Guix

How to use Org mode and Guix to build a reproducible experimental study

Guix HPC Workshop 2023

November 9, 2023

Marek Felšöci
marek.felsoci@inria.fr

$ whoami

Education

- Bachelor's and Master's degrees in computer science @ Unistra
- Ph.D. in computer science @ Université de Bordeaux

Master's intership (ICube Strasbourg)

- implementation of a new programming structure for the C language

Thesis (Inria Bordeaux) ← The Epoch (for me)

- solvers for large coupled sparse/dense linear systems arising from aeroacoustic simulations

Post-doc (Inria Nancy / ICube Strasbourg, current position)

- automatic parallelization of C/C++ programs

Scientific context

AIRBUS Région Nouvelle-Aquitaine

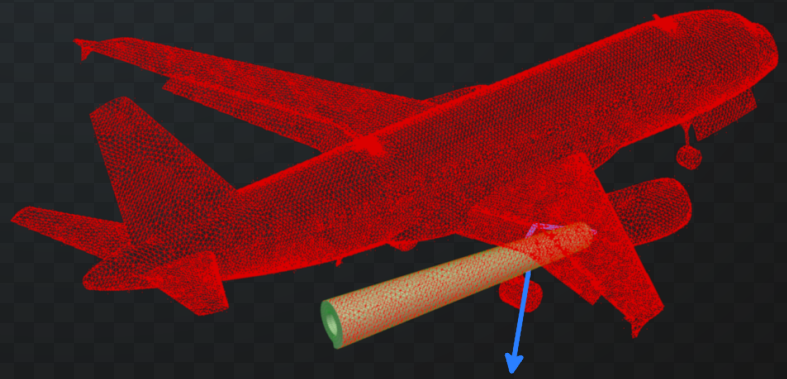
- simulating propagation of soundwaves around an aircraft in flight
- solving large coupled sparse/dense linear systems

Real situation [1].

Scientific context

AIRBUS Région Nouvelle-Aquitaine

- simulating propagation of soundwaves around an aircraft in flight
- solving large coupled sparse/dense linear systems



Discrete numerical 3D/2D model.

Scientific context

AIRBUS Région Nouvelle-Aquitaine

- simulating propagation of soundwaves around an aircraft in flight
- solving large coupled sparse/dense linear systems

$$\begin{bmatrix} A_{vv_{11}} & \cdots & A_{vv_{1n}} & A^T_{sv_{11}} & \cdots & A^T_{sv_{1k}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A_{vv_{n1}} & \cdots & A_{vv_{nn}} & A^T_{sv_{m1}} & \cdots & A^T_{sv_{mk}} \\ A_{sv_{11}} & \cdots & A_{sv_{1n}} & A_{ss_{11}} & \cdots & A_{ss_{1k}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A_{sv_{k1}} & \cdots & A_{sv_{kn}} & A_{ss_{k1}} & \cdots & A_{ss_{kk}} \end{bmatrix} \times \begin{bmatrix} x_{v_1} \\ \vdots \\ x_{v_n} \\ x_{s_1} \\ \vdots \\ x_{s_k} \end{bmatrix} = \begin{bmatrix} b_{v_1} \\ \vdots \\ b_{v_n} \\ b_{s_1} \\ \vdots \\ b_{s_k} \end{bmatrix}$$

Numerical context

Methodology

10 propose and implement an algorithm within an existing codebase

20 evaluate with a series of numerical experiments

- 21 lot of metrics $\rightarrow$ lot of experiments to expect
- 22 on multiple computing platforms

