Implementation of the LINK DECO BLOQ directive under MPI in EUROPLEXUS

Casadei, F.
Larcher, M.
Valsamos, G.
Faucher, V.

2018
Implementation of the LINK DECO BLOQ directive under MPI in EUROPLEXUS

F. Casadei\textsuperscript{1}, M. Larcher\textsuperscript{2}, G. Valsamos\textsuperscript{2}, and V. Faucher\textsuperscript{3}

\textsuperscript{1}Retired from JRC ELSA
\textsuperscript{2}European Laboratory for Structural Assessment (ELSA), Institute for the Protection and Security of the Citizen (IPSC) Joint Research Centre (JRC), 21027 Ispra, Italy
\textsuperscript{3}Département Technologie Nucléaire, CEA Cadarache, DEN/CAD/DTN/Dir

September 18, 2017

Contents

1 Introduction 2

2 The BLOQ link 2
  2.1 Coupled blockage constraint ................................................. 2
  2.2 Decoupled blockage constraint ............................................. 3

3 Implementation 3
  3.1 Sequential version ............................................................. 3
  3.2 MPI version ................................................................. 3
    3.2.1 Module M\_LINK\_BLOQ\_DECO\_DATA ..................................... 4
    3.2.2 Module M\_LINK\_BLOQ .................................................. 4
    3.2.3 Subroutines CALCUL and TLOOPP ..................................... 5
    3.2.4 Module M\_DOMAINE ..................................................... 5
    3.2.5 Subroutine DOMDEC\_SPLIT\_DATA ..................................... 6
    3.2.6 Subroutine D\_BLOQ\_DECO\_SPLIT ..................................... 6
    3.2.7 Subroutine D\_CALFREE ............................................... 7
    3.2.8 Subroutine SOLVE\_BLOQ\_DECO ....................................... 8

4 Numerical examples 10
  4.1 Sequential examples ......................................................... 10
    4.1.1 Case DBLO01 .............................................................. 10
  4.2 MPI examples ............................................................... 11
    4.2.1 Case DBLO02 .............................................................. 11

References 11

Appendix I — Input files 13

List of input files 14

List of Tables

1 Numerical simulations with the sequential version of EPX .......................... 10
2 Numerical simulations with the MPI version of EPX (with 4 processes.) .......... 11
1 Introduction

This report presents the implementation of the LINK DECO BLOQ directive under domain decomposition (MPI version) in the EUROPLEXUS code.

EUROPLEXUS [1] (also abbreviated as EPX) is a computer code jointly developed by the French Commissariat à l’Énergie Atomique (CEA DMT Saclay) and by EC-JRC. The code application domain is the numerical simulation of fast transient phenomena such as explosions, crashes and impacts in complex three-dimensional fluid-structure systems. The Cast3m [2] software from CEA is used as a pre-processor to EPX when it is necessary to generate complex meshes.

The present report is a sequel to reference [3], which described the porting to MPI of the LINK DECO RIGI directive. The same approach as in the case of the RIGI directive is used here, and the implementation is even simpler than in the case of RIGI because no ad-hoc communications (exchange of data) between the various processors are needed to impose plain blockages.

2 The BLOQ link

The link of type BLOQ imposes that a chosen displacement component of a node or set of nodes be zero, i.e. that the node or set of nodes is blocked along the chosen direction.

Like several other types of links in EPX, the constraint can be imposed either in a coupled (LINK COUP) or in a decoupled (LINK DECO) manner. The first form uses an implicit Lagrange multipliers formulation, while the second form uses an explicit formulation, whereby the reaction forces are computed directly. The syntax of the BLOQ command is

\[
\text{BLOQ ( /LECDDL/ /LECTURE/ )}
\]

where /LECDDL/ denotes one or more (local) degrees of freedom and /LECTURE/ denotes one or more nodes. The brackets indicate that the enclosed sequence may be repeated as many times as necessary to define all the needed blockages.

2.1 Coupled blockage constraint

When the coupled form of the directive (LINK COUP) is used, the code imposes the following constraint:

\[
v^n_{i} + \frac{1}{2} v_{i}^{n+1} = 0
\]  

(1)

where \(v\) is the velocity vector, \(I\) is the node index, \(i\) is the local degree of freedom: \(i = 1, \ldots, d\) for translations, with \(d\) the space dimension (which is either 2 or 3) and \(i = d+1, \ldots\) for rotations, if the node has rotational dofs. The superscript \(n\) refers to the time station in the discrete time marching algorithm (central difference) used by EPX.

If the (formally coupled) link happens to be independent of any other links, the solution of eq. (1) leads to the following reaction force:

\[
F_{i}^{\text{ext}} = F_{i}^{\text{int}} - \phi m_{i} v_{i}
\]  

(2)

where \(F_{i}^{\text{ext}}\) is the external force, \(F_{i}^{\text{int}}\) the internal force, \(m\) is the mass, and the coefficient \(\phi\) is given by:

\[
\phi = \frac{2}{\Delta t_{n} + \Delta t_{n+1}}
\]  

(3)
for a generic time step \((n > 0)\), or:

\[
\phi = \frac{2}{\Delta t^{n+1}}
\]

at the initial time \((n = 0)\), with \(\Delta t^n = t^{n+1} - t^n\).

