REPORT ON THE PRESENT PARALLELIZATION LEVEL OF ALADIN CODE

G. RADNOTI: 2003/03/16

Preliminary remark: All the code developments and tests below in this report
----
have been performed on AL15_03. A careful phasing will be necessary.

INTRODUCTION

ALADIN has been able to run in parallel mode using
MPI for many years, but there were strong
limitations in its parallelization level with respect to ARPEGE/IFS:

* In ARPEGE/IFS optionally one can use the so called b-level
parallelization. B-level parallelization means that the computational
domain is distributed among processors in a two-dimensional way (in
a-level parallelization all this is much simpler, distribution is done
only in one dimension). The way of this two-dimensional distribution
varies within the time-step and some transpositon routines take care
of going from one kind of distribution to the other. For example in
the gridpoint space fields are distributed in the two horizontal
directions. In the spectral space fields are distributed according to
groups of zonal wavenumbers and along the vertical. In some parts of
spectral computtions (e.g. semi-implict part), however, one needs all
the vertical levels simultaneously, therefore distribution is
transposed such that all vertical level are recollected and
coefficients of the groups of zonal wavenumbers are further
partitioned. The design of this b-level parallelization is suitable
for ALADIN and in most parts of ALADIN code b-level requirements have
been respected. Nevertheless this option has never worked in
ALADIN. As time goes by, situation would become worse and worse because
developpers have no possibility to test their new developments in
b-level environment therefore sometimes they may even fully neglect these
constraints. That is why we have decided that it is high time to catch up
with the parallelization level of the global model.

* Situation is very similar with the so called "lsplit" option, that
intends to provide a perfect load balance for gridpoint computations
both in a-level and in b-level parallelization by allowing to break
the last "latitude-row" of a given a-set and to start the next a-set
from this "breakpoint". An a-set in gridpoint space means a set of
processors that treat the same group of latitude-rows (in a-level
parallelization therefore an a-set is a single processor, in b-level
all the processors working on the same group of latitudes, but over
different longitudinal bends belong to the same a-set).

In 2001 the spectral transformations have been externalized and put
into a separate package (tfl for ARPEGE/IFS and tal for ALADIN). Since
during the transformations model fields "spend some time" in all
possible "model spaces", basically all parallelization related setup has
been moved to the package. In the external package one can run stand
alone tests without any model computations and it gives a possibility
to test the parallelization design in a much simpler
environment. Therefore we decided to make te first step toward b-level
parallelization and lsplit inside the package. If it works, it proves
that the design and setup of b-level parallelization is perfect for
ALADIN and work can be continued on the model side.

MODIFICATIONS AND TESTS ON TAL PACKAGE
----

b-level: After some basic tests it turned out that problems occur only with the momentum fields. This immediately suggested that the specificity of ALADIN, i.e. the mean wind, is most probably not treated in a b-level-conform way. Indeed, vertical distribution was not included in the code for mean wind after reading spectral wind fields and it was incorrectly (for b-level) distributed to all a-sets in direct transforms, where only the a-set holding wavenumber zero computes mean wind and it has to be distributed to all the other a-sets. After correcting these bugs b-level was perfectly working in the tal package.
Modified routines:
testtrans.F90: Main program of stand alone tal tests, vertical distribution of mean winds was included
euvtvd_mod.F90: b-level conform distribution of mean wind within a b-set: all a-set processors within the same b-set (set of vertical levels) must receive mean winds from the processor holding wavenumber 0).
gath_grid_ctl_mod.F90: a small tfl bug that was identified during tests (bug reported to IFS).

lsplit: The fact that lsplit option did not work in the package was in a way surprising because in this respect there seemed to be no principal difference between tfl and tal. However, it was easy to find that some differences exist and they are related to the aladin specific requirement to distinguish between C+I rows and E rows of the domain as far as computational demand is concerned. The tunable parameter called TCDIS is a pre-defined weight factor and in the aladin code it is taken into account when setting up gridpoint space a-sets. This was the only reason why tal and tfl setup differed conceptually. Nevertheless this concept contradicts in lsplit case with some assumptions of the code and this contradictory assumptions (all processors have as much as possible the same number of gridpoints to treat up to the level of division modulus, i.e. a group of processors have one more points than the rest according to the modulus) and these assumptions have never been correctly replaced in aladin. We decided to drop the TCDIS concept which does not only bring us immediatley to a correctly working lsplit option, but also makes the tal setup very close to the tfl one, which is a very convenient advantage.