30 `GOTO 10`

Reproducibility

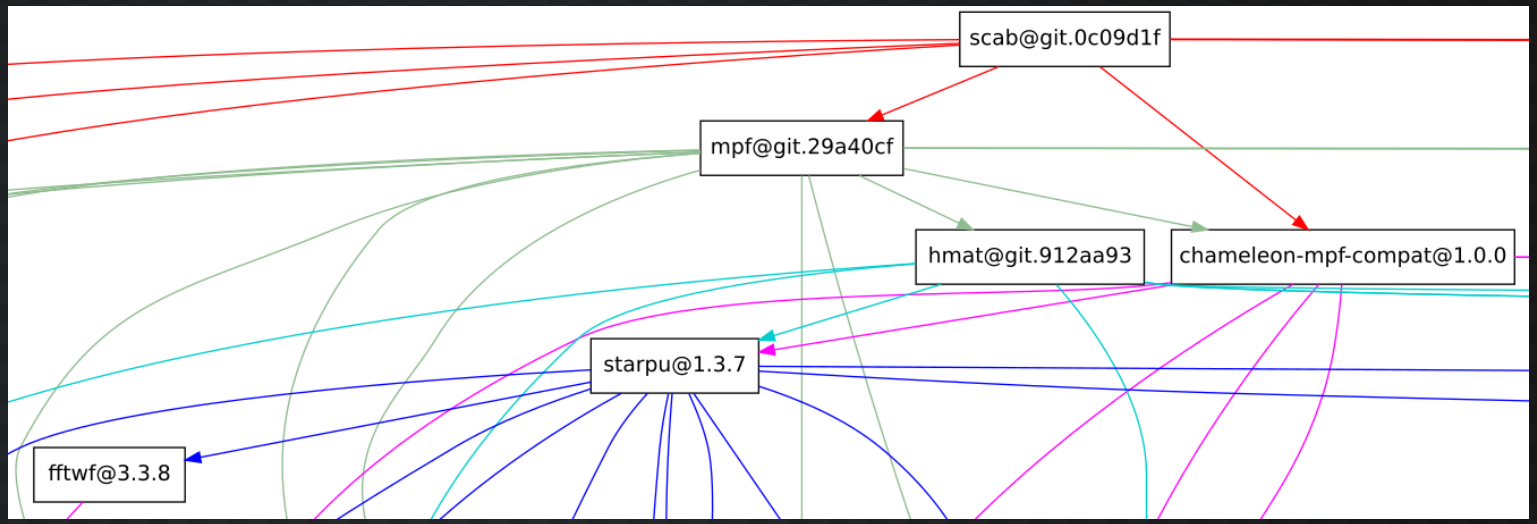
1 reproducibility of the hardware environment

