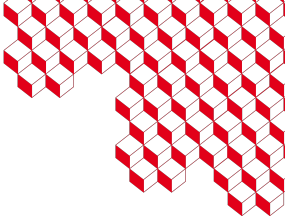


Distributed computing with Uranie

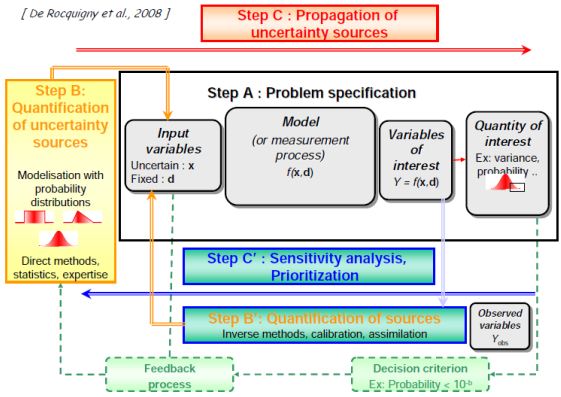
A broad introduction

HPC and Uncertainty Treatment with Open TURNS and Uranie, 29/09/2023

Jean-Baptiste Blanchard (jean-baptiste.blanchard@cea.fr)
CEA DES/ISAS/DM2S/SGLS/LIAD SACLAY

Outline

Reminder: uncertainty quantification



Main steps:

- A: problem definition
  - Uncertain input variables
  - Variable/quantity of interest
  - Model construction
- B: uncertainty quantification
  - Choice of pdfs
  - Choice of correlations
- B': quantification of sources
  - Inverse methods using data to constrain input values and uncertainties
- C: uncertainty propagation
  - Evolution of output variability w.r.t input uncertainty
- C': sensitivity analysis
  - Uncertainty source sorting

These steps are usually model dependent, it might be useful to iterate to help converging to proper conclusions

What should be defined

Main steps

- define the simulation code, along with inputs and outputs;
- identify ways to communicate inputs and outputs to the model;
- check that you remove any unsafe options;
- check locally (even by tempering with the code) that the mechanisms are set properly;

Resources

- check with your local authority the configuration of the cluster and how to submit job;
- check with your local authority the amount of resources you can get your hands on;
- define properly the number of computation to be done and their own level of parallelisation;
- press the red button !

Outline

From where one starts

Simple example

The beam use case, a “xml” external code

- A *"external code"* simulates the **"beam"** problem (if **beam** $\notin$ `$PATH` **; source ...**)

```
beam -h
** Usage: ./beam [-v] [-h|-?] [-x [file]]
      -v : Pass to the verbose mode
   -h|-? : Print the usage message
      -x : Set the input file with XML format [beam_input_with_xml.in]
```

- The parameters of the study, with their deterministic values, are:
  - **E** $(3.0e^{7})$: the Young modulus
  - **F** (300) : the ponctual load
  - **L** (2.5) : the length of the beam
  - **I** $(4.0e^{-6})$: the flexion inerty
- The values of these parameters are stored in the *"XML"* input file ***"beam.xml"***:

```
<?xml version="1.0"?>
<beam>
<description name="beam" title="UseCase beam with XML input file" version="1.0" date="2014-04-07">
    <tool name="beam exe" version="1.0"/>
</description>
<inputs E="3.0e7" F= "300.0" L="2.5" I="4.0e-6"/>
...
</beam>
```

The beam use case, single evaluation

- Evaluate the external code on the *"XML"* input file ***"beam.xml"*** :

```
beam -x beam.xml
```

- The same in *"verbose"* mode (**-v option**)

```
beam -v -x beam.xml
*************************
** The Beam use case
** Nb Argument [4]
...
** verbose mode
** XML Input File[beam.xml]
*************************
...
***************************************
** writeXMLOutputs in the XML file [ _beam_outputs_.xml] ...
** End Of writeXMLOutputs with XML file
***************************************
...
*************************
** beam::printLog
** _sFile[beam.xml]
** Inputs : E[3e+07] F[300] L[2.5] I[4e-06] ************
** Output :
** Deviation[13.0208]
...
```

The beam use case, reading outputs

- The target variable is stored in the *"XML"* output file `_beam_outputs_.xml`

```
<?xml version="1.0"?>
<beam>
<description name="beam" title="UseCase beam with XML input file" version="1.0" date="2014-04-
07">
    <tool name="beam exe" version="1.0"/>
</description>
<inputs F="300.0" E="3.0e7" L="2.5" I="4.0e-6"/>
<computation>
    <derivate activate="on"/>
    <hessian activate="off"/>
</computation>
<outputs deviation="1.3020e+01">
  <derivates partialE="-4.340e-07" partialF="4.340e-02" partialL="1.562e+01" partialI="-
3.255e+06"/>
  <hessian><partialE partialE="2.893518519e-14" partialF="-1.446759259e-09" partialL="-
5.208333333e-07" partialI="1.085069444e-01"/><partialF partialF="0.000000000e+00" partialL="5.208333333e-
02" partialI="-1.085069444e+04"/><partialL partialL="1.250000000e+01" partialI="-3.906250000e+06"/>
<partialI partialI="1.627604167e+12"/></hessian></outputs></beam>
```

