Technical Reports

CENTRO EURO-MEDITERRANEO
PER I CAMBIAMENTI CLIMATICI

SCO – Scientific Computing and Operations Division

OASIS3:

Analysis and Parallelization

Italo Epicoco
University of Salento, Lecce

Silvia Mocavero
Scientific Computing and Operations, CMCC

Enrico Scoccimarro
Istituto Nazionale di Geofisica e Vulcanologia (INGV) and Numerical Applications and Scenarios Division, CMCC

Giovanni Aloisio
Scientific Computing and Operations, CMCC
University of Salento, Lecce

Centro Euro-Mediterraneo
per i Cambiamenti Climatici
www.cmcc.it

January 2009 ■ TR

OASIS3 Parallelization

Summary

This technical report describes the activity of optimization and parallelization of the OASIS3 coupler, used to couple climate models such as the atmospheric model ECHAM5 and the ocean model NEMO.

Keywords: OASIS3, coupling, climate models, MPI2.

JEL Classification: C63

Version: 1.0
Release Date: 07/01/2009

Address for correspondence:
Italo Epicoco
Euro-Mediterranean Centre for Climate Change
Viale Gallipoli, 49
73100 Lecce, Italy.
E-mail: italo.epicoco@unile.it

Table of Contents

Introduction

OASIS3 [1] is a portable set of Fortran 77, Fortran 90 and C routines. At run-time, OASIS3 acts as a separate mono process executable, which main function is to interpolate the coupling fields exchanged between the component models, and as a library linked to the component models, the OASIS3 PRISM Model Interface Library (OASIS3 PSMILe).

The aim of this document is to show the activity of both optimization and parallelization of OASIS3, in order to reduce time spent to execute coupling operations at each temporal step.

The outline of this report is as follows: in Section 1 we present the OASIS3 sequential code. Optimization and parallelization activities are respectively shown in Section 2 and 3, whereas in Section 4 parallel configuration is described. Finally, in Section 5 we show functional test and results.

1. OASIS3 sequential code

Before approaching the optimization and parallelization of software, the analysis of the code, of the internal structures and of the dependence correlation among the functions and modules is mandatory. In this chapter we give an overview of the OASIS3 software structure, a more detailed description of the OASIS3 functionalities can be found on the OASIS3 users guide [1]

1.1. OASIS3 flow-chart

The following steps characterize OASIS3 coupling activity:

- An *initialization* step for:
  - Initialization of internal parameters, descriptors of logical I/O units, data structure for timing, communication environment among processes, grids
  - Arrays allocation
  - Set up of dynamic allocation for all grid-related fields
  - Opening of needed files
- A *time loop* containing:
  - A function for time update
  - A loop on sequential model in order to:
    - Compute the current number of fields to be exchanged between coupled models
    - Retrieve fields value from the models
    - Execute, for each field, the coupling operations (pre-processing, interpolation, “cooking” and post-processing)
    - Provide the models with the new computed values of the fields
  - A function for re-initialization
- A *finalization* step for:
  - Waiting child processes
  - Arrays deallocation

In Figure 1, OASIS3 flow-chart is shown.

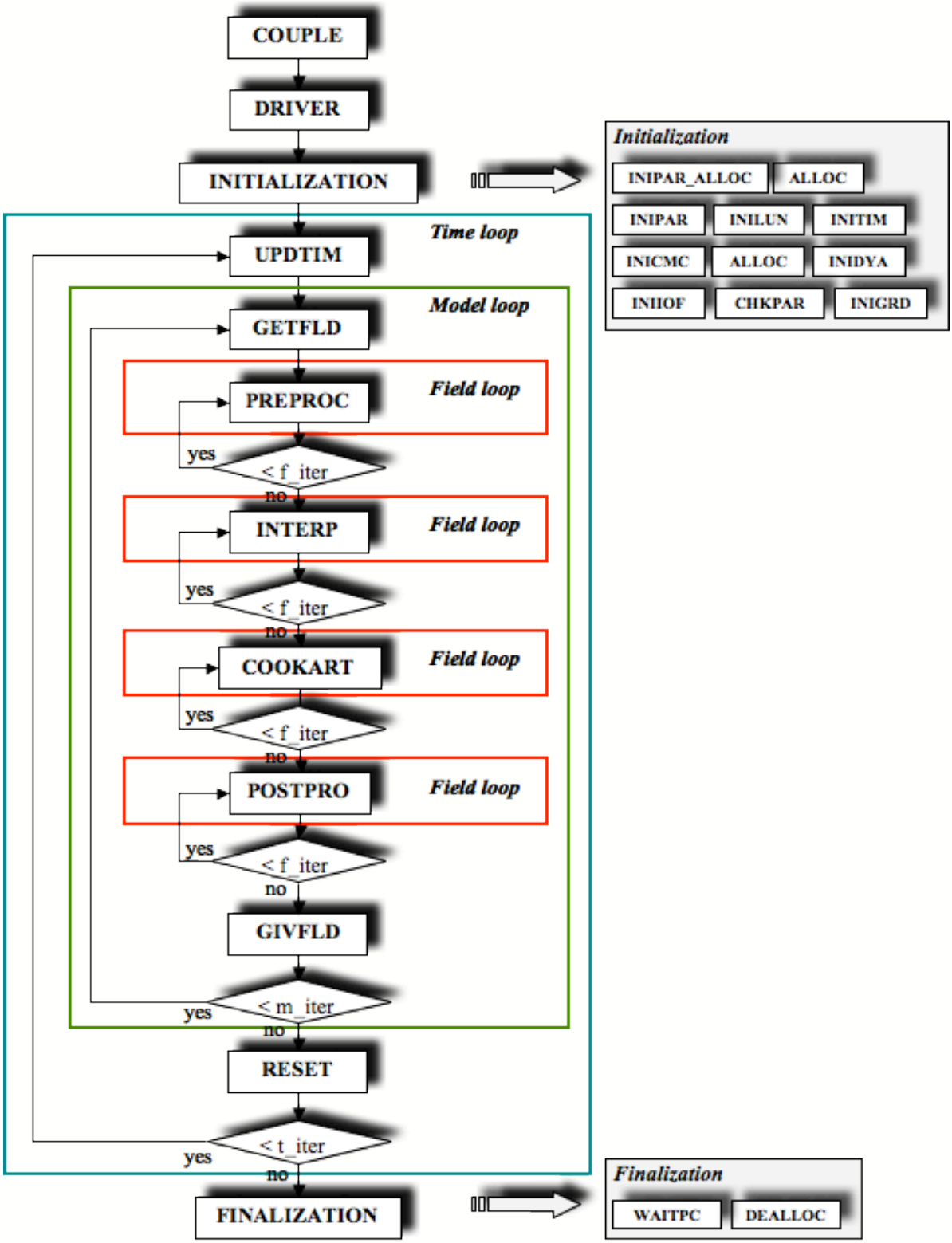

Figure 1 - OASIS3 flow-chart

1.2. OASIS3 coupling

Different transformations and 2D interpolations are available in OASIS3 to adapt the coupling fields from a source model grid to a target model grid. They are divided into five general classes and must by applied with a logical strict order: time transformation (with CLIM/MPI1-MPI2 and PSMILe only), pre-processing, interpolation, "cooking", and post-processing. This order of precedence is conceptually logical, but is also constrained by the OASIS3 software internal structure.

1.2.1. Pre-processing

The following transformations are available in the pre-processing part of OASIS3:

- MASK is used before the EXTRAP operation. A given *real* value *valmask* is assigned to all masked points following the source grid mask, so they can be detected by EXTRAP
- EXTRAP performs the extrapolation of a field over its masked points. The MASK operation must be used just before, so that EXTRAP can identify masked points
- CHECKIN calculates the mean and maximum values of the source field and prints them to the coupler log file *cplout*
- CORRECT reads external fields from binary files and uses them to modify the coupling field. This transformation can be used, for example, to perform flux correction on the field
- Other obsolete transformations such as REDGLO and INVERT.

