panes, logarithmic y scale in the bottom panes, red crosses for the old bounding box definition, green crosses for the new ones), not only the number drops of roughly two order of magnitudes, but the distribution is more uniform. Always notice the increase on preselected cells in the polar regions (the leftmost and rightmost extremities of

the plot) due to large cells in the T359 grid.

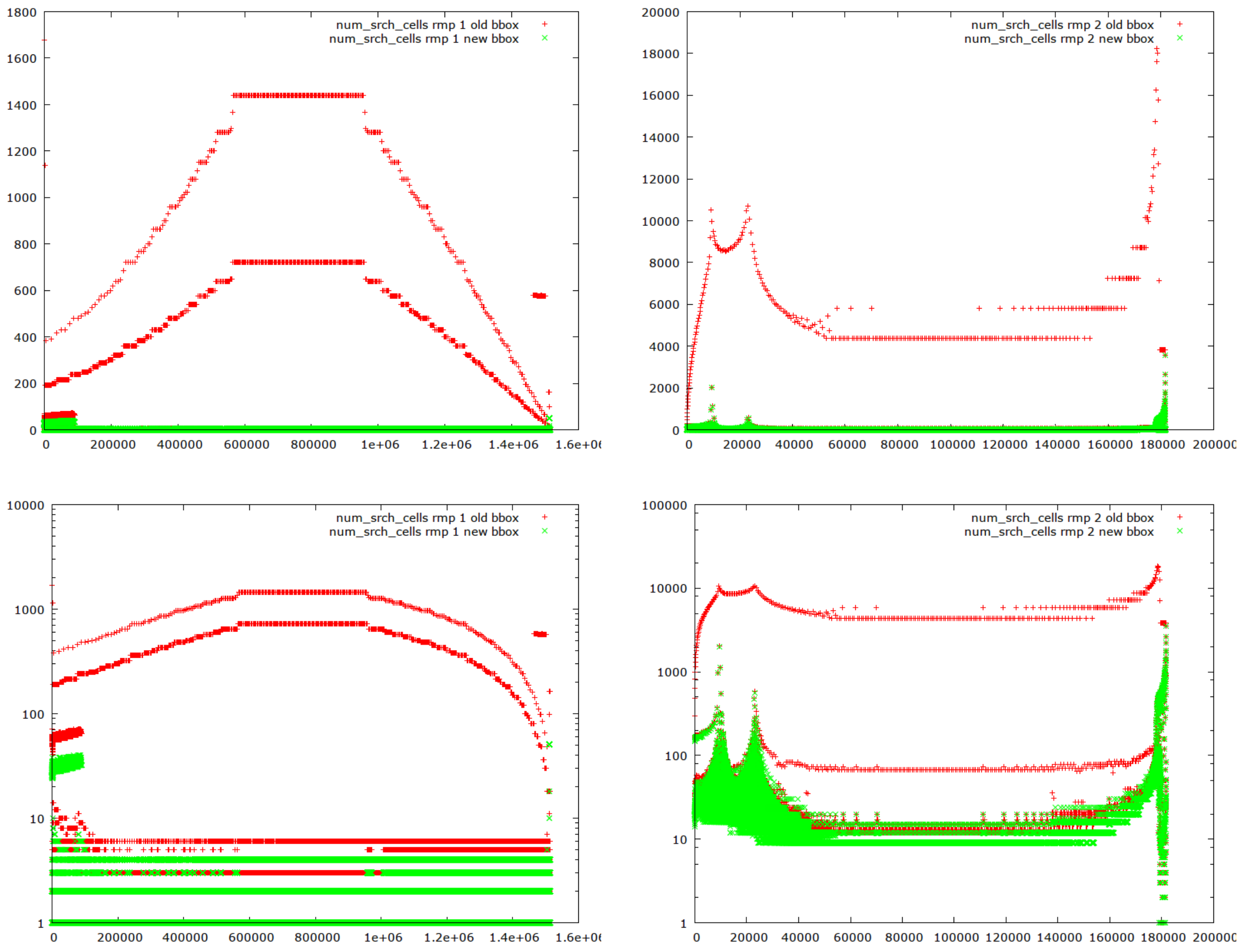


Figure 4

Another important effect is that the first level of computation reduction is based on the latitude binning of the grids cells. Since a cell is assigned to every bin intersected by the cell bounding box, with the old strategy some cells belong to all bins, pushing the associated min and max bin addresses to too large values, with a drastic reduction of the binning effectiveness.