The beam use case, interface with Uranie

- XPath for the inputs attributes in the *"XML"* file `"beam.xml"`

```
<?xml version="1.0"?>
<beam>
<description name="beam" title="UseCase beam with XML input file" version="1.0" date="2014-04-07">
    <tool name="beam exe" version="1.0"/>
</description>
<inputs E="3.0e7" F= "300.0" L="2.5" I="4.0e-6"/>
```

  - type : Attribute, ~~Field~~
  - E :: `//inputs/@E`
  - F :: `//inputs/@F`
  - L :: `//inputs/@L`
  - I :: `//inputs/@I`

- XPath for the output attribute *deviation* in the *"XML"* file `"_beam_outputs_.xml"`

```
<outputs deviation="1.3020e+01">
  <derivates partialE="-4.340e-07" partialF="4.340e-02" partialL="1.562e+01" partialI="-3.255e+06"/>
```

  - type : Attribute, ~~Field~~
  - deviation :: `//outputs/@deviation`

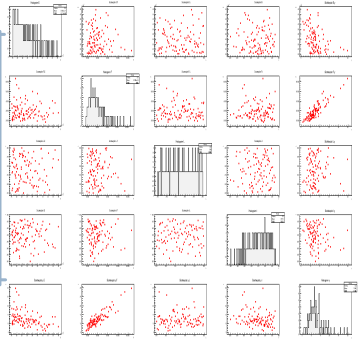
The beam use case, local and sequential

- Evaluate a LHS DoE of $nS = 100$ patterns with the Uranie macro *"macroLauncher.C"*

```
root -l macroLauncher.C
```

- which gives

```
Processing macroLauncher.C...
...
<URANIE::INFO> ** Proc : 1
<URANIE::INFO> (100%, time left: 0 sec : Failure 0)
<URANIE::INFO> time elapsed 14.8 sec
<URANIE::INFO> ** Clean the Working Directory ... Done
...
** npatterns[1000]
** y Mean[12.623]
** y Sigma[4.29351]
** L Mean[255]
** L Sigma[2.8881]
```



The beam use case, what's my cluster ?

- Cluster : **Ruche** (92 nodes CPU with 2 proc./nodes **and** 10 nodes GPU (NVIDIA Tesla V100))
- Use the **SLurm** as Job Submission
  - List the tools in your environment

```
module list
```

  - List the tools available

```
module avail
```

  - Load a tool in your environment

```
module load tool/version/compiler
```

  - Unload a tool from your environment

```
module unload tool/version/compiler
module purge
```

The beam use case, slurm commands

- Useful commands of **SLurm**
  - Submit the job file *"job.sh"* (describe in a next slide)

```
sbatch job.sh
```

  - List SLurm jobs in the cluster

```
squeue <-u user>
```

  - Delete a SLurm job

```
scancel JobID
```

- Help on the Mesocentre (Ruche and Fusion)

The beam use case, submission script

- Job submission file *"job_slurm.sh"*

```
#!/bin/bash
#SBATCH -job-name    = BeamLauncher                # Request name
#SBATCH -output      = run_$(jobname).o$(jobid)    # Standard output with the job "Name" & "Id"
#SBATCH -error       = run_$(jobname).e$(jobid)    # Error output with the job "Name" & "Id"
#SBATCH -nodes       = 50                          # Number of nodes to use
#SBATCH -ntasks      = 50                          # Number of tasks to use
#SBATCH -time        = 00:15:00                    # Elapsed time limit in <hh:mm:ss>
#SBATCH -partition   = cpu_short                   # Specify the Slurm partition for the job

# Load the modules if necessary
module purge
module load ...
module load root/6.24.0/gcc-9.2.0
module load uranie/4.5.0/gcc-9.2.0
module load mpich/3.3.2/gcc-9.2.0

# Source the environment variables if necessary
# source /gpfs/users/.../uranie.bashrc

# Remove all old files
rm -f _launcher_code_.* run_*.o run_*.e

# Execute the Uranie macro with the -q option to quit ROOT
root -l -q macroLauncher.C
# End Of File
```

The beam use case, running it

- Command to submit the job file *"job_slurm.sh"*

```
sbatch job_slurm.sh
```

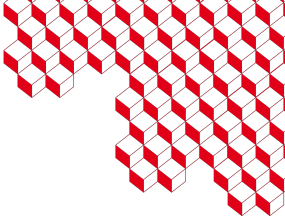
Conclusion and beyond

From what we've seen

- It is the same macro to launch jobs either in sequential or in parallel mode
- It is operational Job Submission systems such as SGE, LSF, LoadLeveler (IBM) and SLurm (Irene, Joliot-Curie/TGCC)
- One should not forget to run the macro in batch mode (-q option in the ROOT command for C++, to quit ROOT when the computation is finished and free the core, or -b in python approach)

Not discussed here

- Depending on your local cluster policy, there are different ways to handle jobs submissions:
  → possible to distribute computation through SSH without any Uranie installation on local clusters.
- One might want to submit jobs for which every single computations is distributed as well (nested distribution).



Thanks! Any questions?