### 2.2 Decoupled blockage constraint

When the decoupled form of the directive (\texttt{LINK DECO}) is used, the complementary force (i.e., the reaction) is computed directly by:

\[
F^c_{Ii} = F^{\text{int}}_{Ii} - F^{\text{ext}}_{Ii}
\]

and then it is added to the external force:

\[
F^{\text{ext}}_{Ii} \leftarrow F^{\text{ext}}_{Ii} + F^c_{Ii}
\]

Note that in applying eqs. (5-6) we tacitly assume that the node has zero (initial) velocity.

### 3 Implementation

#### 3.1 Sequential version

Like for many other models, the \texttt{BLOQ} directive has been first implemented in the sequential version of \texttt{EPX}, i.e. without any domain decomposition. The model basically consists of a fortran 90 module \texttt{M\_LINK\_BLOQ} that contains a simple data structure (used only in the decoupled case):

```fortran
LOGICAL :: HAS_BLOQ_DECO ! ARE THERE ANY BLOQ DECO ??? INTEGER, POINTER :: IS_BLOQ_DECO(:) ! 1 IF DOF IS BLOQ DECO, ELSE 0
```

plus some routines, among which the most important ones are the routine to read the directive from the input file (\texttt{READ_BLOQ}) and the routine to solve the imposed links (\texttt{SOLVE_BLOQ_DECO}) when the decoupled form of the link is chosen, which is described in Section 2.2. Note that, when the coupled form is selected, no specific solving routine is strictly necessary since in that case the links (eqs. 1-6) are solved by the general coupled links formulation of \texttt{EPX}. However, a specific routine \texttt{SOLVE_SINGLE_BLOQ} is included in the module, which solves the (formally coupled) blockage when it is actually independent from any other constraints.

The above listed data structure is not necessary in the coupled case because in that case the links (eqs. 1-4) are directly created by the syntax reading routine (\texttt{READ_BLOQ}) and are added to the system of (coupled) links of the problem. In the decoupled case, the flag \texttt{HAS_BLOQ_DECO} is set to \texttt{.TRUE.} and the variable \texttt{IS_BLOQ_DECO} is set to 1 for all the concerned dofs, while it remains 0 for all other dofs.

The data structure for decoupled \texttt{BLOQ} links must be suitably initialized, saved and restored in case of splitting of the calculation, and destroyed at the end of a calculation. These tasks are performed by routines \texttt{INIT_BLOQ_DECO}, \texttt{SAVE_BLOQ_DECO}, \texttt{RESTORE_BLOQ_DECO} and \texttt{DESTROY_BLOQ_DECO}, respectively.

#### 3.2 MPI version

Porting the \texttt{BLOQ} model to MPI (domain decomposition) did not require any particular development for the \texttt{coupled} version of the link, since this is based on the general (coupled) links data structure of \texttt{EPX}, that is already fully compatible with MPI.

For the \texttt{decoupled} version, however, some developments were necessary, very similar to (actually simpler than) those described in reference [3] for the \texttt{LINK DECO RIGI} directive. The data structure specific to \texttt{LINK DECO BLOQ} described in Section 2 had to be extended to the multi-domain case and the routine \texttt{SOLVE_BLOQ_DECO} implementing the direct solution algorithm of Section 2.2 had to be generalized so as to include also the case with several sub-domains.
The complete set of modifications is presented and shortly discussed hereafter. This may be used as a guide for the porting to MPI of other similar decoupled link directives, or as a starting point for the parallelization of more complex models in EPX.

The first operation to do is adding the specific data structure of the decoupled BLOQ links, i.e. the array \texttt{IS\_BLOQ\_DECO}, to the sub-domains data structure, that is to the \texttt{TYPE DOMAINE} in module \texttt{M\_DOMAINE}. In this way, each sub-domain will have its list of decoupled-blocked dofs. Each list for any given sub-domain will contain only the dofs that belong to this sub-domain, either internal or on the interface with other neighbour sub-domains. The flag \texttt{HAS\_BLOQ\_DECO} is a global quantity (the same for all sub-domains) and needs not be duplicated.

But before doing that, we split the former \texttt{M\_LINK\_BLOQ} module, which contained both the data and the procedures (subroutines), into two modules, one containing only the data (a newly created module \texttt{M\_LINK\_BLOQ\_DECO\_DATA}) and the other containing only the procedures (\texttt{M\_LINK\_BLOQ}.) This distinction is always advisable, since it helps to avoid cycling in the \texttt{USE} of modules, which could result from the subsequent modifications.

Furthermore, the subroutine \texttt{SOLVE\_BLOQ\_DECO} is extracted from the previous module \texttt{M\_LINK\_BLOQ} and placed in a separate program file.

