TRUST ICoCo Tutorial V1.7.6

CEA Saclay

Support team: trust@cea.fr

March 19, 2020

1 Introduction to code coupling and ICoCo

2 TRUST initialization

3 Basic test case

4 Coupled problem with ICoCo

1 Introduction to code coupling and ICoCo

2 TRUST initialization

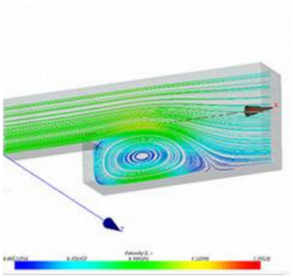
3 Basic test case
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processors

4 Coupled problem with ICoCo
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

Why code coupling?

Code coupling ... what for ?

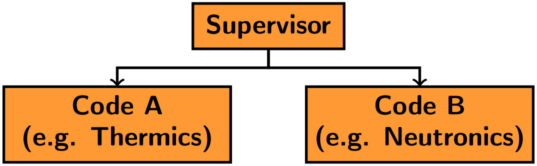
- Traditionnaly *numerical simulation codes* focus on a single physics
  - One code for thermics
  - Another code for mechanics,
  - Etc...
- Real life studies require the simulation of different physics
  - E.g. nuclear reactor simulations require a blend of: thermics, neutronics, mechanics...
- Solution? Code coupling!
  - Have different codes communicating one with another...
  - And having each of them dealing with its own area of expertise



Code coupling

First approach

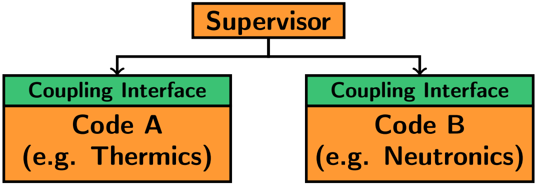
- An entity driving the complete computation is needed: the **supervisor**
  - Initializes code A and code B
  - Loop through code A and code B
  - Centralize the exchanges and conversions between A and B
  - Supervisor is often usually written from scratch as a C++ file, or a Python script
- In a dummy approach : supervisor needs to know *both* A and B API
  - Becomes cumbersome if more than 2 codes to couple ...
  - Or if one supervisor is meant to run with different pairs of codes (code C, D, E ...)



Code coupling

Better: common interface

- **Idea**: define a unique *interface* to which each code must comply
  - Only an *interface*, i.e. a contract (=an API) that each code fulfills
  - *Implementation* (=plugging the wires) of this interface is tightly linked to the details of the code...
- Advantages:
  - All codes implementing the (unique) interface can be passed to the supervisor (almost) without modifying it! Versatility
- Cons:
  - Need to implement the interface for each code to be coupled



ICoCo

Interface for COde COupling

- ICoCo is such a coupling interface
- Stands for Interface for Code Coupling
- Written in C++
- Initially designed for simulation codes exhibiting an **iterative time loop**
- Presents a set of standard methods whose signature is fixed, with no implementation by default:
  - initializeTimeStep()
  - validateTimeStep()
  - abortTimeStep()
  - ...
- Uses the notion of field for the exchange of data between codes
  - A field is a set of values supported by a mesh
  - A possible implementation: **MEDCouplingFieldDouble**, from SALOME's MEDCoupling library

Further reading

Meaningful documentation

- ICOCO presentation in TRUST documentation:
  - "An Interface for Code Coupling ICoCo v1.2":

    ```
    $ source /home/triou/env_TRUST_X.Y.Z.sh
    $ evince $TRUST_ROOT/doc/Kernel/ICoCo_V1.2.pdf &
    ```
  - Ask the "APIProblem.pdf" note to trust@cea.fr

1 Introduction to code coupling and ICoCo

2 TRUST initialization

3 Basic test case
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processors

4 Coupled problem with ICoCo
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

Initialisation of TRUST & ICoCo environment

- Source the TRUST environment:
```
$ source /home/triou/env_TRUST_X.Y.Z.sh
```
- To know if the configuration is ok and where are the sources:
```
$ echo $TRUST_ROOT
```
- Source the ICoCo environment:
```
$ source $TRUST_ROOT/Outils/ICoCo/ICoCo_src/env_MEDICoCo.sh
```
- Test if ICoCo is compiled:
```
$ ls $exec
```
- If you obtain "ls: cannot access **/Outils/ICoCo/ICoCo_src/ICoCo_opt: No such file or directory", then compile ICoCo_src:
```
$ cd $TRUST_ROOT/Outils/ICoCo/ICoCo_src
$ baltik_build_configure -execute
$ make optim debug
$ source full_env_MEDICoCo.sh
```