2 reproducibility of the software environment

3 reproducibility of the experimental study itself

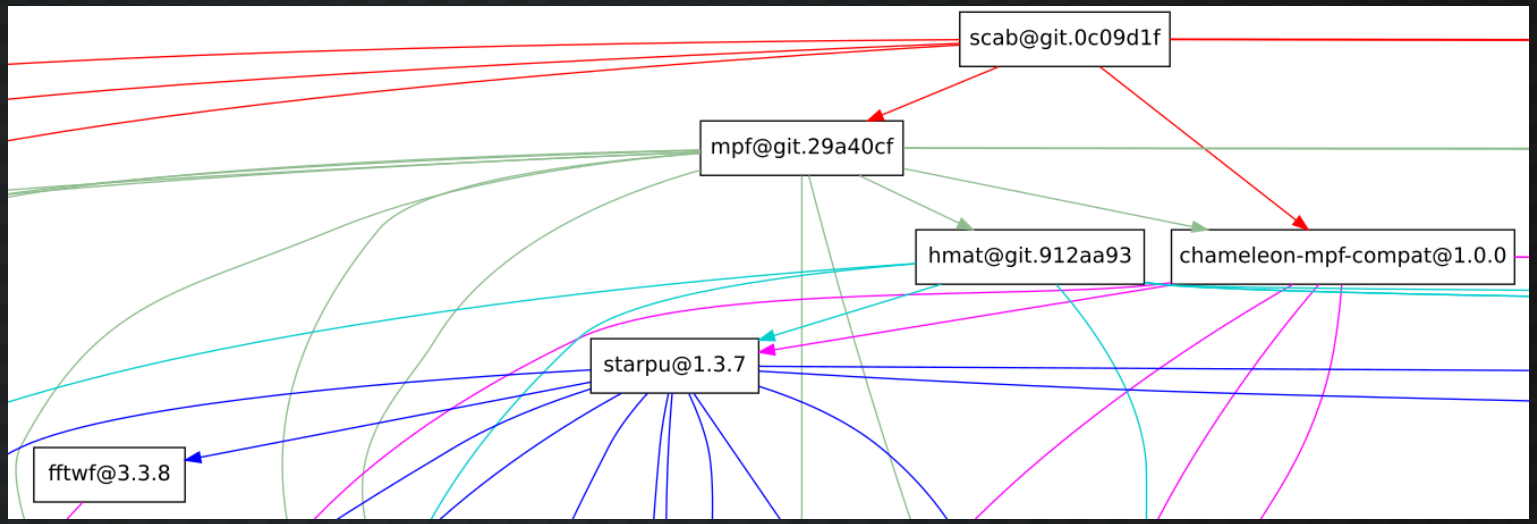
4 long-term conservation of our work

Challenges



1. reproducibility of the hardware environment
2. reproducibility of the software environment
   - complex software stack (lot of dependencies)
   - multiple variations
     - swap dependencies
     - evaluate different versions of our implementation
   - deployment on various platforms

Challenges



1. reproducibility of the hardware environment
2. reproducibility of the software environment
   - complex software stack (lot of dependencies)
   - multiple variations
     - swap dependencies
     - evaluate different versions of our implementation
   - deployment on various platforms

keep track and travel in time

Tool palette

```
apt install
make -j6
module load
```

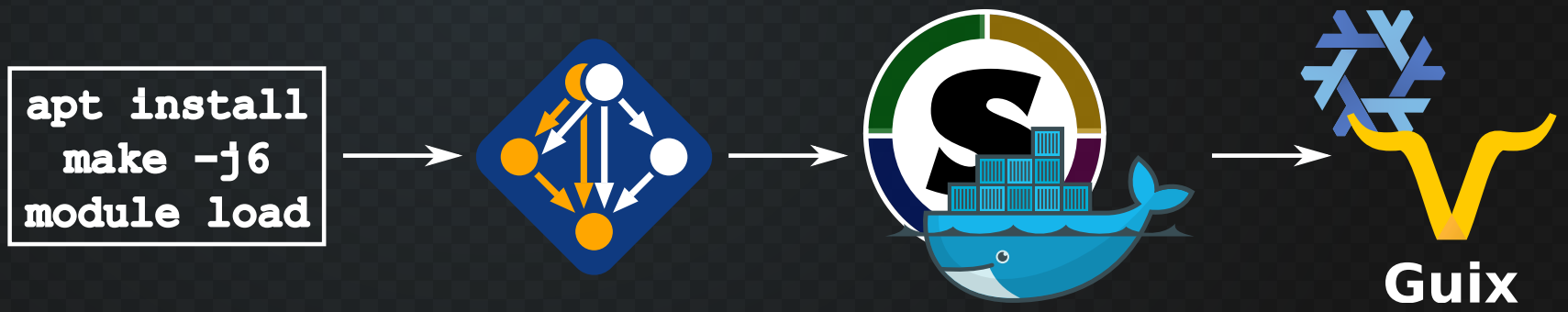
Tool palette

```
apt install
make -j6
module load
```

Tool palette

```
apt install
make -j6
module load
```

Tool palette



GNU Guix

python
python-numpy
mon-solveur

- *remplacer **OpenBLAS** par **MKL** partout*
- *pour « mon solveur » utiliser le commit **6de5b1c**...*

GNU Guix

python
python-numpy
mon-solveur

- *remplacer **OpenBLAS** par **MKL** partout*
- *pour « mon solveur » utiliser le commit **6de5b1c**...*

+

dépôt officiel « guix »
- *commit **1ac4959**...*

dépôt « guix-hpc »
- *commit **9cc4593**...*

One step further . . .

Readme Čítaj-ma

Lisez-moi

1. reproducibility of the hardware environment
2. reproducibility of the software environment
3. reproducibility of the experimental study itself
   - experimental environment - *description, how to reproduce?*
   - experiences - *what?, how?, why?*
   - results - *post-treatment, interpretation*
   - publications - *research report, article, . . .*

. . . with an adequate programming paradigm . . .