### 3.2.1 Module \texttt{M\_LINK\_BLOQ\_DECO\_DATA}

The new data-only module \texttt{M\_LINK\_BLOQ\_DECO\_DATA}, resulting from splitting of the former \texttt{M\_LINK\_BLOQ} module, reads:

```fortran
MODULE M_LINK_BLOQ_DECO_DATA
*
  * data related to the link bloq condition (deco only)
  *
  USE M_ALLOCATION
  *
  IMPLICIT NONE
  *
  PRIVATE
  *
  * public data
  PUBLIC ::
    > HAS_BLOQ_DECO,
    > IS_BLOQ_DECO
  *
  LOGICAL :: HAS_BLOQ_DECO ! ARE THERE ANY BLOQ DECO ???
  INTEGER, POINTER :: IS_BLOQ_DECO(:) ! 1 IF DOF IS BLOQ DECO, ELSE 0
  *
END MODULE M_LINK_BLOQ_DECO_DATA
```

### 3.2.2 Module \texttt{M\_LINK\_BLOQ}

The procedures-only module \texttt{M\_LINK\_BLOQ} reads:

```fortran
MODULE M_LINK_BLOQ
*
  * procedures related to the link bloq condition (coup or deco)
  *
  USE M_ALLOCATION
  USE M_LINK_BLOQ_DECO_DATA ! data for the deco version of this link
  *
  IMPLICIT NONE
  *
  PRIVATE
  *
  * public procedures
  PUBLIC :: READ_BLOQ,
    > ADD_BLOCKAGE,
    > ADD_BLOCKAGE_DECO,
    > INIT_BLOQ_DECO,
    > SAVE_BLOQ_DECO,
    > RESTORE_BLOQ_DECO,
    > DESTROY_BLOQ_DECO,
```
The USE of the new data module M_LINK_BLOQ_DECO_DATA has been added, as highlighted in red above. Furthermore, the subroutine SOLVE_BLOQ_DECO has been removed from the module and is now placed in a separate file.

3.2.3 Subroutines CALCUL and TLOOPP

In the CALCUL subroutine, the former USE statement (in CALCUL_LINK_DECO):

USE M_LINK_BLOQ, ONLY:HAS_BLOQ_DECO, SOLVE_BLOQ_DECO

is replaced by:

USE M_LINK_BLOQ_DECO_DATA, ONLY: HAS_BLOQ_DECO

since the routine SOLVE_BLOQ_DECO is now in a stand-alone file. Similarly, in the TLOOPP routine, the statement:

USE M_LINK_BLOQ

is replaced by:

USE M_LINK_BLOQ_DECO_DATA

3.2.4 Module M_DOMAINE

In the module M_DOMAINE we add the following statements (in red):

MODULE M_DOMAINE

USE M_LINK_BLOQ_DECO_DATA

TYPE DOMAINE

* link bloq deco variables
  INTEGER, POINTER :: IS_BLOQ_DECO(:) ! 1 IF DOF IS BLOQ DECO, ELSE 0

END TYPE DOMAINE

SUBROUTINE CREE_DOMAINE

NULLIFY(DOM_CUR%IS_BLOQ_DECO)

END SUBROUTINE CREE_DOMAINE

SUBROUTINE DEL_DOMAINE

* 41 bloq deco
  IF (ASSOCIATED (DOM_CUR%IS_BLOQ_DECO)) THEN
    DEALLOCATE (DOM_CUR%IS_BLOQ_DECO)
    HAS_BLOQ_DECO = .FALSE.
  ENDIF

END SUBROUTINE DEL_DOMAINE

END MODULE M_DOMAINE
An INTEGER array IS_BLOQ_DECO is added to the derived type DOMAINE, and so becomes part of each sub-domain’s data structure. The array is nullified in the subroutine CREE_DOMAINE. Then, in the subroutine DEL_DOMAINE, which takes care of the disposal of the data structure of each sub-domain (in a loop over all sub-domains), it is necessary to add the de-allocation of the data structure (note that also the global flag HAS_BLOQ_DECO is reset to .FALSE.)

### 3.2.5 Subroutine DOMDEC_SPLIT_DATA

The subroutine DOMDEC_SPLIT_DATA splits the global data structure into the various sub-domains. We add the following statements (in red) in order to split the data structure of interest here:

```fortran
SUBROUTINE DOMDEC_SPLIT_DATA ( . . . )
   USE M_LINK_BLOQ_DECO_DATA
   IF (HAS_BLOQ_DECO) THEN
      CALL D_BLOQ_DECO_SPLIT (IMPI)
      CALL PRINT_CPU_CLOCK(2, 'SPLIT BLOQ DECO')
   ENDIF
END SUBROUTINE DOMDEC_SPLIT_DATA
```

As it can be seen, the splitting task is actually demanded to a (new) lower-level subroutine D_BLOQ_DECO_SPLIT. The call to PRINT_CPU_CLOCK is used to print on the listing the CPU time used in this operation.