1.2.2. Interpolation

The following interpolations are available in OASIS3:

- BLASOLD performs a linear combination of the current coupling field with other coupling fields or with a constant before the interpolation per se
- SCRIPR gathers the interpolation techniques offered by Los Alamos National Laboratory SCRIP library. SCRIPR supports 2D vector interpolation

- INTERP includes different techniques of interpolation such as (i) BILINEAR interpolation using 4 neighbors, (ii) BICUBIC interpolation and (iii) NNEIBOR interpolation, performing a nearest-neighbor interpolation
- MOZAIC performs the mapping of a field from a source to a target grid. The grid-mapping dataset, i.e. the weights and addresses of the source grid points used to calculate the value of each target grid point are defined by the user in a file
- NOINTERP is the analysis that has to be chosen when no other transformation from the interpolation class is chosen. In this case the field does not require interpolation.
- FILLING performs the blending of a regional data set with a climate global one for a Sea Surface Temperature (SST) or a Sea Ice Extent field. This occurs when coupling a non-global ocean model with a global atmospheric model.

1.2.3. "Cooking"

The following operations are supported within the "cooking" transformation of OASIS3:

- CONSERV performs global flux conservation
- SUBGRID can be used to interpolate a field from a coarse grid to a finer target grid (the target grid must be finer over the whole domain)
- BLASNEW performs a linear combination of the current coupling field with any other fields after the interpolation
- MASKP masks the fields after interpolation.

1.2.4. Post-processing

The following operations are available in the post-processing part of OASIS3:

- CHECKOUT calculates the mean and maximum values of an output field and prints them to the coupler output *cplout*
- Other obsolete transformations such as REVERSE and GLORED.

1.3. OASIS3 sources

The OASIS3 source code is organized within the following directory structure:

- <OASIS3_HOME>/src
- <OASIS3_HOME>/libsrc
    - <OASIS3_HOME>/libsrc/anaisg GAUSSIAN interpolation library
    - <OASIS3_HOME>/libsrc/anaism SURFMESH interpolation library
    - <OASIS3_HOME>/libsrc/mpp_io I/O library
    - <OASIS3_HOME>/libsrc/fscint INTERP interpolation library
    - <OASIS3_HOME>/libsrc/clim CLIM-MPI1/2 communication library
    - <OASIS3_HOME>/libsrc/psmile PRISM System Model Interface Library
    - <OASIS3_HOME>/libsrc/scrip SCRIPR interpolation library

2. Sequential code optimization

Before starting the parallelization phase, a deep analysis of the code has been performed in order to optimize sequential code computational time. The outcome of the analysis showed that the extrapolation function is one of the major time consuming functions of the coupler. The optimization effort has been focused on that function.

The `extrap.f`, as mentioned before, performs the extrapolation of a field over its masked points. There are two extrapolation methods:

- WEIGHT, the N-weighted-neighbor method. The user has to build a grid-mapping file, giving for each target grid point the weights and addresses of the source grid points used in the extrapolation (routine *extraw.f*)
- NINENN, the N-nearest-neighbor method. The procedure is iterative. At each iteration, the set of remaining masked points evolves at each iteration (routine *extrap.f*).

Extrap function has been optimized as described in the following. Fields are grouped by dataset.

In the *namecouple* configuration file, for each field the extrapolation method, followed by three values, is specified.

```
# EXTRAP operation for NINENN
NINENN $NV $NIO $NID
```

1) $NV: the minimum number of neighbors required to perform the extrapolation
2) $NIO: the flag that indicates if the weight-address-and-iteration number of related dataset will be calculated and written by OASIS3 (NIO = 1), or only read (NIO = 0) in a file named *nweights,* or only calculated (NIO = 2).
3) $NID: the identificator for the weight-address-iteration-number dataset

The weight-address-and-iteration values, computed or read from file by the *Extrap* function, are also maintained in main memory in order get better performance. A flag variable, DFLAG, is used in order to take into account the availability of the weight-address-and-iteration values on the memory. More in detail the *Extrap* function writes, reads or computes weight-address-and-iteration number, related to the specific dataset, based on $NIO and the DFLAG, using the following algorithm:

```
IF ((NIO >= 1) AND (NOT(DFLAG)) ) THEN
      COMPUTE WEIGHT-ADDRESS-ITERATION NUMBER
      IF (NIO < 2) THEN
            WRITE WEIGHT-ADDRESS-ITERATION NUMBER IN NWEIGHT FILE
      END IF
END IF

IF ((NIO = 0) AND (NOT(DFLAG)) ) THEN
      READ WEIGHT-ADDRESS-ITERATION NUMBER FROM NWEIGHT FILE
END IF
...
IF (NIO < 2) THEN
      DFLAG = TRUE
END IF
```

From a careful analysis of the code it is evident that:

a) the *Extrap* function will fail if the first field of a dataset has $NIO=0; in this case the weight-address-and-iteration number has not been already computed (so that DFLAG is *false*) and consequently *nweight* file does not exists yet; for $NIO=0 the function tries to read from *nweight* file that does not exist.

b) Keeping in mind that the first field of a given dataset must have $NIO=1 for that field the weight-address-and-iteration numbers will be computed, stored into the memory and dumped into *nweight* file. For all the following fields, belonging to the same dataset, the weight-address-and-iteration values will be then already available in memory and hence they will never be read from file. We conclude that the writing and reading operation of the *nweight* file can be safely removed from the code.:

```
IF ((NIO >= 0) AND (NOT(DFLAG)) ) THEN
      COMPUTE WEIGHT-ADDRESS-ITERATION NUMBER
END IF
...
IF (NIO < 2) THEN
      DFLAG = TRUE
END IF
```

3. Parallelization

3.1. Parallel model

The design of the parallel version of OASIS start considering the data dependency and function dependency over all of the coupling process, in particular, prep-processing, interpolation, cookart and post-processing transformations are independently performed on each field, then transformations on a single field can be considered as a separate task. The parallel model follow an embarrassing parallel approach splitting the fields over the available processes (Figure 2). Since the communication pattern is regular and the execution time of a generic task is of order of magnitude of second, a static scheduling policy can be preferred against a dynamic scheduling. This way, a pre-fixed number of tasks are allocated to each process instead of introducing an overhead due to a dynamic scheduling policy.
Generic process, with rank *r*, performs transformations of the *i*-th field if and only if the following equation is satisfied:

$$\left\lceil \frac{r \times n}{p} \right\rceil + 1 \le i \le \left\lceil \frac{(r+1) \times n}{p} \right\rceil \qquad (1)$$

where:
r = process rank
n = total number of fields
p = total number of processes

Since coupling operations, on a specific field, are independent from transformations on the others, transformations loops, divided in sequential code, have been merged in a unique one. Before performing coupling operation, the values of the current fields, which the OASIS master process has directly imported from models, are scattered to the other OASIS processes. Therefore, each process knows about the values of fields to be processed.
After all transformations, each process, except the master, will send new values for fields, to the master which will send them to the models for a new time loop.