For instance, on an Intel(R) Core(TM) i7-4930MX with 3.0GHz clock, the generation of the remapping from orca025 to T359 with the old bounding box definition takes 851 seconds without binning but still 840 seconds with 500 latitude bins (273 in the first sweep and 565 in the second), while with the new bounding box definition it takes 821 seconds without binning and the time drops to 40 (15 in the first sweep and 23 in the second) seconds with 500 bins.

Appendix 2
Bins definition and determination

In order to restrict the search loops in the remapping matrix computation, SCRIP introduced a latitude binning strategy. Bins are meant to associate index spans in both grids representation to latitude bands. With different implementations for the different remappings, they can therefore be used to target subsets of the grids before performing further matching tests or brute force searches.
In recent SCRIP versions a rectangular binning in both latitude and longitude has been implemented as an option.

The binning has been most probably originally designed and implemented for the conservative remapping on logically rectangular grids (and in such configuration they are robust and quite effective) and its use has been further extended to other remappings in SCRIP (nearest neighbour, bilinear, bicubic) and in the SCRIP extensions for OASIS (reduced grids, minimal information on grids descriptions).

Let's point out an important remark having several implications in the following: neither SCRIP, nor its wrapping in OASIS impose any constraint or convention in the way the grid cell are numbered and therefore sorted in the input netcdf files grids.nc and similar.
Each grid is therefore internally represented as a collection of 1D arrays (at least one for the latitude and one for the longitude of the grid cell centres) in the same order as used in files: no hypothesis is made on a relation between the storage index progression and the geographical coordinates growth.
We'll see later that in some specific cases some computations rely heavily on this assumption, leading to some potential error, but we must stress here, that even in the safe cases, the effectiveness of the latitude binning depends on the order of the storage: since a bin is just identified by the min and the max index in the grid storage granting that all the cells falling in the latitude bin are included in the index range, the restriction on the range is effective only if the grids are stored with the longitude increasing first. Paradoxically, for the opposite storage strategy (latitude increasing first), every bin would be associated to the whole index range (but a few elements), with no effect on the optimization.

The original latitude bins are internally stored using three 2D vectors:
`bin_lats(1,n)` and `bin_lats(2,n)` respectively are the minimum and the maximum latitude (in radians) covered by bin n.
Notice that the binning splits the global [-π/2,π/2] interval even for limited area grids.
For NBINS bins, bin n covers the [-π/2+(n-1)*π/NBINS, -π/2+n*π/NBINS] interval.

`bin_addr1(1,n)` and `bin_addr1(2,n)` respectively are the minimum and the maximum index in the source grid storage granting that all the cells associated to bin n are certainly spanned if ranging from the minimum to the maximum index. Notice, once again, that not all the cells between `bin_addr1(1,n)` and `bin_addr1(2,n)` belong to bin n: we can only say that all the cells associated to bin n fall between these two indices. As a consequence, even if the latitude bins are a non overlapping split (but for the boundaries) of the Earth, the index bins can largely overlap depending on the grid storage order.

`bin_addr2(1,n)` and `bin_addr2(2,n)` store the same information for the destination grid.

The association between cells and bins is based on the cell bounding boxes: a cell is associated to a bin

if its bounding box intersect the bin i.e. a cell is associated to all the bins covering the latitude extent of its bounding box. Since the bins always cover the full [-π/2,π/2] interval, every cell is associated to at least a bin.
Notice that a single cell is potentially associated to more than 1 bin even if it centre falls necessary in a single bin (but for the boundaries) because its bounding box can cross the bin boundary (or even several boundaries if the bin extent is smaller than the local latitude resolution).

The accuracy of the bin associated therefore depends on the accuracy of the bounding box determination.
If corners are available, the bounding box is based on the minval and maxval of the corners coordinates. This is a quite robust definition, coherent with the way SCRIP parametrize the cell sides (i.e. as a segment of great circle passing by two corners. Notice that this parametrization fails for curvilinear grids with high curvature, for which a curved side can extend outside the rectangular box encompassing the corners).
Unfortunately, for the usage of SCRIP in OASIS, the corners are read in from the coordinate file only in case of conservative remapping.
For the other remappings, the corners are neglected even if they are stored in the file and the bounding box is estimated by the relative position of the grid centres. The algorithm relies on the implicit hypothesis that the grid is Cartesian and stored with longitude increasing first, therefore it is only valid for LR global grids (with a tentative correction to catch up for limited area grids) with limited rotation of the cells w.r.t. to the i=lon and j=lat axes and requires the convention to sort the grid stored in grids.nc by increasing longitude first and latitude later.
In the current implementation, this definition introduces an error for other grids that can lead to wrong or incomplete binning.