ezone treatment:

In the previous point i have mentioned the meaning of parameter TCDIS, that we have decided to drop. The solution that we have chosen instead is that we do almost the same distribution strategy as in arpege with the difference that for gridpoint computations, when we distribute latitudes among a-sets we consider NDGUX as total number of latitudes instead of NDGL and at the very end we attach all ezone rows to the last a-set (this means a very slight modification of sumplat, sumplatb and sustaonl==>suemplat, suemplatb,suestaonl). This is the case for gridpoint partitioning. In fourier space, before zonal direct transforms fields are transposed to full latitude bends. Therefore a repartitioning is performed where in principle the latitude set of

each a-set may differ from that of gridpoint partitioning. This is
controlled by sumplatf. (In lsplit case this partitioning necessarily
differs from that of gp space, since in gp-space latitudes are broken,
while in fourier space they must not be). Ezone transforms are not
chaper than C+I ones, so it is reasonable to use in fourier space the
same partitioning as in arpege.i.e. not making difference between
ezone rows and C+I rows in this respect.. It was tested in tal and it works
perfectly.
We have to see that with this new solution even when lsplit=false,
fourier space distribution differs from gridpoint one, i.e.
NDGLL=NDGENL will not be true anymore. Therefore in ALADIN code
one has to be very careful with the correct usage of NDGLLvsNDGENL!

Modified routines:
suemplat_mod.F90, suemplatb_mod.F90: new versions are much closer to tfl
counterparts than before, TCDIS fully dropped.
suemp_trans_mod.F90: calls suetaonl instead of sustaonl,
and sumplatf instead of suemplatf
New routine:
suestaonl_mod.F90: a slightly modified version of sustaonl.F90
removed routine: suemplatf_mod.F90

After all these modifications the transform package works for all
parallelization options. Performance tests are not meaningful with
the package itself because no real gridpoint computations are involved.

Further fix on the package:

Later at ALADIN tests a further small bug was found and corrected on
tfl/tal package in routine inv_trans.F90,einv_trans.F90. Bug has been
reported to IFS, so care is taken on its fix in higher cycles.

MODIFICATIONS AND TESTS ON ALADIN (CONFIGURATION 001)

The fixes performed on the TAL package provided a firm basis for
starting ALADIN tests and modifications. Before starting the testing,
debugging, fixing actions I decided to re-design and recode the coupling
data stream because it was known in advance that the original version
is completely not suitable for b-level and lsplit requirements.

I. Design of a new coupling code (only technical and not scientific aspects)

I.1.Why is the original solution not good?

The way how coupling is performed in the original version of ALADIN
is full of compromises that were always taken when we wanted to quickly
adapt our code to the new environment, mainly coming from ARPEGE
developments. The coupling code was designed in the early times to
act on full latitude rows in a way that I+E-zone points of the given
full latitude are modified by the large scale information. For the above
mentioned reason we always kept this concept, though the present version
of ALADIN code is hardly suitable for such an arrangement. This full-row
concept is not only uncomfortable, but more painfully it gives unavoidable
limits to optimization of performance as far as parallelization and load
balance are concerned. To see these problems we have to understand
a little bit how different partitioning concepts are present in the code.

In gridpoint space in the most general case we have latitudinal and longitudinal partitioning (b-level parallelization) and to have a perfect load balance if lsplit=true we may even break the last latitude of each latitudinal partition (a-set). It is obvious that from such a partitioning it is not straightforward to prepare for coupling if the full-row concept is kept. Fortunetely (now i would rather say unfortunately) we could find a way out: After gridpoint computations first one has to perform direct zonal fourier transforms that also act on full latitudes. In the code it is done by going from the above described gridpoint distribution to fourier space distribution where we have bends of full latitudes and the other direction partitioning is replaced by vertical partitioning. At first sight this organization of arrays is suitable for coupling requirement (full rows are produced) and that is how we made our shortcut solution, but

- we cannot use b-level parallelization since the coupling requires all vertical levels at one processor due to semi-implicit character of coupling
- we are forced to use the same latitude-wise partitioning in gp and fourier space ==> lsplit is out of question
  ==> we cant distinguish computation costs of rows in gp space where there is no cost and in fourier space where cost of direct transforms does not depend on e-zone or c+i zone.
- we were forced to call coupling from the tal package because the the fourier space repartitioning is done there

All these encouraged me to propose a new design. Below i write it down.