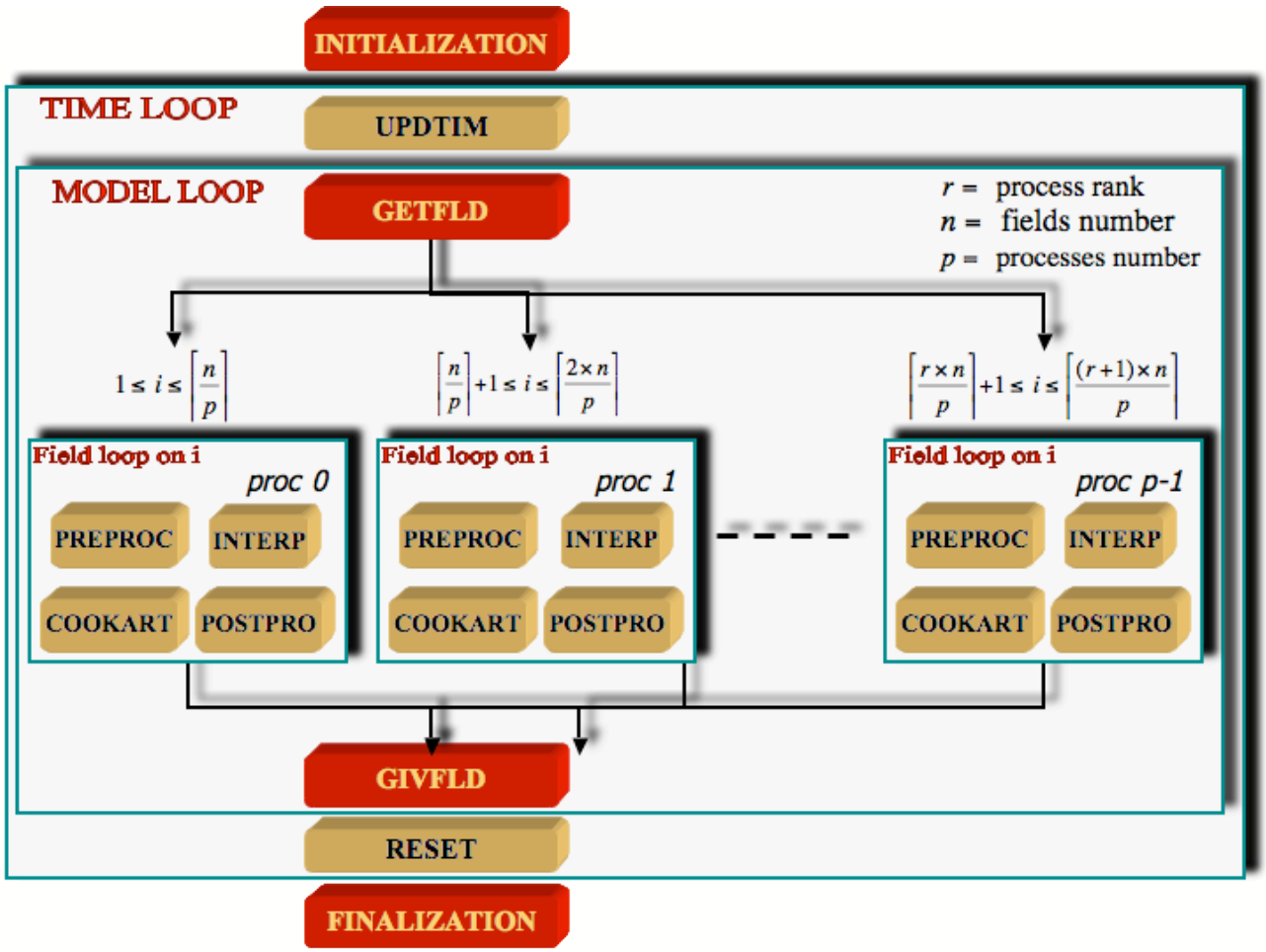

Figure 2 - Parallel OASIS3 flow-chart

3.2. Implementation

Several modifications have been introduced in the source code in order to implement the parallel version of the OASIS3. In particular the following aspects have been introduced or modified in order to keep each OASIS process independent from others and to avoid communication among processes.

- File access: OASIS uses a file to log the events and transformations during the execution; this file is only written and never read by the process. In order to avoid synchronization

among parallel processes, different log files have been created; each process writes its own file. Same solution can be applied for files that contain data related to fields; in this case, because of a process will elaborate the same fields over the whole simulation, we can safely split these files into peaces each one created by a single parallel process. Such a case occurs during the interpolation transformation. In particular when the SCRIP library performs a remapping, it first checks whether the file containing the corresponding remapping weights and addresses exists. If it exists, it reads them from the file; if not, it calculates them and stores them in a file. Also in this case, in order to avoid the synchronization among parallel processes, each process creates its own file containing the weights and addresses of the fields it has in charge.

- Redundancy: the inherently sequential part of the computation is performed by different parallel process instead of having one process that computes and the others waiting for the results. Such a case happens for example during the extrapolation transformation and in particular during the evaluation of the weight-address-and-iteration number dataset. Considering that, during extrapolation, fields are grouped into subset having the same dataset, only for the first field of a given dataset the weight-address should be calculated. For the other fields of the same dataset, the same weight-address will be used. In the parallel version, it could happen that fields belonging to the same dataset are split on different processes; in this case, in order to avoid synchronization and to reduce the process communication at the essential, we forced each process to evaluate the weights when ever it encounter a field belonging to a new dataset, even if other processes have already evaluated the weights for that dataset.
- OASIS Communicator: in order to keep the communication domain distinct, a new communicator has been created to encapsulate the communication related to the processes belonging to OASIS.

3.3. Implementation details

Coupler parallelization has been performed using MPI2 library. The following routines have been modified in order to parallelize the original sequential code:

<OASIS3_HOME>/src/couple.f

Within *couple* routine, *cplout* log file opening has been modified: each process opens a different log file in order to distinguish information related to a specific process.

```
------------------------------------------------------------------
c SCO Division >>
        USE mod_comclim
#include <mpif.h>
        INTEGER sco_imyrank
        CHARACTER*24 sco_cplname

        CALL MPI_INIT(mpi_err)
        IF(mpi_err .NE. MPI_SUCCESS) THEN
           CALL HALTE ('STOP in CLIM_Init_Oasis')
        END IF
        WRITE(UNIT = nulprt,FMT = *)' MPI_INIT done '
        CALL MPI_COMM_RANK(MPI_COMM_WORLD,sco_imyrank,mpi_err)
        IF(sco_imyrank .LE. 9) THEN
          WRITE(sco_cplname,'(''cplout_'',I1)') sco_imyrank
        ELSE
          WRITE(sco_cplname,'(''cplout_'',I2)') sco_imyrank
        ENDIF
c SCO Division <<

c SCO Division has commented out the following lines >>
c      OPEN (UNIT = nulou,FILE ='cplout',STATUS='UNKNOWN',
c     $      FORM ='FORMATTED',ERR = 110,IOSTAT = iost)
        OPEN (UNIT = nulou,FILE =sco_cplname,STATUS='UNKNOWN',
     $      FORM ='FORMATTED',ERR = 110,IOSTAT = iost)
c SCO Division <<
------------------------------------------------------------------
```

<OASIS3_HOME>/src/driver.f

A unique loop with all the transformation operations replaced different loops. A group of fields has been assigned to each process, using equation (1).

```
------------------------------------------------------------------
c SCO Division >>
      USE mod_extrapol
#include <mpif.h>
      REAL sco_down, sco_up
      REAL r_coupler_rank, r_ifield, r_coupler_size
c SCO Division <<
```

```
c SCO Division >>
                 call MPI_COMM_SIZE(coupler_comm, coupler_size, mpi_err)
                 r_coupler_rank=coupler_rank
                 r_ifield=ifield
                 r_coupler_size=coupler_size
                 sco_down=(r_coupler_rank*r_ifield)/r_coupler_size
                 sco_up=((r_coupler_rank+1)*r_ifield)/r_coupler_size

                 DO 123 jsco = ceiling(sco_down)+1, ceiling(sco_up)
                     CALL preproc (iindex(jsco), jsco)
                     CALL interp (iindex(jsco), jsco)
                     CALL cookart (iindex(jsco), jsco)
                     CALL postpro (iindex(jsco), jsco)
 123   CONTINUE
-----------------------------------------------------------------
```

<OASIS3_HOME>/src/inicmc.f

To reduce communication overhead, only the master will call *CLIM_Stepi* routine, which retrieves info related to a specific model.

```
-----------------------------------------------------------------------
c SCO Division >>
      IF (coupler_rank .EQ. 0) THEN
c SCO Division <<
            CALL CLIM_Stepi (cmodnam(jm), infos)
            IF (infos .lt. nbcplproc(jm)) THEN
               WRITE (UNIT = nulou,FMT = *)
     $              'PROBLEM: Got initial informations only from',
     $              infos
               WRITE (UNIT = nulou,FMT = *)
     $              'processes of model',  cmodnam(jm)
               CALL halte ('STOP in inicmc.f')
            ELSE
               IF (nlogprt .GE. 1) THEN
                  WRITE (UNIT = nulou,FMT = *)
     $                 ' Got step informations from model ',
     $                 cmodnam(jm)
               ENDIF
            ENDIF
c SCO Division >>
      END IF
c SCO Division <<
-----------------------------------------------------------------------
```