### 3.2.6 Subroutine D_BLOQ_DECO_SPLIT

The new subroutine D_BLOQ_DECO_SPLIT reads:

```fortran
SUBROUTINE D_BLOQ_DECO_SPLIT (IMPI)
   USE MDOMAIN
   USE M_LINK_BLOQ_DECO_DATA
   USE M_LINK_BLOQ ! for destroy_bloq_deco
   IMPLICIT NONE
   INTEGER, INTENT(IN) :: IMPI
   INTEGER :: JJ, JJ_L, NLIBI
   IF (IMPI /= 1) THEN
      CALL ERRMSS ('D_BLOQ_DECO_SPLIT',
                  'LINK DECO BLOQ N/A WITH SEQUENTIAL DOMAIN DECOMPOSITION')
      STOP 'D_BLOQ_DECO_SPLIT: LINK DECO BLOQ N/A WITH SEQ. DOM. DEC.'
   ENDIF
   NLIBI = DOMAIN(1)%NLIBI ! N. of dofs of this sub-domain
   ALLOCATE (DOMAIN(1)%IS_BLOQ_DECO(NLIBI))
   DO JJ_L = 1, NLIBI ! Loop on local dofs
      JJ = DOMAIN(1)%CONVERS%DOF_LG(JJ_L) ! Global dof corresponding to jj_l
      DOMAIN(1)%IS_BLOQ_DECO(JJ_L) = IS_BLOQ_DECO(JJ)
   ENDDO
   END SUBROUTINE D_BLOQ_DECO_SPLIT
```
Before discussing this subroutine in detail, it is essential to clarify the meaning of \texttt{DOMAIN(1)} in the above code. In fact, it should be noted that the subroutine \texttt{D_RIGI_DECO_SPLIT} is called by (executed on) each MPI process (i.e., for each sub-domain) separately, and not, as one might perhaps expect from its name, only on the first processor and treating all sub-domains. Therefore, in the above code \texttt{DOMAIN(1)} represents the \textit{current} sub-domain (the one associated with the current processor), and \textit{not the first} sub-domain of the decomposition, as one might think at first sight.

It turns out in fact that on the first processor (processor 0 in MPI) \textit{all} sub-domains are known (\texttt{NBDOMAINES} is equal to the number of processors used in the MPI execution), so \texttt{DOMAIN(1)} is indeed the current sub-domain for this processor. But, on \textit{all other} processors, only the “local” sub-domain is known (\texttt{NBDOMAINES} is equal to 1) so that, again, \texttt{DOMAIN(1)} is the current sub-domain (the sub-domain associated with the current processor.)

It should also be mentioned in passing that the possibility of having several domains inside one process is a result of the original implementation of domain decomposition for \textit{sequential} computing. This is why the one-dimensional array \texttt{DOMAIN(:)} is present in the implementation, even though only \texttt{DOMAIN(1)} is basically used with MPI. By the way, even with MPI, the process with rank 0 uses light versions of sub-domains attributed to other processes (\texttt{DOMAIN(I), I > 1}) for outputs gathering purposes.

Note also that in the current implementation, a test producing an error if the \texttt{IMPI} variable is not 1 has been added in the splitting routine, preventing the wrong use of \texttt{DECO BLOQ} links with the sequential domain decomposition.

Basically, the routine copies the global data structure to the \textit{current} sub-domain (\texttt{DOMAIN(1)}), with suitable adaptations. First, a local table \texttt{DOMAIN(1)%IS_DECO} is allocated, with a length \texttt{DOMAIN(1)%NLIBI}, i.e. equal to the number of dofs of the current sub-domain. Then, the value of \texttt{IS_BLOQ} for each local dof \texttt{JJ_L} is set equal to the value for the corresponding global dof \texttt{JJ}. The correspondence between local and global dofs is obtained from the \texttt{DOMAIN(1)%CONVERS%DOF_LG()} table.

Note that it may happen that none of the dofs belong to the current domain be subjected to a \texttt{DECO BLOQ} constraint, even though \texttt{HAS_BLOQ_DECO} is \texttt{.TRUE.}. In this case, \texttt{DOMAIN(1)%IS_BLOQ} is still allocated but it will contain all zeroes.

Once done the copy, the global data structure can be de-allocated by calling \texttt{DESTROY_BLOQ}.

This will produce an error if any attempt is made to use the global data structure during the transient calculation (correctly, since this is forbidden.) The only item of the global data structure that \textit{must} be preserved is the flag\texttt{HAS_BLOQ}, because this quantity will be used by each sub-domain, as already mentioned. Since the routine \texttt{DESTROY_BLOQ} resets \texttt{HAS_BLOQ} to \texttt{.FALSE.}, we must set it back to \texttt{.TRUE.} after the call to \texttt{DESTROY_BLOQ}, as shown in the last part of the routine.

3.2.7 Subroutine D_CALFREE

The subroutine \texttt{D_CALFREE} computes the free accelerations for each sub-domain. This is where we must add the call to the \texttt{SOLVE_BLOQ} subroutine (in case of domain decomposition.) The added statements are highlighted in red below:

```fortran
SUBROUTINE D_CALFREE ( . . . )
    USE M_LINK_BLOQ_DECO_DATA ! For HAS_BLOQ_DECO
    USE M_LINK_BLOQ ! For SOLVE_BLOQ_DECO
    IF (HAS_BLOQ_DECO) THEN
        decoupled blockages (link deco bloq)
        CALL SOLVE_BLOQ_DECO (DM_FI%VAR, DM_FE%VAR, DM_FC%VAR,
            DM_FI%POS)
    END IF
END SUBROUTINE D_CALFREE
```
Note the variables passed in the exchange list when calling SOLVE_RIGI_DECO from the MPI part of the code (i.e. from D_CALFREE.) The same call from the sequential part (i.e., from CALCUL or TLOOPP) reads:

```
CALL SOLVE_BLOQ_DECO (FI, FE, FDECO)
```

We see therefore that DM_FI%VAR is the vector of nodal internal forces for the current sub-domain and corresponds to the FI global array of the sequential case (without domain decomposition.) The DM_FE%VAR are the external forces for the current sub-domain and correspond to FE in sequential. The DM_FC%VAR are the complementary forces (i.e. the decoupled reaction forces in this case) for the current sub-domain and correspond to FDECO in sequential.