I.2. Skeleton of the new design

The natural location of coupling in the code is after the CPG, CPGLAG loops when the GPP (NPROMA,:,NGPBLKS) array is filled with the result of gridpoint computations. Therefore it is natural to couple directly this array. What to do to this end?

I.2.1 SUESC2

The information related to coupling is computed there

In the new plan we should directly couple the GPP(NPROMA,:,NGPBLKS) arrays as they come out from cpg-scanning. More precisely we should at one go collect all I+E-zone points to an array and do the coupling on it.
To make it easy, at the level of suesc2
we have to compute and store:

i)
latitude, longitude dm-local array:

```
NLATGPP(JPROMA,JGPBLKS)
     NLONGPP(JPROMA,JGPBLKS) :
     global latitude and longitude index  of the IPROMA,IGPBLKS element
     of GPP  on the given processor (in the last block, when IGPBLKS=
     NGPBLKS usually there is rubbish in the tail, where we should put
     -99999 both for NLATGPP and for NLONGPP).
```

Computation of NLATGPP,NLONGPP:

```
        NLATGPP(:,:)=-99999
              NLONGPP(:,:)=-99999
        IPROMA=0
              IGPBLKS=1
        DO JGL=NDGSAL,NDGENL
          IGLG=MYLATS(JGL)
          DO JLON=1,NONL((NPTRFLOFF+JGL,MYSETB)
                ILONG=NSTA(NPTRFLOFF+JGL,MYSETB)+JLON-1
            IPROMA=IPROMA+1
                IF (IPROMA > NPROMA) THEN
                  IPROMA=1
                        IGPBLKS=IGPBLKS+1
                ENDIF
                NLATGPP(IPROMA,IGPBLKS)=IGLG
                NLONGPP(IPROMA,IGPBLKS)=ILONG
              ENDDO
            ENDDO
        IF (IGPBLKS /= NGPBLKS) CALL ABOR1("SUESC2: CONFLICT IN NGPBLKS")
```

ii)
latitude, longitude index of each elemnt in GT3BUF
(GT3BUF is the buffer holding
large scale values, but in packed mode, i.e. only I+E-zone
values are stored there:)

```
        NEDLST: nmuber of coupling points on the given processor
                (like before, but computed in simpler way below)
        NLATGT3(NEDLST),NLONGT3(NEDLST):
                global latitude and longitude index array
          of the couling points:
          NEDLST=0
        DO JGL=NDGSAL,NDGENL
              IGLG=MYLATS(JGL)
              DO JLON=1,NONL(NPTRFLOFF+JGL,MYSETB)
                ILONG=NSTA(NPTRFLOFF+JGL,MYSETB)&
             &+JLON-1
             IF (ILONG.LE.NBZONL.OR.ILONG.GT.NDLUXG-NBZONL.&
               &OR.IGLG.LE.NBZONG.OR.IGLG.GT.NDGUXG-NBZONG) THEN
!               point is outside C-zone==>it shouldbe coupled
               NEDLST=NEDLST+1
                       NLATGT3(NEDLST)=IGLG
                       NLONGT3(NEDLST)=ILONG
                 ENDIF
               ENDDO
             ENDDO
```

iii)
The EALFA coupling coefficient array is computed in SUEBICU
(suebicu is called before suesc2 so ealfa
is known at this stage). However, EALFA is an NGPTOT array.