<OASIS3_HOME>/src/getfld.f

In *getfld* routine, a call to the *SCO_getdimimp*, computing dimension (in bytes) of fields the master must send to the other processes, has been introduced. After importing fields from the models, the master broadcasts imported values to all processes.

```
------------------------------------------------------------------------
c  SCO Division >>
      USE mod_comclim
#include <mpif.h>
c  SCO Division <<
IF (kiter .EQ. 0 .AND. ilagn .GT. 0) THEN
c SCO Division >>
           CALL SCO_getdimimp(ig_portin_id(iloc), jf, info(iloc))
           IF (info(iloc) .NE. 0) THEN
           CALL prcout('Problem in SCO getdimimp reading field on port',
     $             clname, 1)
              CALL prtout('error code number', info(iloc), 2)
              CALL HALTE ('STOP in getfld')
           ENDIF
c SCO Division <<

c SCO Division >>
        IF (coupler_rank .EQ. 0) THEN
!$omp critical
c SCO Division <<
            CALL CLIM_Import
     $           (ig_portin_id(iloc), kiter*nstep, fldold(iadrold),
     $           info(iloc))
!$omp end critical
            IF (info(iloc) .NE. CLIM_Ok) THEN
               CALL prcout('Problem in reading field on port',
     $              clname, 1)
               CALL prtout('error code number', info(iloc), 2)
               CALL HALTE ('STOP in getfld')
            ENDIF
c SCO Division >>
        END IF
        call MPI_BCAST( fldold(iadrold), sco_ig_nbrecv(jf), MPI_BYTE,
     $      0, coupler_comm, error )
        IF (error .NE. MPI_SUCCESS) THEN
            CALL prtout('SCO: Error MPIBCAST-fldold',1,2)
            CALL HALTE ('STOP in getfld')
        ENDIF
c SCO Division <<
------------------------------------------------------------------------
```

<OASIS3_HOME>/src/preproc.f
<OASIS3_HOME>/src/cookart.f
<OASIS3_HOME>/src/postpro.f

In the *preproc*, *cookart* and *postpro* routines, the fields loop has been moved outside routines (in *driver* routine). Then, there is an external loop (on fields number) calling transformations.

```
 ----------------------------------------------------------------------
c SCO Division >>
c SCO Division has commented out the following lines
c        DO 210 jf = 1, kfield
c           ifield = kindex(jf)
c SCO Division <<
...
c SCO Division >>
c SCO Division has commented out the following line
c 210   CONTINUE
c SCO Division <<
----------------------------------------------------------------------
```

<OASIS3_HOME>/src/interp.f

Also in *interp* routine the field loop has been commented out (for the same reason). Moreover, the calls to *scriprmp* e *scriprmp_vector* routines have been modified adding a parameter meaning the calling process rank.

```
----------------------------------------------------------------------
c SCO Division >>
                  CALL scriprmp_vector (
     $                 fldnew(iadrnew), fldnew(iadrnew_J),
     $                 fldold(iadrold), fldold(iadrold_J),
     $                 cficbf(ifield),  cficbf(ifield_J),
     $                 cficaf(ifield),  cficaf(ifield_J),
     $                 isizold, isiznew,
     $                 mskold(iadrold_grid), mskold(iadrold_grid_J),
     $                 msknew(iadrnew_grid), msknew(iadrnew_grid_J),
     $                 xgrold(iadrold_grid),  ygrold(iadrold_grid),
     $                 xgrold(iadrold_grid_J),ygrold(iadrold_grid_J),
     $                 xgrnew(iadrnew_grid),  ygrnew(iadrnew_grid),
     $                 xgrnew(iadrnew_grid_J),ygrnew(iadrnew_grid_J),
     $                 ilonbf, ilatbf,
     $                 ilonaf, ilataf,
     $                 cmap_method(ifield), clgrdtyp,
     $                 il_sper, il_tper,
     $                 clsper, cltper,
     $                 cnorm_opt(ifield), corder(ifield),
     $                 crsttype(ifield), nbins(ifield),
     $                 lextrapdone(ifield), lrotate(ifield),
     $                 varmul(ifield), nscripvoi(ifield),
```

```
     $                  coupler_rank)
c                   CALL scriprmp_vector (
c     $                  fldnew(iadrnew), fldnew(iadrnew_J),
c     $                  fldold(iadrold), fldold(iadrold_J),
c     $                  cficbf(ifield),  cficbf(ifield_J),
c     $                  cficaf(ifield),  cficaf(ifield_J),
c     $                  isizold, isiznew,
c     $                  mskold(iadrold_grid), mskold(iadrold_grid_J),
c     $                  msknew(iadrnew_grid), msknew(iadrnew_grid_J),
c     $                  xgrold(iadrold_grid),  ygrold(iadrold_grid),
c     $                  xgrold(iadrold_grid_J),ygrold(iadrold_grid_J),
c     $                  xgrnew(iadrnew_grid),  ygrnew(iadrnew_grid),
c     $                  xgrnew(iadrnew_grid_J),ygrnew(iadrnew_grid_J),
c     $                  ilonbf, ilatbf,
c     $                  ilonaf, ilataf,
c     $                  cmap_method(ifield), clgrdtyp,
c     $                  il_sper, il_tper,
c     $                  clsper, cltper,
c     $                  cnorm_opt(ifield), corder(ifield),
c     $                  crsttype(ifield), nbins(ifield),
c     $                  lextrapdone(ifield), lrotate(ifield),
c     $                  varmul(ifield), nscripvoi(ifield))
c SCO Division <<

c SCO Division >>
                     CALL scriprmp (
     $                   fldnew(iadrnew), fldold(iadrold),
     $                   isizold, isiznew,
     $                   mskold(iadrold_grid), msknew(iadrnew_grid),
     $                   xgrold(iadrold_grid), ygrold(iadrold_grid),
     $                   ilonbf, ilatbf,
     $                   xgrnew(iadrnew_grid), ygrnew(iadrnew_grid),
     $                   ilonaf, ilataf,
     $                   cmap_method(ifield), clgrdtyp,
     $                   il_sper, il_tper, clsper, cltper,
     $                   cficbf(ifield), cficaf(ifield),
     $                   cnorm_opt(ifield),
     $                   corder(ifield), crsttype(ifield),
     $                   nbins(ifield),
     $                   lextrapdone(ifield), varmul(ifield),
     $                   nscripvoi(ifield), coupler_rank)

c                     CALL scriprmp (
c     $                   fldnew(iadrnew), fldold(iadrold),
c     $                   isizold, isiznew,
c     $                   mskold(iadrold_grid), msknew(iadrnew_grid),
c     $                   xgrold(iadrold_grid), ygrold(iadrold_grid),
c     $                   ilonbf, ilatbf,
c     $                   xgrnew(iadrnew_grid), ygrnew(iadrnew_grid),
c     $                   ilonaf, ilataf,
c     $                   cmap_method(ifield), clgrdtyp,
c     $                   il_sper, il_tper, clsper, cltper,
c     $                   cficbf(ifield), cficaf(ifield),
```

```
c     $                   cnorm_opt(ifield),
c     $                   corder(ifield), crsttype(ifield),
c     $                   nbins(ifield),
c     $                   lextrapdone(ifield), varmul(ifield),
c     $                   nscripvoi(ifield))
c SCO Division <<
-----------------------------------------------------------------------
```

<OASIS3_HOME>/src/extrap.f

Changes within *extrap* routine derives from optimization process described in section 2. Even if *nweight* file reading code has not been commented out, it will never executed (reading conditions are never satisfied).

```
-----------------------------------------------------------------------
c SCO Division has commented out the following line >>
c          IF (krdwt .GE. 1 .AND. .NOT. lweight(knb)) THEN
          IF (krdwt .GE. 0 .AND. .NOT. lweight(knb)) THEN
c SCO Division <<
.
.
c SCO Division has commented out the following lines >>
c               CALL locwrite (clweight, zweights, ig_maxnfn*9*kxlon*
c     $              kylat, nulgn, iflag)
c               IF (iflag .NE. 0) THEN
c                   WRITE (UNIT = nulou,FMT = *)
c     $             'Problem in writing on UNIT = ', nulgn
c                   WRITE (UNIT = nulou,FMT = *)
c     $             'String locator is = ', clweight
c                   CALL HALTE ('Stop in extrap writing weights')
c               ENDIF
C
C* Adresses
C
c               CALL locwrint (cladress, iaddress, ig_maxnfn*9*kxlon*
c     $              kylat, nulgn, iflag)
c               IF (iflag .NE. 0) THEN
c                   WRITE (UNIT = nulou,FMT = *)
c     $             'Problem in writing on UNIT = ', nulgn
c                   WRITE (UNIT = nulou,FMT = *)
c     $             'String locator is = ', cladress
c                   CALL HALTE ('Stop in extrap writing addresses')
c               ENDIF
C
C* Iteration numbers
C
c               CALL locwrint (clincrem, iincre, ig_maxnfn*kxlon*kylat,
c     $                   nulgn, iflag)
```

```
c               IF (iflag .NE. 0) THEN
c                   WRITE (UNIT = nulou,FMT = *)
c     $             'Problem in writing on UNIT = ', nulgn
c                   WRITE (UNIT = nulou,FMT = *)
c     $             'String locator is = ', clincrem
c                   CALL HALTE ('Stop in extrap writing iteration number')
c               ENDIF
c SCO Division <<
.
.
c SCO Division >>
      IF (krdwt .EQ. 0) lweight(knb)= .TRUE.
c SCO Division <<
-------------------------------------------------------------------------
```