The DM_FI%POS array is the array of pointers into the nodal degrees of freedom for the current sub-domain (POSP in sequential.) This array is the same for all nodal variables such as displacements, velocities, accelerations, forces, masses etc. (but not for nodal coordinates.) Therefore one can choose the pointer of any of these variables, in this case the internal forces, to represent POSP.

It should be noted that for nodes along the interface of the current sub-domain, i.e. for nodes that belong to one or more other neighbouring sub-domains besides the present one, the above arrays of forces contain the total force contributions, i.e. the forces resulting from assembly of elements of the current as well as of neighbour sub-domains.

### 3.2.8 Subroutine SOLVE_BLOQ_DECO

As already mentioned, the subroutine SOLVE_BLOQ_DECO is extracted from the module M_LINK_BLOQ and placed in a stand-alone file. Furthermore, the routine is generalized so that it can be called from either the sequential part (CALCUL or TLOOPP) or the MPI part (D_CALFREE) of the code. The treatment of the multi-domain case is highlighted in red and commented step-by-step below:

```
SUBROUTINE SOLVE_BLOQ_DECO (FI, FE, FDECO, POSP)

USE M_LINK_BLOQ_DECO_DATA
USE M_DOMAINE
USE M_DOMAINE_MPI
USE M_INTERFACE ! For CORRESP

REAL(8), INTENT(IN) :: FI(*)
REAL(8), INTENT(INOUT) :: FE(*), FDECO(*)
INTEGER, INTENT(IN) :: POSP(*) ! Ptr in the dofs, used only with s/d

INTEGER :: K, KMAX, NOD, NODG, NLIBI
INTEGER, POINTER :: IBD(:)
REAL(8) :: WGT

SUBROUTINE SOLVE_BLOQ_DECO (FI, FE, FDECO)

The USE of the new data-only module and of various modules related to sub-domains is added, in order to access the necessary quantities.