1 Introduction to code coupling and ICoCo

2 TRUST initialization

3 Basic test case
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processors

4 Coupled problem with ICoCo
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

First with TRUST:

Launch calculation with TRUST

- We create a first reference test case, launched with TRUST.
- Then we want to launch it with ICoCo and compare the results.
- Copy the following test case in your repository:

```
$ mkdir ICoCo_exercises
$ cd ICoCo_exercises
$ trust -copy Vahl_Davis_hexa
```

- Rename the folder to sort your tests:

```
$ mv Vahl_Davis_hexa Vahl_Davis_hexa_trust
$ cd Vahl_Davis_hexa_trust
```

- Rename the data file to be consistent with the name of the repository:

```
$ mv Vahl_Davis_hexa.data Vahl_Davis_hexa_trust.data
```

- Launch calculation:

```
$ trust Vahl_Davis_hexa_trust
```

- You obtained a Vahl_Davis_hexa_trust.lml file. We will use it in the following part.

With ICoCo

Adjusting the data file

- Now lets do the same with ICoCo:
  ```
  $ cd ..
  ```
- You are now in your folder "ICoCo_exercises", create a new directory in order to launch your test case with ICoCo:
  ```
  $ trust -copy Vahl_Davis_hexa
  $ mv Vahl_Davis_hexa Vahl_Davis_hexa_ICoCo
  ```
- Rename the data file to be consistent with the name of the repository:
  ```
  $ cd Vahl_Davis_hexa_ICoCo
  $ mv Vahl_Davis_hexa.data Vahl_Davis_hexa_ICoCo.data
  ```
- Edit the data file and modify it to add ICoCo instructions:
  - Add the following line after the definition of the dimension: *Nom ICoCoProblemName Lire ICoCoProblemName pb*
  - Comment the "solve pb" line at the end of the data file.

With ICoCo

Creation of the main.cpp file

- Create the main.cpp which will launch the calculation:
  ```
  $ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/main1.cpp main.cpp
  ```
- You can edit it and see the main method which creates the objects needed to do the information exchanges.
- You can use ICoCo with 1 or more processors.
- Here we use only one processor to solve the problem.

Creation of the makefile

- Create a makefile for your calculation on 1 proc:
  ```
  $ sh $project_directory/share/bin/create_Makefile 1
  ```
- Compile it:
  ```
  $ make
  ```
- It creates an executable "couplage" and a data file "couplage.data".

With ICoCo

Launch calculation

- Execute it:
  `$./couplage`
- You may obtain the same results as with TRUST executable.
- Compare it:
  ```
  $ compare_lata Vahl_Davis_hexa_ICoCo.lml
  ../Vahl_Davis_hexa_trust/Vahl_Davis_hexa_trust.lml
  ```
- The files are the sames!

1 Introduction to code coupling and ICoCo

2 TRUST initialization

3 Basic test case
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processors

4 Coupled problem with ICoCo
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

First input with ICoCo

Adjusting the main.cpp file

- Copy the following test case in your repository:

```
$ cd ICoCo_exercises
$ cp -r Vahl_Davis_hexa_ICoCo Vahl_Davis_hexa_ICoCo_exchange
$ cd Vahl_Davis_hexa_ICoCo_exchange
```

- Clean the repository and rename the data file:

```
$ trust -clean
$ mv Vahl_Davis_hexa_ICoCo.data
Vahl_Davis_hexa_ICoCo_exchange.data
```

- We have to add a block in the main.cpp file to made exchanges.
- Copy the following file in your folder:

```
$ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/main2.cpp main.cpp
```

First input with ICoCo

Adjusting the data file

- Then modify the data file Vahl_Davis_hexa_ICoCo_exchange.data to made the input:
  - Change the line:
    "Gauche Paroi_temperature_imposee Champ_Front_Uniforme 1 10."
    to
    "Gauche Paroi_temperature_imposee ch_front_input { nb_comp 1 nom TEMPERATURE_IN_DOM probleme pb }"
  - You can see that the name "TEMPERATURE_IN_DOM" is the one employed in the main.cpp file.

First input with ICoCo

Launch calculation

- Now we can compile and launch the calculation:
  ```
  $ make
  $ ./couplage
  ```
- You may obtain the same results as with TRUST executable.
- Compare it:
  ```
  $ compare_lata Vahl_Davis_hexa_ICoCo_exchange.lml
  ../Vahl_Davis_hexa_trust/Vahl_Davis_hexa_trust.lml
  ```
- The files are the sames! (but not for the first time!!!!!)