<OASIS3_HOME>/src/givfld.f

In *givfld* routine, a call to the *SCO_getdimexp*, computing dimension (in bytes) of fields the master must collect from the other processes, has been introduced. Point-to-point communications have been used from processes to send computed fields to the master, which will subsequently export to the models.

```
-------------------------------------------------------------------------
c SCO Division >>
      USE mod_comclim
#include <mpif.h>
c SCO Division <<
c SCO Division >>
      REAL sco_down, sco_up
      REAL r_coupler_rank, r_kfield, r_coupler_size
      INTEGER (kind=ip_intwp_p)              :: coupler_req,
     $  coupler_status(MPI_STATUS_SIZE)
c SCO Division <<
c SCO Division >>
        coupler_req = MPI_REQUEST_NULL
        coupler_status(:) = 0
c SCO Division <<
c SCO Division >>
        CALL SCO_getdimexp(ig_portout_id(iloc), jf, info(jf))
        IF (info(jf) .NE. 0) THEN
           CALL prcout('Problem in SCO getdimexp reading field on port',
     $             clname, 1)
           CALL prtout('error code number', info(jf), 2)
           CALL HALTE ('STOP in givfld')
        ENDIF

        r_coupler_rank=coupler_rank
        r_kfield=kfield
```

```
        r_coupler_size=coupler_size
        sco_down=(r_coupler_rank*r_kfield)/r_coupler_size
        sco_up=((r_coupler_rank+1)*r_kfield)/r_coupler_size
        coupler_req = MPI_REQUEST_NULL
        coupler_status(:) = 0
        IF (coupler_rank .EQ. 0) THEN
             IF(jf .GT. ceiling(sco_up)) THEN
                CALL MPI_Irecv (fldnew(iadrnew), sco_ig_nbsend(jf),
     $          MPI_BYTE,((jf-1)*coupler_size)/kfield, 0, coupler_comm,
     $          coupler_req, error )
             ENDIF
        ELSE
             IF (jf.GE.(ceiling(sco_down)+1).and.
     $       jf.LE.(ceiling(sco_up))) THEN
                CALL MPI_ISEND(fldnew(iadrnew), sco_ig_nbsend(jf),
     $          MPI_BYTE, 0, 0, coupler_comm, coupler_req, error )
             ENDIF
        ENDIF

        IF (coupler_req .NE. MPI_REQUEST_NULL) THEN
            call MPI_WAIT(coupler_req, coupler_status, error)
        ENDIF
        IF (coupler_rank .EQ. 0) THEN
c SCO Division <<
!$omp critical
             CALL CLIM_Export
     $            (ig_portout_id(iloc), kiter*nstep,
     $            fldnew(iadrnew), info(jf))
             infos = infos + info(jf)
c SCO Division >>
        ENDIF
C
      IF (coupler_rank .EQ. 0) THEN
c SCO Division <<
          IF (infos .NE. CLIM_Ok) THEN
              DO 220 jf = 1, kfield
                IF (info(jf) .NE. CLIM_Ok) THEN
                    CALL prcout
     $                  ('WARNING: problem in writing field on port',
     $                  clname, 1)
                    CALL prtout
     $                  ('error code number', info(jf), 2)
                ENDIF
 220          CONTINUE
              CALL HALTE ('STOP in givfld')
          ENDIF
c SCO Division >>
      ENDIF
c SCO Division <<
-----------------------------------------------------------------------
```

<OASIS3_HOME>/libsrc/clim/CLIM_Init_Oasis.F

In *CLIM_Init_Oasis* routine, coupler communicator is created, arrays for storing dimension (in bytes) of fields to be exchanged are allocated, some operations such as the exchange operations among models and the models spawn, are executed only by the master.

```
------------------------------------------------------------------------
c SCO Division >>
c       Create a communicator for couplers

         CALL MPI_COMM_DUP(MPI_COMM_WORLD,coupler_comm,mpi_err);
         IF (mpi_err .NE. MPI_SUCCESS) THEN
             WRITE (nulprt,*) 'SCO: Error MPI_Comm_dup'
             CALL HALTE ('STOP in CLIM_Init_Oasis')
         ENDIF

         CALL MPI_COMM_RANK(coupler_comm,coupler_rank,mpi_err)
         IF(mpi_err .NE. MPI_SUCCESS) THEN
             WRITE (nulprt,*) 'SCO: Error MPI_Comm_rank = ',coupler_rank
             CALL HALTE ('STOP in CLIM_Init_Oasis')
         ENDIF

         IF (coupler_rank .EQ. 0) THEN
c SCO Division <<

c SCO Division: has commented the following
c        CALL MPI_COMM_DUP(MPI_COMM_WORLD,impi_newcomm(1),mpi_err)
c        WRITE(nulprt,*)'Init - - comm_dup done= ',impi_newcomm(1)
c SCO Division <<
c SCO Division >>
         CALL MPI_COMM_DUP(MPI_COMM_SELF,impi_newcomm(1),mpi_err)
         WRITE(nulprt,*)'Init - - comm_dup done= ',impi_newcomm(1)
c SCO Division <<
...
c SCD Division >>
        ENDIF
c SCO Division <<

c SCO Division >>
      ALLOCATE (sco_ig_nbrecv(ig_total_nfield), stat=il_err)
      IF (il_ERR.ne.0) WRITE(nulprt,*)'Error in "sco_ig_nbrecv"
     $    allocation in CLIM_Init_Oasis routine ! '
      sco_ig_nbrecv(:)=0
      ALLOCATE (sco_ig_nbsend(ig_total_nfield), stat=il_err)
      IF (il_ERR.ne.0) WRITE(nulprt,*)'Error in "sco_ig_nbsend"
     $    allocation in CLIM_Init_Oasis routine ! '
      sco_ig_nbsend(:)=0
c SCO Division <<
c SCO Division >>
        IF (coupler_rank .EQ. 0) THEN
```

```
c SCO Division <<
...
c SCO Division >>
        ENDIF
c SCO Division <<
-----------------------------------------------------------------------
```

<OASIS3_HOME>/libsrc/clim/CLIM_Quit.F

Code for deallocation of variables introduced in parallel version has been added. Some useless code sections have been deleted.

```
-----------------------------------------------------------------------
c SCO Division has commented out the following lines >>
c      CALL MPI_Comm_Rank(mpi_comm,il_rank,mpi_err)
c
c      DO ji = 0, ncplprocs-1
c        IF (il_rank.eq.modtid(ji)) THEN
c SCO Division <<
c SCO Division >>
            DEALLOCATE (sco_ig_nbrecv, stat = il_ERR)
            IF (il_ERR.ne.CLIM_Ok) WRITE(nulprt,*)' Problem
     $          in "sco_ig_nbrecv" deallocation in CLIM_Quit !'
            DEALLOCATE (sco_ig_nbsend, stat = il_ERR)
            IF (il_ERR.ne.CLIM_Ok) WRITE(nulprt,*)' Problem
     $          in "sco_ig_nbsend" deallocation in CLIM_Quit !'
c SCO Division <<
c SCO Division >>
c               CALL MPI_Buffer_Detach(dl_bufaddr,il_bufsizebyt,mpi_err)
c SCO Division <<
c SCO Division has commented out the following lines >>
c        ENDIF
c      END DO
c SCO Division <<
-----------------------------------------------------------------------
```

<OASIS3_HOME>/libsrc/clim/CLIM_Start_MPI.F

Only the master executes some initialization operations.