In the case of bilinear or bicubic remappings from source global reduced grids (grids for which the number of longitude subdivisions varies with the latitude), a specific bin decomposition has been introduced. Notice that this technique does not require the bounding boxes since it relies on the consideration that the reduced grids are oriented in the latitude and longitude directions and therefore that a cell coincides with its bounding box).
Yet, once again, it requires a convention on how to sort the grid stored in grids.nc: by increasing longitude first and by decreasing latitude. Independently of the number of bins indicated by the user, the final number of bins coincide with the number of latitude circles in the grid (minus one, to be precise).
This assumption conceptually differs from the original definition for two important features: the number of bins depends on the grid and it is not the same for both source and destination grids and the latitude extent of each bin is potentially different (if the latitude deltas on the grid are not constant, as it happens for Gauss grids).
It defines one 2D vector and one 4D vector:
`bin_lats_r(1,n)` and `bin_lats_r(2,n)` respectively are the latitude of the centre of the cells on row n and and of the centre of the cells on row n+1. Remember that because of the specific storage convention, row n+1 is south of row n and therefore `bin_lats_r(2,n)` is smaller than `bin_lats_r(1,n)`. Notice that the bins do not span the whole Earth since they skip the very high latitudes beyond the grid centres of the first and last latitude circle.
`bin_addr1_r(1,n)` and `bin_addr1_r(2,n)` respectively are the first and last index in the source grid storage for the latitude circle n.
`bin_addr1_r(3,n)` and `bin_addr1_r(4,n)` respectively are the first and last index in the source grid storage for the latitude circle n+1 (for the last value N of n, `bin_addr1_r(3,N)` and

bin_addr1_r(4,N) are set to the values of bin_addr1_r(1,N-1) and bin_addr1_r(2,N-1) respectively. A doubt arises here, it it should have been either bin_addr1_r(1,N) and bin_addr1_r(2,N) or bin_addr1_r(3,N-1) and bin_addr1_r(4,N-1))

Bins utilisation in different remaps

conserv (N.B. only robust case for all grids)

Aim: to restrain the search interval for the bounding box intersections during grid sweeps
Implementation: starting from point grid1_add on grid 1, loop on all the bins and if the bounding box of grid1_add intersects bin n for grid 1 (Notice the use of the bins for the same grid the point belongs to), update the min and max addresses for the loop on grid 2 to be sure to include index bin_addr2(1,n) on the min side and index bin_addr2(2,n) on the max side in the search loops range.
Therefore if grid1_add bounding box crosses more than one bin, the search loop will span the ensemble of the representations of these bins on grid 2.
This implementation is robust because the same numbering of the bins has been used for both grids, the bin splitting covers the whole Earth and their representation on both grids are used.
Potential extensions: use of the bins to restrict the search for overlap points in TREAT_OVERLAY and for the search of the nearest neighbour in case of missing weights (fault of grid overlap)

bilinear and bicubic

Aim: to restrain the search of the source grid cell whose bounding box contains the destination grid cell centre
Implementation: direct comparison of plat and plon (coordinates of the destination grid cell centre) and the bin_lats bounds of the source grid.
Caveat: if a grid point has at least one neighour from the original bilinear/bicubic stencil masked, it is handled, in the current implementation, by a distance weighted sum of the 4 nearest neighbour values. The nearest neighbour search is restricted by the same binning, leading to a strong dependency of the relative position of the selected points and the size of the bins.
Note: if a destination point falls outside the source grid bin coverage the search loop is not even performed and the complementary nearest-neighbour search on the whole grid is activated.