Remark:
(ATTENTION: ealfa is initialized to NBDYSZ
size, but i doubt that it has an acceptable reason.
Someone should revise the use of NBDYSZ, NBDYSZG
in ALADIN and replace by NGPTOT,NGPTOTG wherever it is possible!!!!!!!!
Moreover, the initialization
of NBDYSZ is completely wrong in SUEGEO2 if we keep in mind LSPLIT,B-LEVEL
options!!!! This should be
also revised!!!!!!! In the arpege counterpart of AL15 NBDYSZ and NGPTOT
are the same.(??) NGPTOT
is coming from tfl, NBDYSZ from sugem1b, but they are set the same way.

As i could see later on, in newer cycles NBDYSZ has been pruned, so
probably my previous comments are right.)

We should perhaps not drop this EALFA(NGPTOT,:) definition because EALFA
is or can be used elsewhere, but
here in suesc2 we should copy the relevant part of ealfa to an EALFAGT3,
i.e. only the non-zero ealfa
coefficients ordered in the same way as the packed large scale values.
Coupling needs
map factor as well(for semi-implicit part). So we introduce a GMGT3
to capture relevant part of GM. :

```
        IGPTOT=0
             IDLST=0
        DO JGL=NDGSAL,NDGENL
          IGLG=MYLATS(JGL)
          DO JLON=1,NONL(NPTRFLOFF+JGL,MYSETB)
            ILONG=NSTA(NPTRFLOFF+JGL,MYSETB)+JLON-1
            IGPTOT=IGPTOT+1
            IF (ILONG.LE.NBZONL.OR.ILONG.GT.NDLUXG-NBZONL.OR.IGLG.LE.&
             &NBZONG.OR.IGLG.GT.NDGUXG-NBZONG) THEN
!                point is outside C-zone==>it shouldbe coupled
               IDLST=IDLST+1
               EALFAGT3(IDLST,:)=EALFA(IGPTOT,:)
                        GMGT3(IDLST)=GM(IDLST)
            ENDIF
          ENDDO
        ENDDO
```

iv)
The allocation part of the GT3BUF can remain basically as it was,
just use the above computed NEDLST.

v)
Prune the unnecessary variables and their computation part:
 NBZONLW ,NBZONLE ,NELOEN   ,NBZONC
Apart from the standard coupling routines that will follow this structure,
the pruned variables are
used only in ERDLSGRAD, consultation with Claude is necessary to adjust it
to the new structure.

I.2.2 Filling GT3BUF (EPAK3W)

Filling coupling buffer is done in EPAK3W

The calling tree is:

```
CNT3--->ELSAC
                        ---->     ELSWA3-->EPAK3W
CNT4,EDFI3---->ELSRW--->ERLBC
```

In these calling trees only ELSWA3 and EPAK3W have to be rewritten at this
stage of cleaning:

ELSWA3:

Up to the level where ZGT3 and ZGP are allocated, nothing has to change.
The small part calling epak3w has to be rewritten like:

ZGP is unneeded

```
   ALLOCATE(ZGT3(NEDLST*IGT0))
   IND=0
   DO JGPBLKS=1,NGPBLKS
   DO JPROMA=1,NPROMA
     IF (NLATGPP(JPROMA,JGPBLKS) > 0) THEN
       ILATG=NLATGPP(JPROMA,JGPBLKS)
            ILONG=NLONGPP(JPROMA,JGPBLKS)
       IF (ILONG.LE.NBZONL.OR.ILONG.GT.NDLUXG-NBZONL.OR.&
       &IGLG.LE.NBZONG.OR.IGLG.GT.NDGUXG-NBZONG) THEN
         IND=IND+1
              ZGT3((IND-1)*IGT0+1:IND*IGT0)=GPP(JPROMA,1:IGT0,JGPBLKS)
       ENDIF
     ENDIF
   ENDDO
   ENDDO
   CALL EPAK3W(ZGT3,IFLDSGT0,.TRUE.)
   DEALLOCATE(ZGT3)
```

EPAK3W:

According to the modifications in ELSWA3 the argument list will be

```
   EPAK3W(PDATA,KFIELDS,LDGP)
```
where
```
   REAL_B,    INTENT(IN) :: PDATA(NEDLST*KFIELDS)
```
and we need a
```
   REAL_B, ALLOCATABLE :: Z00(:)
```
The part to be rewritten is the IF (LDGP) part:
All the JAREA, IZGT part is unneded, all NEDLST points are treated in one go
What will remain from the LDGP part:

```
      IF (LQCPL) THEN
        IF (NDD01 == 1) THEN
          GT3BUF(ISWP1+1:ISWP1+NEDLST*KFIELDS)=PDATA(1:NEDLST*KFIELDS)
        ELSEIF (NDD01 == 2) THEN
          GT3BUF(ISWP3+1,ISWP3+NEDLST*KFIELDS)=(PDATA(1:NEDLST*KFIELDS)&
          &-GT3BUF(ISWP1+1:ISWP1+NEDLST*KFIELDS))*ZRVFRCL
          GT3BUF(ISWP2+1:ISWP2+NEDLST*KFIELDS))=0
        ELSEIF (NDD01 == 0) THEN
              ALLOCATE(Z00(NEDLST*KFIELDS))
          Z00(1:NEDLST*KFIELDS) = GT3BUF(ISWP2+1:ISWP2+NEDLST*KFIELDS)&
          &*ZSQFRCL &
           &+ GT3BUF(ISWP3+1:ISWP3+NEDLST*KFIELDS)*ZFRCL &
           &+ GT3BUF(ISWP1+1:ISWP1+NEDLST*KFIELDS)
          GT3BUF(ISWP3+1:ISWP3+NEDLST*KFIELDS)=&
           &_HALF_*ZRVFRCL*(PDATA(1:NEDLST*KFIELDS)-&
           &GT3BUF(ISWP1+1,ISWP1+NEDLST*KFIELDS))
          GT3BUF(ISWP1+1:ISWP1+NEDLST*KFIELDS)=Z00(1:NEDLST*KFIELDS)
          GT3BUF(ISWP2+1:ISWP2+NEDLST*KFIELDS)=&
           &(PDATA(1:NEDLST*KFIELDS)-GT3BUF(ISWP1+1:ISWP1+NEDLST*KFIELDS)&
           &-GT3BUF(ISWP3+1:ISWP3+NEDLST*KFIELDS)*ZFRCL &
           &)*ZRVSQFRCL
          DEALLOCATE(Z00)
        ELSE
          CALL ABOR1('EPAK3W : INTERNAL ERROR NDD01')
        ENDIF
      ELSE
```

```
  GT3BUF(ISWAP+1:ISWAP+NEDLST*KFIELDS)=PDATA(1:NEDLST*KFIELDS)
 ENDIF
```

I.2.3 Coupling itself

According to all above call of coupling will not be done from
tal so all tal related coupling stuff
has to be removed from the package and its interfaces:
LDCPL has to be removed from the routines in ald/transform and
down from the called package routines;
Same for the CPL_PROC argument.

ECOUPL1 has to be called either from scan2mdm or from stepo.
The former corresponds
to the solution up to A12 the latter is closer to the AL15 solution.
Since the al15 solution seemed to be safe for all
the configurations I would recommend
and here i will develop the stepo solution. However it can be consulted.
(If scan2mdm is chosen, the al12 solution would be to call coupling
before session " WRITE OUT UPPER AIR GRID-POINT DATA").
The stepo solution is to call coupling just before etransdirh:
-remove the llcpl argument from etransdirh and introduce:

```
  IF (LLCPL) CALL ECOUPL1
```