<OASIS3_HOME>/libsrc/clim/mod_comclim.F90

In *mod_comclim* module, the declaration of some global variables, used by clim libraries, has been added.

```
------------------------------------------------------------------------
! SCO Division >>
  INTEGER(kind=ip_intwp_p), DIMENSION(:), ALLOCATABLE :: sco_ig_nbrecv
  INTEGER(kind=ip_intwp_p), DIMENSION(:), ALLOCATABLE :: sco_ig_nbsend
! SCO Division <<
! SCO Division <<
  INTEGER(kind=ip_intwp_p) :: coupler_rank, coupler_comm, coupler_size
! SCO Division >>
------------------------------------------------------------------------
```

<OASIS3_HOME>/libsrc/scrip/scriprmp.F

In *scriprmp* routine, a parameter indicating calling process rank has been added. It is used to create different remapping files in order to solve some concurrency problems of I/O operations (see section 4).

```
------------------------------------------------------------------------
c SCO Division >>
c      SUBROUTINE scriprmp (dst_array, src_array, src_size, dst_size,
c     $                     src_mask, dst_mask,
c     $                     src_lon, src_lat, nlon_src, nlat_src,
c     $                     dst_lon, dst_lat, nlon_dst, nlat_dst,
c     $                     map_method, cdgrdtyp,
c     $                     id_sper, id_tper, cd_sper, cd_tper,
c     $                     src_name, dst_name,
c     $                     normalize_opt, order, rst_type, n_srch_bins,
c     $                     lextrapdone, rl_varmul, id_scripvoi)
      SUBROUTINE scriprmp (dst_array, src_array, src_size, dst_size,
     $                     src_mask, dst_mask,
     $                     src_lon, src_lat, nlon_src, nlat_src,
     $                     dst_lon, dst_lat, nlon_dst, nlat_dst,
     $                     map_method, cdgrdtyp,
     $                     id_sper, id_tper, cd_sper, cd_tper,
     $                     src_name, dst_name,
     $                     normalize_opt, order, rst_type, n_srch_bins,
     $                     lextrapdone, rl_varmul, id_scripvoi,
     $                     icoupler_rank)
c SCO Division <<
c SCO Division >>
      INTEGER icoupler_rank
      CHARACTER*2 scoupler_rank
c SCO Division <<
c SCO Division >>
      IF (icoupler_rank .LE. 9) THEN
        WRITE(scoupler_rank,'(''0'',I1)') icoupler_rank
      ELSE
        WRITE(scoupler_rank,'(I2)') icoupler_rank
```

```
      END IF
c SCO Division <<
C
C* -- get the name of the file containig the remapping matrix
C
      SELECT CASE (map_method)
      CASE ('CONSERV')          ! conservative remapping
       cmapping =
     $     src_name(1:4)//' to '//dst_name(1:4)//' '//map_method//
     $     ' '//normalize_opt(1:4)//' remapping'

c SCO Division >>
       crmpfile =
     $      'rmp_'//src_name(1:4)//'_to_'//dst_name(1:4)//'_'//
     $      map_method(1:7)//'_'//normalize_opt(1:8)//'_'//
     $      scoupler_rank//'.nc'
c SCO Division <<
      CASE DEFAULT
       cmapping =
     $     src_name(1:4)//' to '//dst_name(1:4)//' '//map_method//
     $     ' remapping'

c SCO Division >>
       crmpfile =
     $      'rmp_'//src_name(1:4)//'_to_'//dst_name(1:4)//'_'//
     $      map_method(1:7)//'_'//scoupler_rank//'.nc'
c SCO Division <<
      END SELECT
-----------------------------------------------------------------------
```

<OASIS3_HOME>/libsrc/scrip/scriprmp_vector.F90

In *scriprmp_vector* routine, a parameter indicating calling process rank has been added. It is used to create different remapping files in order to solve some concurrency problems of I/O operations (see section 4).

```
-----------------------------------------------------------------------
! SCO Division >>
!SUBROUTINE scriprmp_vector(              &
!     dst_arrayI_dstA, dst_arrayJ_dstB,   &
!     src_arrayI_srcA, src_arrayJ_srcB,   &
!     grd_name_srcA,   grd_name_srcB,     &
!     grd_name_dstA,   grd_name_dstB,     &
!     src_size,        dst_size,          &
!     msk_srcA,        msk_srcB,          &
!     msk_dstA,        msk_dstB,          &
!     lon_srcA,        lat_srcA,          &
!     lon_srcB,        lat_srcB,          &
```

```
!     lon_dstA,         lat_dstA,          &
!     lon_dstB,         lat_dstB,          &
!     nlon_src,         nlat_src,          &
!     nlon_dst,         nlat_dst,          &
!     map_method,       cdgrdtyp,          &
!     id_sper,          id_tper,           &
!     cd_sper,          cd_tper,           &
!     normalize_opt,    order,             &
!     rst_type,         n_srch_bins,       &
!     lextrapdone,      lprojcart,         &
!     rl_varmul,        id_scripvoi
SUBROUTINE scriprmp_vector(               &
     dst_arrayI_dstA, dst_arrayJ_dstB,    &
     src_arrayI_srcA, src_arrayJ_srcB,    &
     grd_name_srcA,   grd_name_srcB,      &
     grd_name_dstA,   grd_name_dstB,      &
     src_size,        dst_size,           &
     msk_srcA,        msk_srcB,           &
     msk_dstA,        msk_dstB,           &
     lon_srcA,        lat_srcA,           &
     lon_srcB,        lat_srcB,           &
     lon_dstA,        lat_dstA,           &
     lon_dstB,        lat_dstB,           &
     nlon_src,        nlat_src,           &
     nlon_dst,        nlat_dst,           &
     map_method,      cdgrdtyp,           &
     id_sper,         id_tper,            &
     cd_sper,         cd_tper,            &
     normalize_opt,   order,              &
     rst_type,        n_srch_bins,        &
     lextrapdone,     lprojcart,          &
     rl_varmul,       id_scripvoi,        &
     icoupler_rank )
! SCO Division <<
INTEGER(KIND=int_kind), INTENT(in) :: &
       nlon_src, nlat_src, & ! number of longitudes and latitudes on source grid
       nlon_dst, nlat_dst, & ! number of longitudes and latitudes on target grid
       src_size, dst_size, & ! number of source/target grid cells
       n_srch_bins,        & ! number of search bins for SCRIP
       msk_srcA(src_size), & ! grid mask for srcA
       msk_srcB(src_size), & ! grid mask for srcB
       msk_dstA(dst_size), & ! grid mask for dstA
       msk_dstB(dst_size), & ! grid mask for dstB
       id_sper, &          ! number of overlapping points for source grid
       id_tper, &          ! number of overlapping points for target grid
       id_scripvoi, &      ! number of neighbour for DISTWGT and GAUSWGT
! SCO Division >>
       icoupler_rank         ! process rank
! SCO Division <<
!SCO Division >>
      INTEGER icoupler_rank
      CHARACTER*2 scoupler_rank
!SCO Division <<
```

```
! SCO Division >>
      IF (icoupler_rank .LE. 9) THEN
        WRITE(scoupler_rank,'(''0'',I1)') icoupler_rank
      ELSE
        WRITE(scoupler_rank,'(I2)') icoupler_rank
      END IF
! SCO Division <<
-----------------------------------------------------------------------
```