bilin and bicubic for reduced source grids

Aim: to restrain the search of the source grid cell whose centre is the nearest to the destination grid centre
Implementation: direct comparison of plat and plon (coordinates of the destination grid cell centre) and the bin_lats_r bounds of the source grid. Detect grid cells falling beyond the grid centres of the first and last latitude circle. For the points too far north, the min_add for the source grid search is set to 0, for the points too far south, the max_add for the source grid search is set to nx+1, nx being the size of the grid. In both cases, the search if flagged as invalid, but the information on min_add and max_add is later used for the alternative nearest neighbour interpolation.
For destination grid cells falling inside the latitude source grid extent, the bin indices bounds are used to restrain the search for the search on the longitude coordinate.
Caveat: notice the specific convention for storing reduced grids: the latitude is decreasing. If the convention is not respected, the bins definition will not fail but the search grid will systematically reject

points and the interpolation will become a 4 points distance weighted nearest neighbour.

nearest neighbour (dist or gaus)

Aim: to restrain the number of evaluation of the angular distance between the destination cell centre and the source grid cell centres (costly because trigonometric)
Implementation: direct comparison of `plat` and `plon` (coordinates of the destination grid cell centre) and the `bin_lats` bounds. If the destination grid cell centre belongs to bin n, uses bins n-1, n, n+1 to select the range (on the source grid) for the angular distance computation.
Caveat: for a large number of bins when looking for a large number of neighbours, taking three bins could not be enough and the search could fail.
Moreover, the search simultaneously looks for the nearest non masked point to be used if the nearest neighbours (in an absolute sense) are all masked. For very high resolution grids, this point could be outside the 3 bins.
On the other way, for unstructured source grids without any sorting of the coordinates storage, the indices associated to the bins could potentially include the whole grid with no real reduction of the source grid search.
Potential extensions: introduce a quick but rougher distance estimate in order to discard source grid points from the search with a cut-off test instead of trying to limit the search range by bins.

Appendix 3
Bi-linear/cubic remapping bug

In the case of bilinear and bicubic remappings, grids cells are not defined as tiles surrounding the grid centre, but rather as the logical rectangles connecting four neighbouring grid centres.
Cells take their number from the logically south western corner (better stated as the corner first encountered in the centres array). This is the point counted as 1 in the cell. Point 2 is the logically south east corner (it is obtained by mapping back the first corner address to the *i* and *j* grid indexes and recomputing the address for *i+1* and *j*) and it is also noted as point e (east of the first corner). Point 3 is the logically north east corner (*i+1, j+1*), also noted as point ne (north-east of the first corner). Point 4 is the logically north west corner (*i, j+1*), also noted as point n (north of the first corner).

Because of the periodicity, since all the centres coordinates have been translated into the [0,2π] interval, the longitude of the east and north-east points can be numerically smaller than the longitude of the reference point 1, if this one is close to 2π rad of longitude.

Once again, the bounding boxes being defined by the minimum and maximum numerical values of the longitudes, for the grids encompassing the periodicity threshold, the bounding box longitude boundaries are meaningless and for such grids, the bounding box only limits the latitude.
Notice that for the cells not crossing the periodicity threshold, if the grid is not twisted nor rotated, the bounding box coincides with the cell, otherwise it defines the minimum longitude-latitude rectangle containing the cell.

Given a destination point of coordinates plon, plat, the source grid is searched to find the cell containing the destination point.

The search is in two steps: first only the cells whose bounding box contains the destination point are selected, then for these cells a cross product based criterion is applied.

The bounding box criterion is based on the plain numerical comparison of coordinates:
bounding box min longitude ≤ destination longitude ≤ bounding box max longitude
bounding box min latitude ≤ destination latitude ≤ bounding box max latitude

If the source grid is not twisted nor rotated only one box contains the destination point, otherwise it is contained in the overlapping bounding boxes of the cells neighbouring the true source cell.

Notice that, by construction, the cells crossing the periodicity threshold are searched for all destination points with latitude falling in the latitude bounds of the cell.

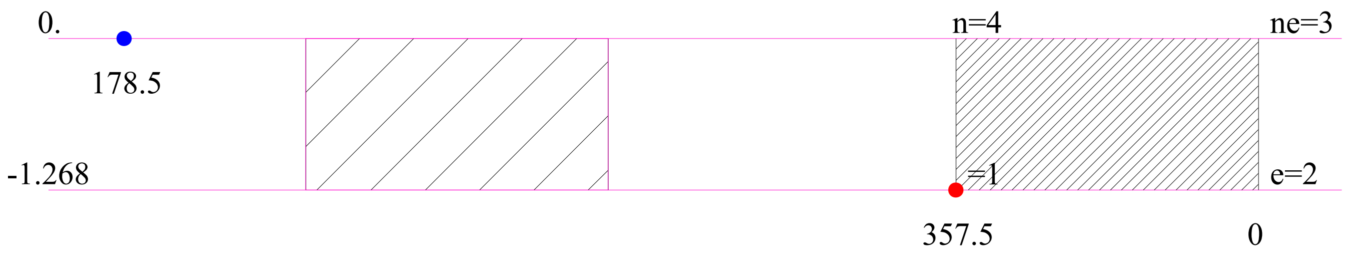
The cross product criterion aims to assess if the destination point lays always to the right or always to the left of all the sides of the cell. It translates, mathematically, by checking that the cross product between the vector corresponding to a cell side and the vector extending from the beginning of the cell side to the destination point has the same sign for all the cell sides.