ECOUPL1 should be without any argument, from the global
environment it must get
all the information needed. (1 stands for timelevel t1,
if we decide to forget
t0 coupling it would be nice to rename the ecoupl1,elscot1,esrlxt1
sequence to
ecoupl,elsco,esrlx).
Let's see the new ecoupl(1)

```
SUBROUTINE ECOUPL1
! Bla-bla-bla
USE ....(see from code below what is needed)
local declarations...(see from code below what is needed,
partly what is in old ecoupl1)
.....
CALL  SC2CGAP(IVORT1,IDIVT1,IUT1,IVT1,IUT1L,IVT1L &
         &,IHVT1,ITT1,IQT1,IO3T1,ILT1,IIT1,ISPD1,ISVD1 &
              &,IHVT1L,ITT1L,IQT1L,IO3T1L,ILT1L,IIT1L,ISPD1L,ISVD1L &
         &,IHVT1M,ITT1M,IQT1M,IO3T1M,ILT1M,IIT1M,ISPD1M,ISVD1M &
         &,ISVT1,ISVT1L,ISVT1M,ISPT1,ISPT1L,ISPT1M,INUL,IAT1,INUL &
         &,INUL,INUL,IFLDSGT1,IFLDSFLT,.FALSE.,.FALSE.,.FALSE.&
         &,1,NFLEVG)

ALLOCATE(ZGT1(NEDLST,IFLDSGT1))
IND=0
DO JGPBLKS=1,NGPBLKS
DO JPROMA=1,NPROMA
    IF (NLATGPP(JPROMA,JGPBLKS) > 0) THEN
      ILATG=NLATGPP(JPROMA,JGPBLKS)
      ILONG=NLONGPP(JPROMA,JGPBLKS)
      IF (ILONG.LE.NBZONL.OR.ILONG.GT.NDLUXG-NBZONL.OR.IGLG&
      &.LE.NBZONG.OR.IGLG.GT.NDGUXG-NBZONG) THEN
           IND=IND+1
```

```
        ZGT1(IND,1:IFLDSGT1)=GPP(JPROMA,1:IFLDSGT1,JGPBLKS)
      ENDIF
    ENDIF
ENDDO
ENDDO
CALL ELSCOT1(GMGT3,EALFAGT3,&
         &,ZGT1(1,IUT1),ZGT1(1,IVT1),ZGT1(1,ITT1)&
         &,ZGT1(1,IQT1),ZGT1(1,IO3T1),ZGT1(1,ILT1),ZGT1(1,IIT1)&
         &,ZGT1(1,ISPD1),ZGT1(1,ISVD1),ZGT1(1,ISVT1)&
         &,ZGT1(1,ISPT1))
! Orography was passed but never used, so pruned here
DO JGPBLKS=1,NGPBLKS
DO JPROMA=1,NPROMA
    IF (NLATGPP(JPROMA,JGPBLKS) > 0) THEN
       ILATG=NLATGPP(JPROMA,JGPBLKS)
       ILONG=NLONGPP(JPROMA,JGPBLKS)
       IF (ILONG.LE.NBZONL.OR.ILONG.GT.NDLUXG-NBZONL.OR.IGLG.&
       &LE.NBZONG.OR.IGLG.GT.NDGUXG-NBZONG) THEN
         IND=IND+1
         GPP(JPROMA,1:IFLDSGT1,JGPBLKS)= ZGT1(IND,1:IFLDSGT1)
       ENDIF
    ENDIF
ENDDO
ENDDO
END SUBROUTINE ECOUPL1
```

elscot1:

```
SUBROUTINE ELSCOT1 (PGM,PALFA,&
 &PUT1,PVT1,PTT1,PQT1,PO3T1,PLT1,PIT1,PSPD1,PSVD1,PSVT1,PSPT1)
! POROG pruned, KGL disappeared (all NEDLST points treated in one go)
```

Otherwise everything untouched just NEDLST instead of NDLON and NONL
remove KGL from argument list of ESC2R

esc2r:

```
SUBROUTINE ESC2R(KTIMLEV,LDGP,KFIELDS,PDATA,KDIM)
! KGL disappeared
```

Rewriting exactly the same way as epak3w:
no need of JAREA business, everything goes
from 1 to NEDLST with implicit loopig.
One difference to what was done in epak3w:
for fields we can not do implicit looping
here, because PDATA is 2 dimensional array (KDIM,KFIELDS) and GT3BUF is 1D==>
for fields we need an explicit loop with defining offsets, i.e.
the whole part after "time management" has to remain in the
  DO JF=1,KFIELDS loop
Then e.g.
the LQCPL part will read as

```
PDATA(1:NEDLST,JF)==ZWB0*GT3BUF(ISWP0+(JF-1)*NEDLST+1:ISWP0+JF*NEDLST)&
&+ZWB1*GT3BUF(ISWP1+(JF-1)*NEDLST:ISWP1+JF*NEDLST)&
&+ZWB2*GT3BUF(ISWP2+(JF-1)*NEDLST:ISWP2+JF*NEDLST)
```

Similar rewriting for the other cases has to be done like in epak3w.
(So again the whole IGT3,IZGT shift is removed, i
everything is done in contiguous
way on the NEDLST set, field-wise)

esrlxt1:
no change is needed at all!!!!!!!!

deello.F90:
Deallocation of arrays has to be adjusted to new and pruned arrays.