<OASIS3_HOME>/src/ sco_getdim.f

Finally, the file *sco_getdim.f* has been added. It contains the code of the new routines *SCO_getdimimp* and *SCO_getdimexp*.

```
-----------------------------------------------------------------------
SUBROUTINE SCO_getdimimp(id_port_id,sco_jf, kinfo)
c
#if defined use_comm_MPI1 || defined use_comm_MPI2 || (!defined
use_comm_MPI1 && !defined use_comm_MPI2 && !defined use_comm_SIPC &&
!defined use_comm_GMEM && !defined use_comm_PIPE && !defined
use_comm_NONE)
      USE mod_comclim
#include <mpif.h>
INTEGER (kind=ip_intwp_p)       kinfo
INTEGER (kind=ip_intwp_p)     ilk, iseg, is, ilgb, ilen, sco_jf
c
c*    0. Entering
c
      IF (coupler_rank .EQ. 0) THEN
      kinfo = 0
c*    1. check for this port in my list
      iport = -1
      IF (myport(1,id_port_id).eq.CLIM_In) iport=id_port_id
      IF (iport.lt.0) THEN
         kinfo = -1
         WRITE(nulprt,FMT='(A,A)')
     *         'SCO getdimimp - WARNING - Invalid port out: ',
     $        cports(id_port_id)
         GO TO 1010
      ENDIF

c*    2. check for connected ports (in)
      ityp = myport(2,iport)
      DO 290 ip=1,myport(5,iport)
         ilk  = myport(5+ip,iport)
         iseg = mylink(4,ilk)
         ilgb = 0
         DO 260 is=1,iseg
                  ilen = mylink(4+2*is,ilk)
```

```
                  ilgb = ilgb + ilen
 260           CONTINUE
c
 290  CONTINUE
      sco_ig_nbrecv(sco_jf)=ilgb*8
      ENDIF
      call MPI_BCAST( sco_ig_nbrecv(sco_jf), 1, MPI_INTEGER,
     $       0, coupler_comm, error )
          IF (error .NE. MPI_SUCCESS) THEN
              CALL prtout('SCO: Error MPIBCAST',1,2)
              CALL HALTE ('STOP in sco_getdimimp')
          ENDIF
c
 1010 CONTINUE
#endif
      RETURN
      END

SUBROUTINE SCO_getdimexp(id_port_id,sco_jf, kinfo)

#if  defined  use_comm_MPI1  ||  defined  use_comm_MPI2  ||  (!defined
use_comm_MPI1  &&  !defined  use_comm_MPI2  &&  !defined  use_comm_SIPC  &&
!defined   use_comm_GMEM   &&   !defined   use_comm_PIPE   &&   !defined
use_comm_NONE)
c      USE mod_kinds_oasis
c      USE mod_clim
      USE mod_comclim
#include <mpif.h>

      INTEGER (kind=ip_intwp_p)       kinfo
      INTEGER (kind=ip_intwp_p)     ilk, iseg, is, ilgb, ilen, sco_jf
c*    0. Entering
      IF (coupler_rank .EQ. 0) THEN
      kinfo = 0
c
c*    1. check for this port in my list
      iport = -1
c
      IF (myport(1,id_port_id).eq.CLIM_Out) iport=id_port_id
      IF (iport.lt.0) THEN
         kinfo = -1
         WRITE(nulprt,FMT='(A,A)')
     *         'SCO getdimimp - WARNING - Invalid port out: ',
     $        cports(id_port_id)
         GO TO 1020
      ENDIF

c*    2. check for connected ports (in)
      ityp = myport(2,iport)
      DO 290 ip=1,myport(5,iport)
         ilk  = myport(5+ip,iport)
         iseg = mylink(4,ilk)
         ilgb = 0
```

```
         DO 260 is=1,iseg
             ilen = mylink(4+2*is,ilk)
             ilgb = ilgb + ilen
 260      CONTINUE
c
 290  CONTINUE
      sco_ig_nbsend(sco_jf)=ilgb*8
      ENDIF
      call MPI_BCAST( sco_ig_nbsend(sco_jf), 1, MPI_INTEGER,
     $       0, coupler_comm, error )
         IF (error .NE. MPI_SUCCESS) THEN
             CALL prtout('SCO: Error MPIBCAST',1,2)
             CALL HALTE ('STOP in co_getdimexp')
         ENDIF
1020 CONTINUE
#endif
      RETURN
      END
-------------------------------------------------------------------------
```

4. Parallel configuration

In the present section, we describe the changes made on the run script in order to perform parallel jobs (see Appendix A):

- MPISUSPEND environmental variable. During the execution of the whole coupled model, when the atmospheric and ocean models run, coupler processes are waiting, and vice-versa, when coupler runs, models processes are blocked. In order to reduce the number of allocated processors, it is possible to execute both the models and the coupler on the same processors. This can be obtained using the environmental variable MPISUSPEND; setting MPISUSPEND ='on', a blocked process first spin waits for a certain amount of time, after which it suspends itself and releases the CPU resources; the process preserves this state until another process has posted a resume call. The MPISUSPENDCOUNT variable allows specifying the number of unsuccessful attempts before suspending. Setting MPISUSPENDCOUNT=1, coupler processes immediately suspend their activity. Setting MPISUSPEND='on' a degradation of performance should occur. However, available resources are used more efficiently. A more detail performance analysis of pros and cons of using MPISUSPEND can show whenever set it.
- SCRIP files. As already mentioned, each OASIS process handles its own SCRIP file. In the sequential version, the file is created in the working directory and is called *rmp_ATMRES_srcg_to_tgtg_ XXXXXXX.nc* where *srcg* and *tgtg* are the acronyms of respectively the source and the target grids, *XXXXXXX* is the interpolation type (i.e. DISTWGT , GAUSWGT , BILINEA , BICUBIC). In a parallel version, a copy for each parallel OASIS3 process is created using the postfix <*procnum*> at the file name.
- MULTINODE run. There is a section of the run script defining the options for a multimode jobs. An argument string for the execution of models is defined for splitting the processes of parallel model on different nodes. Performed tests have shown some bugs during the definition of the argument string, then some changes over the section have been applied.

5. Functional test

After optimization and parallelization activities, several test have been performed in order to verify the accuracy of achieved results. To this aim, a shell script has been developed; it extracts data from output netcdf files and saves them in a text file, then compares results with original output provided by a sequential run.

Tests have been performed on the four node of NEC SX8 cluster (Figure 3). In particular, both singlenode and multinode configurations have been considered. During the run, a day has been simulated, performing 8 exchanges of the fields between the two models.

The coupled model used to test parallel OASIS3 is characterized by three components: Atmosphere, Ocean and coupler.

- The atmospheric component is ECHAM5 [2]. The release used is the MPI (Message Passing Interface) parallelized version.
- The oceanic model is NEMO [3]

At each temporal iteration, models send their fields to OASIS3 using the PSMILE libraries. The coupler processes the entire dataset and sends new values back to the models.

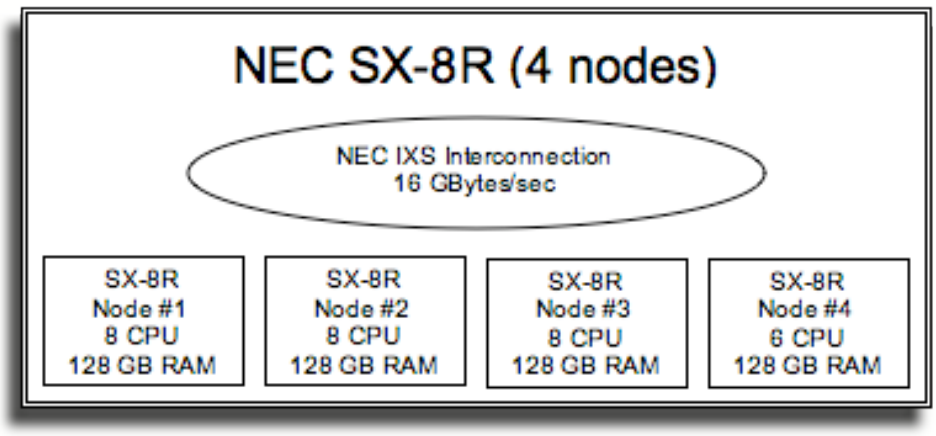

Figure 3 - NEC SX-8R architecture

The following configurations have been tested:

- SINGLENODE run.
  - **Runs characterized by the full sharing of processors**

- ECHAM5 on 2 procs – NEMO on 1 proc - OASIS3 on 3 procs (total procs:3)
- ECHAM5 on 4 procs – NEMO on 1 proc - OASIS3 on 5 procs (total procs:5)
- ECHAM5 on 6 procs – NEMO on 1 proc - OASIS3 on 7 procs (total procs:7)

- **Run characterized by no sharing of processors**
  - ECHAM5 on 2 procs – NEMO on 1 proc - OASIS3 on 3 procs (total procs:6)
  - ECHAM5 on 2 procs – NEMO on 1 proc - OASIS3 on 5 procs (total procs:8)
  - ECHAM5 on 4 procs – NEMO on 1 proc - OASIS3 on 3 procs (total procs:8)
- **Run characterized by partial sharing of processors**
  - ECHAM5 on 2 procs – NEMO on 1 proc - OASIS3 on 7 procs (total procs:8)
  - ECHAM5 on 4 procs – NEMO on 1 proc - OASIS3 on 7 procs (total procs:8)

- MULTINODE run.
  - **Runs characterized by the partial sharing of processors on 3 nodes**
    - ECHAM5 on 8 procs – NEMO on 1 proc - OASIS3 on 3 procs (total procs:9)
    - ECHAM5 on 14 procs – NEMO on 1 proc - OASIS3 on 5 procs (total procs:15)
    - ECHAM5 on 20 procs – NEMO on 21 proc - OASIS3 on 7 procs (total procs:21)

All of the test have been produced the same output compared with sequential output.

Bibliography

[1] Sophie Valcke, *OASIS3 User Guide*, 2006.