```

The USE of the new data-only module and of various modules related to sub-domains is added, in order to access the necessary quantities.

```

Several local variables are added, in particular the INTEGER pointer IBD and the real quantity WGT representing the weight of a node, which is used only in the case with sub-domains, and will be described in the following.
attention: here we use a simpler formula than in solve_single_bloq, tacitly assuming that a blocked node has initial velocity 0!

IF (IDOM_CURRENT == 0) THEN

normal case (without sub-domains)

IBD => IS_BLOQ_DECO
DO K = 1, NLIB ! NLIB is always the global number of dofs
IF (IBD(K) /= 0) THEN
  FDECO(K) = FI(K) - FE(K)
  FE(K) = FE(K) + FDECO(K)
ENDIF
END DO
ELSE

case with sub-domains : in this case IDOM_CURRENT is /= 0.
However, it does not represent the actual index of the current s/d,
but rather the local index of the s/d on the current processor.
In MPI this seems to be typically 1 for all s/ds (at least if there
is just one s/d per processor ...)

IBD => DOMAIN(1)%IS_BLOQ_DECO
NLIBI = DOMAIN(1)%NLIBI ! Local n. of dofs for this s/d

NOD = 1 ! Local node index
KMAX = POSP(NOD+1) - 1 ! Last dof of this local node

DO K = 1, NLIBI
  IF (K > KMAX) THEN ! Increment local node index
    NOD = NOD + 1
    KMAX = POSP(NOD+1) - 1
  ENDIF
  IF (IBD(K) /= 0) THEN
    NODG = DOMAIN(1)%CONVERS%NOEUD_LG(NOD) ! Global node index
    WGT = 1.D0 / DBLE (CORRESP(0,NODG)) ! 1 / Number of sub-domains
    containing NODG
    FDECO(K) = (FI(K) - FE(K))*WGT
    fe will be treated later in the case with sub-domains (MPI)
  ENDIF
END DO
ENDIF

END SUBROUTINE SOLVE_BLOQ_DECO

The local pointer IBD is associated with the global array IS_BLOQ_DECO in the sequential case (IDOM_CURRENT == 0), while it is associated with the local array of the current sub-domain DOMAIN(1)-%IS_BLOQ_DECO in the MPI case.

If domain decomposition is active the nodal weight WGT must be computed for each of the involved nodes. The weight is set to the inverse of the number of sub-domains which actually contain the node under consideration. However, the main loop is performed over the degrees of freedom (from 1 to NLIBI) of the current sub-domain, and not on its nodes. (Note incidentally that NLIBI must be used with sub-domains and not NLIB, which is the global number of degrees of freedom.) Therefore, from the (local) dof it is necessary to compute the corresponding (local) node. This is accomplished by using POSP, which points to the first (local) dof of each node.

Then the node index is converted from local to global (NODG) by means of the DOMAIN(1)%CONVERS-%NOEUD_LG(NOD). Next, the number of sub-domains that include the (global) node (NODG) is returned from the INTEGER function CORRESP(0,NODG) (in M_INTERFACE.) Finally, the nodal weight WGT is set to the inverse of the above number and then the complementary forces (reactions) are computed, and are multiplied by the nodal weight. This compensates the fact that the forces used in the routine are the assembled ones (over all sub-domains) and therefore they include the contribution of all sub-domains for nodes on the interfaces belonging to more than one sub-domain.

The main difference in the MPI case with respect to the sequential case is that the computed reaction force must be multiplied by the (nodal) weight, in order to account for the fact that a node may belong to more than one sub-domain.

Another notable difference between the two cases is that in the sequential case the newly computed reactions (FDECO) are added also to the corresponding external forces (FE), while this operation must
not be done in the MPI case, since in MPI the treatment of external forces is different.

In fact, in MPI the decoupled link forces must be first transferred to remote nodes (if any), for
links based on spatial proximity such as contact or FLSR, and then summed over the interfaces of the
sub-domains to obtain the correct vector to be added to the FE vector. This is why in MPI they must
not be added immediately to FE and included only into FC (FDEC0 here), to which the generic parallel
treatment will then be applied.

Note that, unlike for the porting of the LINK DECO RIGI directive to MPI described in reference [3],
here no ad hoc extra communications between processors (e.g. by calling the FMP RALLSUM_IN_PLACE
routine) are necessary here, since all calculations can be done on the current sub-domain (current
processor) only, simply by taking into account the nodal weights WGT as explained above.

4 Numerical examples

We first consider some numerical examples using the sequential version of EPX (as a reference.) Then,
(conceptually) the same examples are run by the MPI version. All examples are taken from the EPX
non-regression series of benchmark tests.

4.1 Sequential examples

The sequential examples are listed in Table 1.

<table>
<thead>
<tr>
<th>Test name</th>
<th>Mesh</th>
<th>Description</th>
<th>Final time [ms]</th>
<th>Steps</th>
<th>CPU [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLO01</td>
<td>200 Q41L</td>
<td>Sequential</td>
<td>3.0</td>
<td>195</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Table 1: Numerical simulations with the sequential version of EPX.

4.1.1 Case DBLO01

The geometry of the problem is illustrated in Fig. 1(a). A 2D rectangle is discretized in 200 square
elements of type Q41L. The body has an initial velocity directed along the positive x-axis. The right
(vertical) side of the rectangle (c2 object) is blocked along the x-direction by the DECO BLOQ condition:

\[
\text{LINK DECO BLOQ 1 LECT c2 TERM}
\]

The good quality of the solution is confirmed in Fig. 2, where on the left we see that the blocked
x-displacements are indeed 0, while on the right we see that all the x-reactions (here FDEC) at blocked
nodes are non-zero.

![](image1)

Figure 1: Test case DBLO01.

The good quality of the solution is confirmed in Fig. 2, where on the left we see that the blocked
x-displacements are indeed 0, while on the right we see that all the x-reactions (here FDEC) at blocked
nodes are non-zero.
4.2 MPI examples

The MPI examples are listed in Table 2.

<table>
<thead>
<tr>
<th>Test name</th>
<th>Mesh</th>
<th>Description</th>
<th>Final time [ms]</th>
<th>Steps</th>
<th>CPU [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLO02</td>
<td>200 Q4I</td>
<td>MPI (4 procs)</td>
<td>3.0</td>
<td>195</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 2: Numerical simulations with the MPI version of EPX (with 4 processes.)

4.2.1 Case DBLO02

This test is similar to the sequential case DBLO01, but it is run under MPI on 4 processors. The writing of a log file (OPTI LOG) has been disabled because this is believed to penalize MPI calculations (although in this simple case this is of course completely irrelevant.)

The good quality of the solution is confirmed in Fig. 3, where we see that results are identical to those of the sequential case. Note, however, that in this case (MPI) we had to plot the external forces (FEXT) in Fig. 3(b) in order to obtain the reactions, because plotting the complementary forces FDEC like in the sequential case DBLO01 resulted in zero signals.
References


Appendix I — Input files

All the input files used in the previous Sections are listed below.

**dblo01.dgibi**

```plaintext
opti echo 1;
opti dime 2 elem qua4;
opti trac psc ftra 'dblo01_mesh.ps';
opti sauv form 'dblo01.msh';
p1 = 0 0;
p2 = 1 0;
p3 = 1 2;
p4 = 0 2;
pc = 1 1;
c1 = p1 d 10 p2;
c2 = p2 d 10 pc d 10 p3;
c3 = p3 d 10 p4;
c4 = p4 d 20 p1;
mesh = dall c1 c2 c3 c4 plan;
tass mesh noop;
trac qual mesh;
sauv form mesh;
fin;
```

**dblo01.epx**