1 Introduction to code coupling and ICoCo

2 TRUST initialization

3 Basic test case
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processors

4 Coupled problem with ICoCo
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

First input with ICoCo

Adjusting the main.cpp file

- Copy the following test case in your repository:

```
$ cd ICoCo_exercises
$ cp -r Vahl_Davis_hexa_ICoCo_exchange
Vahl_Davis_hexa_ICoCo_para
$ cd Vahl_Davis_hexa_ICoCo_para
```

- Clean the repository and rename the data file:

```
$ trust -clean
$ rm main.cpp
$ mv Vahl_Davis_hexa_ICoCo_exchange.data
Vahl_Davis_hexa_ICoCo_para.data
$ trust -partition Vahl_Davis_hexa_ICoCo_para
```

- Copy the following file in your folder:

```
$ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/main3.cpp main.cpp
```

First input with ICoCo

Adjusting the main.cpp file

- Open the main.cpp file and search for the MPI command lines.
- Open this file and look where:
  - the processors are added: search for "dom_ids"
  - the names of the data files: search for "data_file"
- Compile your new file:
  ```
  $ make
  ```
- To run parallel, you have to use the following mpirun command:
  ```
  $ mpirun -np 2 ./couplage
  ```
- Compare your results with the sequential ones:
  ```
  $ compare_lata PAR_Vahl_Davis_hexa_ICoCo_para.lml
  ../Vahl_Davis_hexa_trust/Vahl_Davis_hexa_trust.lml
  ```

1 Introduction to code coupling and ICoCo

2 TRUST initialization

3 Basic test case
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processors

4 Coupled problem with ICoCo
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

First with TRUST

Launch calculation with TRUST

- Copy a TRUST test case:

```
$ cd ICoCo_exercises
$ trust -copy docond_VEF_3D
$ mv docond_VEF_3D docond_VEF_3D_trust
```

- Launch calculation:

```
$ cd docond_VEF_3D_trust
$ trust docond_VEF_3D
```

- Let do it in parallel also:

```
$ trust -partition docond_VEF_3D
$ trust PAR_docond_VEF_3D 2
```

- Compare the two results:

```
$ compare_lata docond_VEF_3D.lml PAR_docond_VEF_3D.lml
```

- The results are the same. Differences are below the threshold: 10^{-5}!

Separate a coupled problem into two new problems

Separate the meshes

- Create your ICoCo test case:

```
$ cd ICoCo_exercises
$ trust -copy docond_VEF_3D
$ mv docond_VEF_3D docond_VEF_3D_ICoCo
$ cd docond_VEF_3D_ICoCo
```

- Separate mesh and calculation datas in two data files:

```
$ cp docond_VEF_3D.data docond_VEF_3D_mesh1.data
```

- Edit the file docond_VEF_3D_mesh1.data and remove:
  - the time scheme
  - the problem definitions
  - the 'scatter' block
  - the discretization, medium, gravity definition blocks
  - the associations
  - the definition and "Read pb1"/"Read pb2" block
  - the "fichier pb1"/"fichier pb2" lines
  - the "solve" keyword

Separate a coupled problem into two new problems

Separate the meshes

- Keep only the domain definitions, meshes and cutting steps.
- Then create a data file for each domain:
  ```
  $ cp docond_VEF_3D_mesh1.data docond_VEF_3D_mesh2.data
  ```
- In the file docond_VEF_3D_mesh1.data, keep only the informations of the solide domain.
- In the file docond_VEF_3D_mesh2.data, keep only the informations of the fluide domain.
- Uncomment the 'partition' step for each domain.
- Run these data files:
  ```
  $ trust docond_VEF_3D_mesh1
  $ trust docond_VEF_3D_mesh2
  ```
- You must have now the four files:
  DOM1_0000.Zones DOM2_0001.Zones
  DOM1_0001.Zones DOM2_0000.Zones

Separate a coupled problem into two new problems

Run with separated meshes

- In docond_VEF_3D.data file:
  - remove the mesh and cutting command (which are already in the mesh data file)
  - keep the 'scatter' command and uncomment it
- Launch the calculation:

```
$ trust docond_VEF_3D 2
```

- Compare results with previous parallel results:

```
$ compare_lata docond_VEF_3D.lml
../docond_VEF_3D_trust/PAR_docond_VEF_3D.lml
```

- The results are the sames.
- Separate results in two lml files:
  - add "fichier pb1.lml" in the "Post_processing" block of the solid domain,
  - add "fichier pb2.lml" in the "Post_processing" block of the fluid domain,
- Run calculation to create this two files:

```
$ trust docond_VEF_3D 2
```

