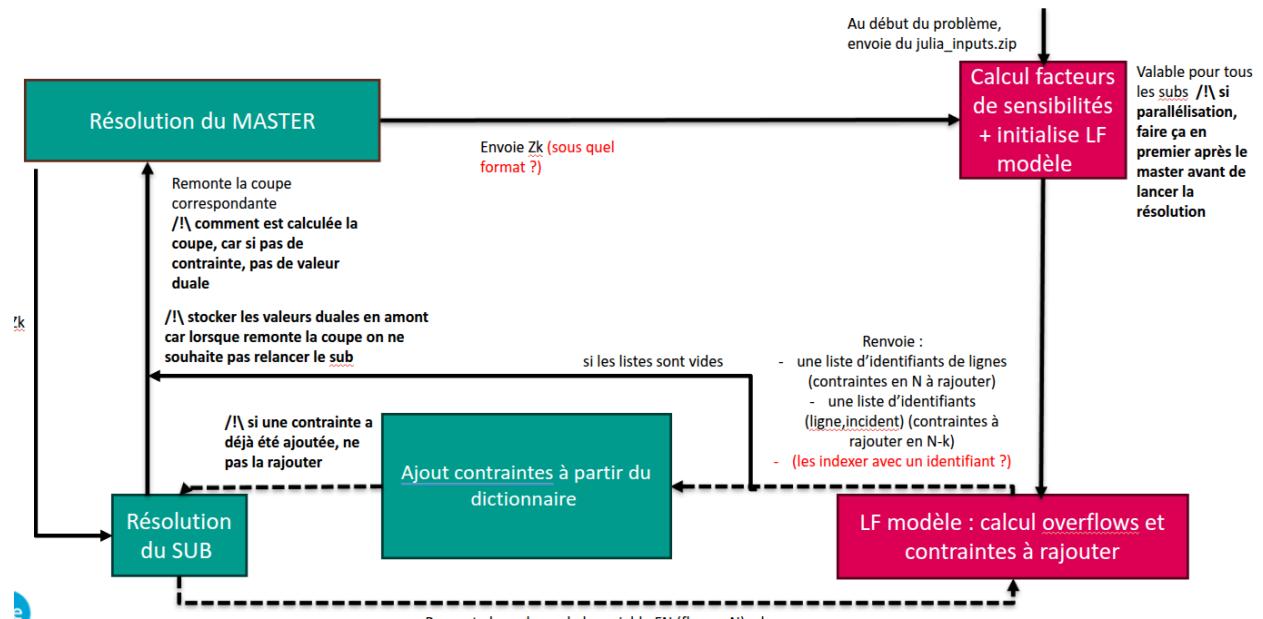


Julia code compilation

Context



In order to integrate the micro iteration workflow in benders code, we need to do some calls to some Julia code.

As the code is not robust enough and can often change (depending on the grid modelisation process), it is necessary to keep the julia code in that language and find a way to integrate it in the c++ in the form of a dynamic library.

Julia C ABI :

So Julia provide the `PackageCompiler` a julia library that allow compiling julia code into a dynamic library (.so on Ubuntu).

In the project , [compile.il](#) handles the process of compiling the [MyLibrary.il](#) code as it's shown here :

```
create_library(  
    ".",  
    "libmylib";  
    lib_name="mylib",  
    precompile_execution_file="src/MyLib.jl",  
    include_lazy_artifacts=false  
)
```

So This code will compile the srd/[MyLib.jl](#) into a collection of so files that we will find in libmylib repo.

Julia code to compile

[MyLib.jl](#) contains the whole code that will build the library (it's all based on micro_benders_iterations_PTDF.jl and micro_iterations_benders_LOADFLOW.jl).

So the functions that will be called in the c++ side should have a particular signature : they should start with [Base.@ccallable](#) and only takes as inputs and outputs c compatible types (sturctures, pointers , Cstring, Cint etc).

For our workflow we identify 3 main methods :

```
Base.@ccallable function jl_load_variables(subproblems_ids::SubProblemsIds)::Cvoid
```

```
struct SubProblemsIds  
    subProblems_ids::Ptr{Cstring}  
    n_subproblems::Cint  
end
```

This method will be called before the benders begins. It will allow loading all the global variables (from .jls and .csv) that we will need at each benders operation

It takes as input a structure that contains a list of sub problems identifiers

```
Base.@ccallable function jl_compute_factors_for_microiterations(candidates::MasterBendersInput)::Cvoid
```

```
struct CandidateLineMasterIterationResult
    candidate_line_id::Cstring
    is_invested::Cint
end

struct MasterBendersInput
    candidates_res::Ptr{CandidateLineMasterIterationResult}
    size::Cint
end
```

This function will be called after solving the master problem at each iteration. It takes as input a list to the candidate line (invested or not)

```
Base.@ccallable function jl_return_constraints_for_micro_iteration(subproblem_id::Ptr{UInt8}, flow_list::FlowNList)::Cvoid
```

```
struct FlowNList
    flows::Ptr{FlowN}
    size::Cint
end
```

This method will be called for each subproblem. It takes as input the subproblem identifier and a list of surveyed flows

How to build the compiled library

In the project root :

- 1- Get into the root of the project
- 2- Launch Julia CLI
- 3- “using Pkg”
- 4- “Pkg.activate(“.”)”
- 5- “Pkg.resolve()”
- 6- “Pkg.add([“PackageCompiler”, “Serialization”, “CSV”, “NamedArrays”, “SparseArrays”, “DataFrames”, “LinearAlgebra”])”
- 7- “Pkg.instantiate()”
- 8- exit()

9- julia [compile.jl](#)