```plaintext
DBLO01
ECHO
!CONV WIN
LAGR CPLA
CAST mesh
GEMM Q41L mesh TERM
COMP EPAI 1. LECT mesh TERM
! COUL ROSE LECT mesh TERM ! Colors N/A under MPI
MATE VM23 RO 8000. YOUN 1.D11 NU 0.3 ELAS 2.D8
TRAC 3 2.D8 2.D-3 3.D8 1. 3.1D8 2.
LECT mesh TERM
LINK DECO
BLOQ 1 LECT c2 TERM
INIT VITE 1 100.0 LECT mesh DIFF c2 TERM
ECRI DEPL VITE ACCE FIST FEXT FIA FDEC CONT ECRD TERM 1.0-4
FICN ALIC TERM 1.0-4
FICN PVTK TERM 1.0-4
GROUP AUTO
VAR1 DEPL VITE ACCE ECRD
OPTI PAS AUTO NOTE LOG 1
CALC TINI 0. TEND 3.D-3
*=================================================================
SUIT
Post-treatment (time curves from alice file)
ECRO
RESU ALIC GARD PSCR
SORT GRAP
AXTE 1.0 'Time [s]'
COUR 1 'dx_p1' DEPL COMP 1 NOEU LECT p1 TERM
COUR 2 'dx_p2' DEPL COMP 1 NOEU LECT p2 TERM
COUR 3 'dx_p3' DEPL COMP 1 NOEU LECT p3 TERM
COUR 4 'dx_p4' DEPL COMP 1 NOEU LECT p4 TERM
COUR 11 'rx_p2' FDEC COMP 1 NOEU LECT p2 TERM
COUR 12 'rx_p3' FDEC COMP 1 NOEU LECT p3 TERM
COUR 13 'rx_p4' FDEC COMP 1 NOEU LECT p4 TERM
TRAC 1 2 3 4 AXES 1.0 'DEPL. [M]' YZER
TRAC 11 12 13 AXES 1.0 'REAC. [N]' YZER
*=================================================================
QUAL DEPL COMP 1 LECT p1 TERM REFE 8.07906E-2 TOLE 1.E-3
DEPL COMP 1 LECT p2 TERM REFE 0.00000E+0 TOLE 1.E-3
DEPL COMP 1 LECT p3 TERM REFE 0.00000E+0 TOLE 1.E-3
FDEC COMP 1 LECT p2 TERM REFE 4.37838E+5 TOLE 1.E-3
FDEC COMP 1 LECT p3 TERM REFE 4.37838E+5 TOLE 1.E-3
FDEC COMP 1 LECT p4 TERM REFE 4.37838E+5 TOLE 1.E-3
FIN
BEGIN DESCRIPTION
This bench checks the decoupled form of the BLOQ link
(LINK DECO BLOQ) in sequential.
It is the reference solution for the MPI version of the test.
END DESCRIPTION
```

**dblo02.dgibi**

```plaintext
opti echo 1;
opti dime 2 elem qua4;
opti trac psc ftra 'dblo02_mesh.ps';
opti sauv form 'dblo02.msh';
p1 = 0 0;
p2 = 1 0;
p3 = 1 2;
p4 = 0 2;
pc = 1 1;
c1 = p1 d 10 p2;
c2 = p2 d 10 pc d 10 p3;
c3 = p3 d 10 p4;
c4 = p4 d 20 p1;
mesh = dall c1 c2 c3 c4 plan;
tass mesh noop;
trac qual mesh;
sauv form mesh;
fin;
```

**dblo02.epx**

```plaintext
DBLO02
ECHO
BMPI
!CONV WIN
LAGR CPLA
CAST mesh
GEMM Q41L mesh TERM
COMP EPAI 1. LECT mesh TERM
! COUL ROSE LECT mesh TERM ! Colors N/A under MPI
MATE VM23 RO 8000. YOUN 1.D11 NU 0.3 ELAS 2.D8
TRAC 3 2.D8 2.D-3 3.D8 1. 3.1D8 2.
LECT mesh TERM
LINK DECO
BLOQ 1 LECT c2 TERM
INIT VITE 1 100.0 LECT mesh DIFF c2 TERM
ECRI DEPL VITE ACCE FIST FEXT FIA FDEC CONT ECRD TERM 1.0-4
FICN ALIC TERM 1.0-4
FICN PVTK TERM 1.0-4
GROUP AUTO
VAR1 DEPL VITE ACCE ECRD
OPTI PAS AUTO NOTE LOG 1
CALC TINI 0. TEND 3.D-3
*=================================================================
SUIT
Post-treatment (time curves from alice file)
ECRO
RESU ALIC GARD PSCR
SORT GRAP
AXTE 1.0 'Time [s]'
COUR 1 'dx_p1' DEPL COMP 1 NOEU LECT p1 TERM
COUR 2 'dx_p2' DEPL COMP 1 NOEU LECT p2 TERM
COUR 3 'dx_p3' DEPL COMP 1 NOEU LECT p3 TERM
COUR 4 'dx_p4' DEPL COMP 1 NOEU LECT p4 TERM
COUR 11 'rx_p2' FDEC COMP 1 NOEU LECT p2 TERM
COUR 12 'rx_p3' FDEC COMP 1 NOEU LECT p3 TERM
COUR 13 'rx_p4' FDEC COMP 1 NOEU LECT p4 TERM
TRAC 1 2 3 4 AXES 1.0 'DEPL. [M]' YZER
TRAC 11 12 13 AXES 1.0 'REAC. [N]' YZER
*=================================================================
QUAL DEPL COMP 1 LECT p1 TERM REFE 8.07906E-2 TOLE 1.E-3
DEPL COMP 1 LECT p2 TERM REFE 0.00000E+0 TOLE 1.E-3
DEPL COMP 1 LECT p3 TERM REFE 0.00000E+0 TOLE 1.E-3
FDEC COMP 1 LECT p2 TERM REFE 4.37838E+5 TOLE 1.E-3
FDEC COMP 1 LECT p3 TERM REFE 4.37838E+5 TOLE 1.E-3
FDEC COMP 1 LECT p4 TERM REFE 4.37838E+5 TOLE 1.E-3
FIN
BEGIN DESCRIPTION
This bench checks the decoupled form of the BLOQ link
under MPI.
END DESCRIPTION
```

**dblo01.epx**