Separate a coupled problem into two new problems

Separate data files

- Create a data file for the solid domain:
  ```
  $ cp docond_VEF_3D.data docond_VEF_3D_dom1.data
  ```
- In docond_VEF_3D_dom1.datafile, remove the lines:
  - Probleme_Couple pbc
  - Associate pbc pb1
  - Associate pbc pb2
- Change the following lines:
  - Associate pbc sch $\rightarrow$ Associate pb sch
  - Discretize pbc dis $\rightarrow$ Discretize pb dis
  - Solve pbc $\rightarrow$ Solve pb
- Create a data file for the fluide domain:
  ```
  $ cp docond_VEF_3D_dom1.data docond_VEF_3D_dom2.data
  ```
- In docond_VEF_3D_dom1.data, keep only the informations about the solid domain (pb1, dom_solide).
- Subtitute pb1 $\rightarrow$ pb

Separate a coupled problem into two new problems

Separate data files

- In docond_VEF_3D_dom2.data, keep only the informations about the fluide domain (pb2, dom_fluide).
- Subtitute pb2 → pb
- You can see that the heat exchange is made on the "Paroi_echange1" boundary for solid domain and "Paroi_echange2" boundary for the fluid domain.
- Modify docond_VEF_3D_dom1.data file to have:
  Paroi_echange1 paroi_contact pb Paroi_echange2
  →
  Paroi_echange1 paroi_temperature_imposee Champ_Front_Uniforme 1 50.
- Modify docond_VEF_3D_dom2.data file to have:
  Paroi_echange2 paroi_contact pb Paroi_echange1
  →
  Paroi_echange2 paroi_temperature_imposee Champ_Front_Uniforme 1 50.

Separate a coupled problem into two new problems

Separate data files

- Run docond_VEF_3D_dom1.data and docond_VEF_3D_dom2.data in parallel:
  ```
  $ trust docond_VEF_3D_dom1 2
  $ trust docond_VEF_3D_dom2 2
  ```
- The two problems must run.
- Notice that there is no coupling at all for the moment.

Adjusting data files

- To create the ICoCo problem, just after 'dimension 3', add in docond_VEF_3D_dom1.data and docond_VEF_3D_dom2.data: 'Nom ICoCoProblemName Lire ICoCoProblemName pb'
- Remove 'Solve pb' because the solving step will be made by ICoCo.

Run with ICoCo

Creation of the main.cpp file

- We have to create a new executable which will use our data files.
- Copy the following main.cpp file in your repository:

```
$ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/main3.cpp main.cpp
```

- Open this file and look where:
  - the processors are added: search for "dom1_ids"
  - the names of the data files: search for "data_file"
  - the loop to iterate on time steps: search for "while"

Run with ICoCo

Compiling and launching

- Create a makefile to compile your main.cpp file:
  ```
  $ sh $project_directory/share/bin/create_Makefile 4
  ```
- Compile the main.cpp file:
  ```
  $ make
  ```
- Launch calculation:
  ```
  $ mpirun -np 4 ./couplage
  ```
- Compare the results to the results of the coupled problem:
  ```
  $ compare_lata pb1.lml docond_VEF_3D_dom1.lml
  $ compare_lata pb2.lml docond_VEF_3D_dom2.lml
  ```
- As expected, there are differences between the results because there is no coupling here, we impose the temperature in the data files.

1 Introduction to code coupling and ICoCo

2 TRUST initialization

3 Basic test case
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processors

4 Coupled problem with ICoCo
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

Run with ICoCo

Adjusting data file

- Now we want to send the temperature from the Paroi_echange2 boundary of the domain dom2 to the Paroi_echange1 boundary of the domain dom1.
- Create a new directory:

```
$ cd ICoCo_exercises
$ cp -r docond_VEF_3D_ICoCo docond_VEF_3D_ICoCo_coupling1
$ cd docond_VEF_3D_ICoCo_coupling1
```

- In the docond_VEF_3D_dom2.data data file, add in the "Definition_champs" block of the post-processings:

```
   TEMPERATURE_OUT_DOM2 Interpolation {
     localisation elem
     domaine dom_fluide_boundaries_Paroi_echange2
     source refChamp { Pb_Champ pb temperature }
   }
```

where "dom_fluide_boundaries_Paroi_echange2" is a predefined name for the boundary Paroi_echange2 of the domain dom2.