[2] http://www.mpimet.mpg.de/en/wissenschaft/modelle/echam/echam5.html

[3] http://www.locean-ipsl.upmc.fr/NEMO/

Appendix A

run.couple – Script to submit the coupled model (ECHAM5 T255L31 – NEMO2 – OASIS3(SCOversion)) simulation on NEC-SX8R vector machine

```
######!###/bin/ksh
……
#PBS -S /bin/ksh
#PBS -q batch
#PBS -l elapstim_req=1000
#PBS -l memsz_job=150gb
#PBS -l cpunum_job=3
#PBS -l cpunum_prc=1
#PBS -b 3
#PBS -T mpisx
#PBS -v ECHAM5_THREADS=0
#PBS -v MPIMULTITASKMIX=ON
#PBS -v MPIPROGINF=ALL_DETAIL
#PBS -v F_PROGINF=DETAIL
#PBS -l tasknum_prc=1
#PBS -j o
…….
#SCO Division has commented out the following line
#export MPISUSPEND=off ;
…….
#
# Number of MPI-processors/openMP-threads
# ---------------------------------------
nprocatm=8
ncplprocatm=1
nthreadatm=0

nprococe=1
ncplprococe=1
nthreadoce=0

nprocice=0
nthreadice=0
ncplprocice=0

(( ntproc = nprocatm + nprococe + nprocice))
#SCO Division: noasis is the number of oasis procs
(( noasis = 3 ))
#SCO Division: nproctot is the total procs number
(( nproctot = ntproc + noasis ))
# SCO Division: Performance improvement can be obtained running oasis on
a single node

……
```

```
# if [ ${jobnum} -gt 1 ]; then
# SCO Division: rmp files must be replicated to avoid concurrent access
by oasis procs
i=0
while [ $i -lt ${noasis} ]; do
     get_file                              ${coupler}                    input
rmp_${pr}${res_atm}_to_or1t_BILINEA.nc
rmp_${pr}${res_atm}_to_or1t_BILINEA_0${i}.nc
     get_file                              ${coupler}                    input
rmp_${pr}${res_atm}_to_or1u_BICUBIC.nc
rmp_${pr}${res_atm}_to_or1u_BICUBIC_0${i}.nc
     get_file                              ${coupler}                    input
rmp_${pr}${res_atm}_to_or1v_BICUBIC.nc
rmp_${pr}${res_atm}_to_or1v_BICUBIC_0${i}.nc
     get_file                              ${coupler}                    input
rmp_or1t_to_${pr}${res_atm}_BILINEA.nc
rmp_or1t_to_${pr}${res_atm}_BILINEA_0${i}.nc
     get_file                              ${coupler}                    input
rmp_or1t_to_${pr}${res_atm}_CONSERV_FRACAREA.nc
rmp_or1t_to_${pr}${res_atm}_CONSERV_FRACAREA_0${i}.nc
     get_file                              ${coupler}                    input
rmp_or1t_to_${pr}${res_atm}_DISTWGT.nc
rmp_or1t_to_${pr}${res_atm}_DISTWGT_0${i}.nc
        i=`expr $i + 1`
done

…….
# Options for multi-node run
#
nprocnode=3
flag0=0
flag1=0
flag2=0
flag3=0
node0="sx00"
proc0=${nprocnode}
node1="sx01"
proc1=${nprocnode}
node2="sx02"
proc2=${nprocnode}
node3="sx03"
proc3=${nprocnode}
#
# Wether to run opa and oasis3 on the same node
#opa_same_oasis="no"
opa_same_oasis="yes"
#
if [ "${multinode}" = "yes" ]; then
  noderun=`echo ${PS1} | cut -d '=' -f2 | tr -d '>'`
  echo "Running on node: ${noderun}"
  nodeavail=`echo ${_MPILNODELIST} | sed -e "s/[0-9]*://g" -e "s/\ \ */
/g" | tr ' ' '\n'`
```

```
  echo "Executing on multiple nodes: ${nodeavail}"
# Echam
  arg=""
  for i in ${nodeavail}; do
    case ${i} in
      ${node0})
       if [ "${noderun}" = "${node0}" ]; then
# SCO Division: procs number must be re-setted, the node can be
considered available two times
        if [ "${flag0}" = 0 ]; then
          proc0=`expr ${proc0} - 1`
          flag0=1
        else
          proc0=${nprocnode}
        fi
       fi
        arg="${arg} _${node0} ${proc0}"
;;
      ${node1})
       if [ "${noderun}" = "${node1}" ]; then
        # SCO Division: procs number must be re-setted, the node can be
considered available two times
        if [ "${flag1}" = 0 ]; then
          proc1=`expr ${proc1} - 1`
          flag1=1
        else
          proc1=${nprocnode}
        fi
       fi
        arg="${arg} _${node1} ${proc1}"
;;
      ${node2})
       if [ "${noderun}" = "${node2}" ]; then
# SCO Division: procs number must be re-setted, the node can be
considered available two times
        if [ "${flag2}" = 0 ]; then
          proc2=`expr ${proc2} - 1`
          flag2=1
        else
          proc2=${nprocnode}
        fi
       fi
        arg="${arg} _${node2} ${proc2}"
;;
      ${node3})
       if [ "${noderun}" = "${node3}" ]; then
# SCO Division: procs number must be re-setted, the node can be
considered available two times
        if [ "${flag3}" = "${nprocnode}" ]; then
          proc3=`expr ${proc3} - 1`
          flag3=1
        else
          proc3=${nprocnode}
```

```
        fi
       fi
        arg="${arg} _${node3} ${proc3}"
;;
         *)
         echo "Wrong node name: ${i}"
         exit
         ;;
    esac
  done
#
  echo "${arg}"
  arg1="${arg}"
#
# OPA
  if [ "${opa_same_oasis}" = "yes" ]; then
    arg2="_${noderun} 1"
  else
    for i in ${nodeavail}; do
      if [ "${noderun}" != "${i}" ]; then
        arg2="_${i} 1"
        break
      fi
    done
  fi
#
  if [ "${arg2}" = "" ]; then
    exit
  fi
#
fi
…..

# SCO Division: MPISUSPEND variable allows oasis procs to be suspended
during models run
MPISUSPEND=ON; export MPISUSPEND
MPISUSPENDCOUNT=1; export MPISUSPENDCOUNT

#set -e
date
if [ ${chan} = MPI2 ]; then
# SCO Division: oasis must be runned on noasis procs
  mpiexec -nn 1 -nnp ${noasis} -max_np ${nproctot} oasis.x    || {
  ls -lta
  exit 1
  }
elif [ ${chan} = MPI1 ]; then
  mpiexec -n 1 oasis.x : -n ${nprocatm} ${atmmod} : -n ${nprococe}
${ocemod}.xx || {
  ls -lta
  exit 1
  }
else
```

```
  echo Invalid channel specified !
  exit 1
fi

…..

# Log files

# SCO Division: all cplout files must be saved
i=0
while [ $i -lt ${noasis} ]; do
save_file             ${coupler}             log             cplout_0${i}
cplout_${date}_${enddate}
i=`expr $i + 1`
done
…..

# Interpolation files
if [ ${jobnum} -eq 1 ]; then
# SCO Division: the rmp file must be renamed
     save_file                          ${coupler}                  input
rmp_${pr}${res_atm}_to_or1t_BILINEA_00.nc
rmp_${pr}${res_atm}_to_or1t_BILINEA.nc
     save_file                          ${coupler}                  input
rmp_${pr}${res_atm}_to_or1u_BICUBIC_00.nc
rmp_${pr}${res_atm}_to_or1u_BICUBIC.nc
     save_file                          ${coupler}                  input
rmp_${pr}${res_atm}_to_or1v_BICUBIC_00.nc
rmp_${pr}${res_atm}_to_or1v_BICUBIC.nc
     save_file                          ${coupler}                  input
rmp_or1t_to_${pr}${res_atm}_BILINEA_00.nc
rmp_or1t_to_${pr}${res_atm}_BILINEA.nc
     save_file                          ${coupler}                  input
rmp_or1t_to_${pr}${res_atm}_CONSERV_FRACAREA_00.nc
rmp_or1t_to_${pr}${res_atm}_CONSERV_FRACAREA.nc
     save_file                          ${coupler}                  input
rmp_or1t_to_${pr}${res_atm}_DISTWGT_00.nc
rmp_or1t_to_${pr}${res_atm}_DISTWGT.nc
fi

……
```