The way the sign comparison is implemented has a flaw: it tests if the product of the last and the current evaluation of the cross-product is not strictly < 0. If the two first cross product have the same sign and the third one 0, regardless of the sign of the fourth and last cross-product, the product of the third and fourth results is 0, therefore not strictly < 0 and the cell is accepted.

This situation arises if the destination point is aligned with the logically north side of the cell.

Even if the bggd and nogt grid resolutions are different, this situation happens in the case of a destination point from nogt (ORCA1) on the equator and the grid cells from bggd (LMDZ) just under the equator.

Yet for the source grid cells away from the periodicity threshold (rough hashing), the destination point does not fall inside the bounding box (pink lines) and the cross product is not even evaluated. For the cells that cross or touch the periodicity threshold (fine hashing), the bounding box only limits the latitudes, therefore the cross-product is evaluated and the cell is chosen,



In the bggd nogt case, it happens for the destination points (ORCA1) with longitudes 177.5, 178.5, 179.5 and latitude 0 that are all assigned to the bggd cell with corners with longitude 357.5 and 0 and latitude -1.268 and 0.
Notice that for points on the equator with longitude smaller than 177.5, they are considered to the left of the 1-2 side (south) and to the right of the 2-3 side (east) therefore this source grid is rejected.
For points on the equator with longitude larger than 180.0, even if they fall in the degenerated bounding box and would pass the cross-product test, they do not represent a problem because the correct source cell has a lower index than the flawed one, therefore the search loop is interrupted before testing for the flawed cell.

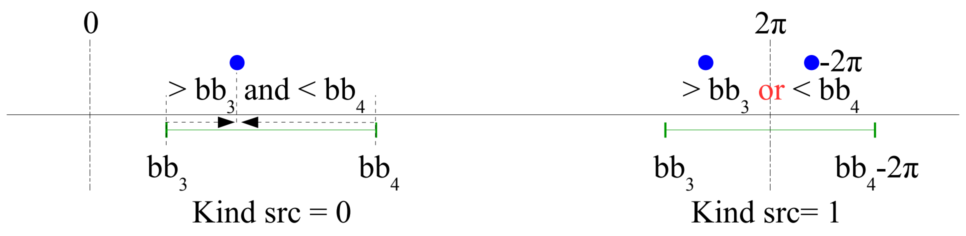
Two fixes now solve the problem:

1. In the cross-product test, do not consider the null cross-products in the sign evaluation
   Replace the line
   ```
   cross_product_last = cross_product
   ```
   by
   ```
   if (cross_product.ne.0) cross_product_last = cross_product
   ```

2. In the bounding box definition, assing kind 0 to standard cells, then detect grids crossing the periodicity threshold (they have longitude size > π), switch the upper and lower longitude bounds and assign to them kind 1 (notice that all the cells are constructed toward the East of the reference point, therefore only the crossing beyond 2π has to be dealt with).
   In the bilinear remapping modify the bounding box test from
   ```
       if (plat <= src_grid_bound_box(2,srch_add) .and.  &
               plat >= src_grid_bound_box(1,srch_add) .and.  &
               plon <= src_grid_bound_box(4,srch_add) .and.  &
               plon >= src_grid_bound_box(3,srch_add)) then
   ```

to a detailed test (see function lf_lon_in_box) using the same test if the grid kind is 0 but, for grid kind 1 (notice the OR condition)

```
     plon <= src_grid_bound_box(4,srch_add) .or.  &
            plon >= src_grid_bound_box(3,srch_add)
```


0
> bb3 and < bb4
bb3
bb4
Kind src = 0
2π
-2π
> bb3 or < bb4
bb3
bb4-2π
Kind src= 1