Run with ICoCo

Adjusting data file

- In the docond_VEF_3D_dom1.data data file, change the boundary condition on the 'Paroi_echange1' boundary to:
  Paroi_echange1 paroi_temperature_imposee ch_front_input { nb_comp 1 nom TEMPERATURE_IN_DOM1 probleme pb }
- Copy the main.cpp file for this exchange:
  ```
  $ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/main4.cpp main.cpp
  ```
- Compare the new main.cpp file to the previous one:
  ```
  $ tkdiff main.cpp ../docond_VEF_3D_ICoCo/main.cpp
  ```
- You can see where the new fields TEMPERATURE_IN_DOM1 and TEMPERATURE_OUT_DOM2 added to the data files, are used in a new part which makes exchanges.
- Notice that we use two new objects: one TrioDEC object and one TrioField object.
- Some comments are written to help you.

Run with ICoCo

Adjusting data file

- Compile the main.cpp file:
```
$ make
```
- Launch calculation:
```
$ mpirun -np 4 ./couplage
```
- Compare results with those without coupling:
```
$ compare_lata docond_VEF_3D_dom1.lml
../docond_VEF_3D_ICoCo/docond_VEF_3D_dom1.lml
$ compare_lata docond_VEF_3D_dom2.lml
../docond_VEF_3D_ICoCo/docond_VEF_3D_dom2.lml
```
- The results on the domain dom2 are the same as this calculation made only one more post-traitment.
- But we can see that the coupling works well because the results on the domain dom1 changes.

1 Introduction to code coupling and ICoCo

2 TRUST initialization

3 Basic test case
- ICoCo: without exchange
- ICoCo: first input
- ICoCo: on 2 processors

4 Coupled problem with ICoCo
- ICoCo: without exchange
- ICoCo: one way coupling
- ICoCo: two way coupling

Run with ICoCo

Adjusting data file

- Two way coupling for thermal problems should use Dirichlet and Neumann boundary conditions (using only Dirichlet boundary conditions for coupling both sides would not work).
- So we want to send the heat flux from the Paroi_echange1 boundary of the domain dom1 to the Paroi_echange2 boundary of the domain dom2.
- Create a new directory:

  ```
  $ cd ICoCo_exercises
  $ cp -r docond_VEF_3D_ICoCo_coupling1
  docond_VEF_3D_ICoCo_coupling2
  $ cd docond_VEF_3D_ICoCo_coupling2
  ```

- Inspire you of the previous part to make a Neumann boundary condition (heat flux imposed):

Run with ICoCo

Adjusting data file

- In the docond_VEF_3D_dom1.data data file, add in the "Definition_champs" block of the post-processings:

```
FLUX_SURFACIQUE_OUT_DOM1 Interpolation {
  localisation elem
  domaine dom_fluide_boundaries_Paroi_echange1
  source Morceau_equation {
   type operateur numero 0 option flux_surfacique_bords
   source refChamp { Pb_Champ pb temperature }
  }
}
```

  where "dom_solide_boundaries_Paroi_echange1" is a predefined name for the boundary Paroi_echange1 of the domain dom1.

- In the docond_VEF_3D_dom2.data data file, change the boundary condition on the 'Paroi_echange2' boundary to:
  Paroi_echange2 paroi_flux_impose ch_front_input { nb_comp 1 nom FLUX_SURFACIQUE_IN_DOM2 probleme pb }

Run with ICoCo

Adjusting data file

- Modify the main.cpp file to add a new exchange:
  - Create a new TrioDEC object to made exchange from domain dom2 to domain dom1:
    TrioDEC dec_flux_surfacique2(dom2_ids, dom1_ids);
  - Create a new TrioField object:
    TrioField field_flux_surfacique2;
  - Add code lines into the while loop to made this exchange of informations.
- You can have a look at the main5.cpp file for this exchange:
  ```
  $ cp $TRUST_ROOT/doc/TRUST/exercices/ICoCo/main5.cpp
  main5.cpp
  ```
- Compare it to the previous one:
  ```
  $ tkdiff main5.cpp ../docond_VEF_3D_ICoCo_coupling1/main.cpp
  ```
- You can see the use of the new fields FLUX_SURFACIQUE_IN_DOM2 and FLUX_SURFACIQUE_OUT_DOM1 added to the data files.

Run with ICoCo

Adjusting data file

- Compile your main.cpp file:
```
$ make
```
- Launch calculation:
```
$ mpirun -np 4 ./couplage
```
- Compare results with the first ones:
```
$ compare_lata docond_VEF_3D_dom1.lml
../docond_VEF_3D_ICoCo/pb1.lml
$ compare_lata docond_VEF_3D_dom2.lml
../docond_VEF_3D_ICoCo/docond_VEF_3D_dom2.lml
```