```plaintext
DBLO01
ECHO
!CONV WIN
LAGR CPLA
CAST mesh
GEMM Q41L mesh TERM
COMP EPAI 1. LECT mesh TERM
! COUL ROSE LECT mesh TERM ! Colors N/A under MPI
MATE VM23 RO 8000. YOUN 1.D11 NU 0.3 ELAS 2.D8
LECT mesh TERM
LINK DECO
BLOQ 1 LECT c2 TERM
INIT VITE 1 100.0 LECT mesh DIFF c2 TERM
ECRI DEPL VITE ACCE FIST FEXT FIA FDEC CONT ECRD TERM 1.0-4
FICN ALIC TERM 1.0-4
FICN PVTK TERM 1.0-4
GROUP AUTO
VAR1 DEPL VITE ACCE ECRD
OPTI PAS AUTO NOTE LOG 1
CALC TINI 0. TEND 3.D-3
*=================================================================
SUIT
Post-treatment (time curves from alice file)
ECRO
RESU ALIC GARD PSCR
SORT GRAP
AXTE 1.0 'Time [s]'
COUR 1 'dx_p1' DEPL COMP 1 NOEU LECT p1 TERM
COUR 2 'dx_p2' DEPL COMP 1 NOEU LECT p2 TERM
COUR 3 'dx_p3' DEPL COMP 1 NOEU LECT p3 TERM
COUR 4 'dx_p4' DEPL COMP 1 NOEU LECT p4 TERM
COUR 11 'rx_p2' FDEC COMP 1 NOEU LECT p2 TERM
COUR 12 'rx_p3' FDEC COMP 1 NOEU LECT p3 TERM
COUR 13 'rx_p4' FDEC COMP 1 NOEU LECT p4 TERM
TRAC 1 2 3 4 AXES 1.0 'DEPL. [M]' YZER
TRAC 11 12 13 AXES 1.0 'REAC. [N]' YZER
*=================================================================
QUAL DEPL COMP 1 LECT p1 TERM REFE 8.07906E-2 TOLE 1.E-3
DEPL COMP 1 LECT p2 TERM REFE 0.00000E+0 TOLE 1.E-3
DEPL COMP 1 LECT p3 TERM REFE 0.00000E+0 TOLE 1.E-3
FDEC COMP 1 LECT p2 TERM REFE 4.37838E+5 TOLE 1.E-3
FDEC COMP 1 LECT p3 TERM REFE 4.37838E+5 TOLE 1.E-3
FDEC COMP 1 LECT p4 TERM REFE 4.37838E+5 TOLE 1.E-3
FIN
BEGIN DESCRIPTION
This bench checks the decoupled form of the BLOQ link
(LINK DECO BLOQ) in sequential.
It is the reference solution for the MPI version of the test.
END DESCRIPTION
```
List of input files

D

dblo01.dgibi ............................................. 12

dblo01.epx ............................................. 12

dblo02.dgibi ............................................. 12

dblo02.epx ............................................. 12
GETTING IN TOUCH WITH THE EU

In person
All over the European Union there are hundreds of Europe Direct information centres. You can find the address of the centre nearest you at: http://europea.eu/contact

On the phone or by email
Europe Direct is a service that answers your questions about the European Union. You can contact this service:
- by freephone: 00 800 6 7 8 9 10 11 (certain operators may charge for these calls),
- at the following standard number: +32 22999696, or
- by electronic mail via: http://europa.eu/contact

FINDING INFORMATION ABOUT THE EU

Online
Information about the European Union in all the official languages of the EU is available on the Europa website at: http://europa.eu

EU publications
You can download or order free and priced EU publications from EU Bookshop at: http://bookshop.europa.eu. Multiple copies of free publications may be obtained by contacting Europe Direct or your local information centre (see http://europa.eu/contact).
JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre’s mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub
ec.europa.eu/jrc

@EU_ScienceHub
EU Science Hub - Joint Research Centre
Joint Research Centre
EU Science Hub