I.2.4 Memory considerations

When all the above is coded and works one has to consider the memory overhead. In the earlier solution all the coupling was done on NDLON
slices (with the limitations described
in the introduction). The above design works on the whole NEDLST piece.
In DM environment
with many processors this is not more expensive in terms of memory. However, with small processor number it may become more expensive.
If it is the case, it is not too dificult
to introduce chunk loops both for the call of epak3w and for ecoupl1.
Certainly the loop
in this case has to do chunks according to NEDLST and not according to fields.

After these modifications tests were done: in a-level the code works, gives identical results with old version. b-level tests need other debugging of ALADIN (see later in this document).

I.3 Important Remarks

Adjoint aspects were not elaborated above,
when everything works in the direct code,
some adjoint expert should be involved.

When phasing the code according to the
above it is worthwhile to do other cleaning-pruning
as well. To consider e.g. if the spectral field
treatment for RUBC should remain in the
coupling code or it is obsolete. I hope it can be dropped,
because i have not elaborated the LRUBC related modifications.

ERDLSGRAD was fully dropped because it uses the old coupling setup and
it did not take into account b-level constraints at the dm-distribution
involved in the routine. It should be rewritten.

The call of coupling is removed from the tranform package. As a short-cut
solution i pass from stepo to etransdirh llcpl=.false.. It would be nice
to prune all the llcpl related stuff from the package.

II. Debugging of ALADIN in b-level with new coupling

After rewriting aladin coupling in the way described above i could
start b-level and lsplit tests on aladin.

Below i list the additional modifications that i had to introduce to make ALADIN
configuration 001 work in b-level/lsplit mode:

espconvert.F90: mis-use of NFLEVL<-->NFLEVG

espuv.F90: initializing IVSETUV and introducing it to calling sequence of EINV_TRANS
sueorog.F90,suecuv.F90: initializing IVSETSC and introducing it to calling sequence
    of EINV_TRANS,EDIR_TRANS

euvgeovd.F90: mis-use of KLEV<-->KFLSUR + introducing kvsetuv
disgrid.F90: Arpege bug, reported, care taken by GMAP
etrmtos.F90: "mis-typing errors", trivial fixes.
sump0.F90: remove protection of aladin lsplit
wrmlppadm.F90: mis-use of NFLEVL<-->NFLEVG
ewrplppdm.F90: introducing kvsetuv
suegeo2.F90: wrong size and definition of NLOEN + forcing NBDYSZ=NGPTOT (see earlier remark)
suemp.F90: 2 fixes:

- care had to be taken that when spectral a-sets are further splitted for SI computations, the breaking of NSPEC2 arrays into several NSPEC2V arrays should not break inside a wavenumber pair, i.e. the 4 coefficients of a wavenumber pair should remain on the same processor.
- wrong definition of NDGUXL

espnormave.F90: removing mean wind from norm computation: if it is to be present, code for distribution of mean wind for diagnostic purpose has to be elaborated.
suspec.F90 suspeca.F90,suspeca.h: AL15 code is bugged, corrected suspeca was taken from higher cycles.
Furthermore, explicit array bounds had to be removed from dummies to provide norm identity after suspeca (a general little bug!!!!)

PERFORMANCE TESTS:

After all the above described modification b-level and lsplit options worked
in configuration 001. Norm identity was guaranteed up to 6 digits in a 6 hour integration.

I made some performance tests with NPROC=16,NPRGPNS=NPRGPEW=NPRTRW=NPRTRV=4
on a domain with NFLEVG=31,NDLON=240,NDGL=216

The b-level option in this configuration brings a solid 10 percent performance
improvement. LSPLIT does not make any measurable change in performance. The
eefectivity of LSPLIT highly depends on the NDLUXG/NPRGPNS ratio and modulus,
so tests should be repeated with several such configurations.