Literate programming [3]

- combines source code and formatted text → « weave » and « tangle »
- allows for a free order of source code blocks
- WEB . . . Org mode for Emacs [2]

```
#+PROPERTY: header-args :tangle rss.py ...
...
Memory usage statistics of a particular process are stored in
~/proc/<pid>/statm~ where ~<pid>~ is the process identifier
(PID). In this file, the field =VmRSS= holds the amount of
real memory used by the process at instant $t$. See the
associated function below.

#+BEGIN_SRC python
def rss(pid):
    with open("/proc/%d/statm" % pid, "r") as f:
        line = f.readline().split();
        VmRSS = int(line[1])
        return VmRSS
#+END_SRC
...
```

Memory usage statistics of a particular process are stored in `/proc/<pid>/statm` where `<pid>` is the process identifier (PID). In this file, the field `VmRSS` holds the amount of real memory used by the process at instant t. See the associated function below.

```
def rss(pid):
    with open("/proc/%d/statm" % pid, "r") as f:
        line = f.readline().split();
        VmRSS = int(line[1])
        return VmRSS
```

```
#!/usr/bin/env python3
...
def rss(pid):
    with open("/proc/%d/statm" % pid, "r") as f:
        line = f.readline().split();
        VmRSS = int(line[1])
        return VmRSS
...
```

Example A.1: how to reproduce a study?

Table of Contents

Date: 13/04/2022 | 18:11:11
Author: Emmanuel Agullo, Marek Felšöci, Guillaume Sylvand
Email: emmanuel.agullo@inria.fr, marek.felsoci@inria.fr, guillaume.sylvand@airbus.com

number of threads per MPI process. The `parallel{map}`, `parallel{rank}` and `parallel{bind}` keys indicate the mapping, the ranking and the binding of the MPI processes, respectively.

`dense` sets the parameter of the dense solver. Although in this first benchmark definition we consider only one dense solver configuration, it is not always the case and this way we shall be able to reuse the same template files.

The Cartesian product of all the map tuplets under `template_instantiation` gives the total number of generated benchmarks. The `batch` in the `job` map allows us to group multiple benchmarks into a single *slurm* job for a more efficient job schedule (see Section 3.2).

Note that `&IN_CORE` and `&PARALLEL_DEFAULT` are Yaml aliases to the corresponding data allowing us to reuse them later in the document using `*IN_CORE` and `*PARALLEL_DEFAULT`, respectively.

```
-
  id: "ic-multi-solve-{job[batch]}-{dense[solver]}-{job[nbrhs]}-\
  {job[nbpts]}"
  template_files: &IN_CORE [ "wrapper-in-core", "sbatch-in-core" ]
  template_instantiation:
    scheduler:
      - { prefix: "ic-multi-solve", platform: "plafrim", family: "miriel",
          nodes: 1, tasks: 24, time: "1-03:00:00" }
    parallel:
      - &PARALLEL_DEFAULT { np: 1, nt: 24, map: "node", rank: "node",
        bind: "none" }
    job:
      # N = 1M
      - { nbpts:  1000000, nbrhs: 128, batch:  1 }
      - { nbpts:  1000000, nbrhs: 256, batch:  1 }
      - { nbpts:  1000000, nbrhs: 512, batch:  1 }
      # N = 3M
      - { nbpts:  3000000, nbrhs: 128, batch:  1 }
      - { nbpts:  3000000, nbrhs: 256, batch:  1 }
      - { nbpts:  3000000, nbrhs: 512, batch:  1 }
      # N = 7M
      - { nbpts:  7000000, nbrhs: 128, batch:  2 }
      - { nbpts:  7000000, nbrhs: 256, batch:  3 }
      - { nbpts:  7000000, nbrhs: 512, batch:  4 }
    dense:
      - { solver: "spido" }
```

Follows the task corresponding to this benchmark. In this case, we only have to indicate the `options` of `test_FEMBEM` specific to this set of benchmarks.

```
Tasks:
```

Example A.2: research report in HTML

Table of Contents

Date: 13/04/2022 | 18:11:16
Author: Emmanuel Agullo, Marek Felšöci, Guillaume Sylvand
Email: emmanuel.agullo@inria.fr, marek.felsoci@inria.fr, guillaume.sylvand@airbus.com

In the *compressed Schur multi-factorization* variant (see Fig. 6), we compress the X_{ij} Schur block into a temporary compressed matrix as soon as the sparse solver returns it. Hence, the final assembly step becomes a compressed assembly $A_{ss_{ij}} \leftarrow A_{ss_{ij}} + \texttt{Compress}(X_{ij})$. Like in the case of *compressed Schur multi-solve*, this operation implies a recompression of the initally compressed $A_{ss_{ij}}$.

4 Experimental study

4.1 Multi-solve

4.1.1 Single-node out-of-core

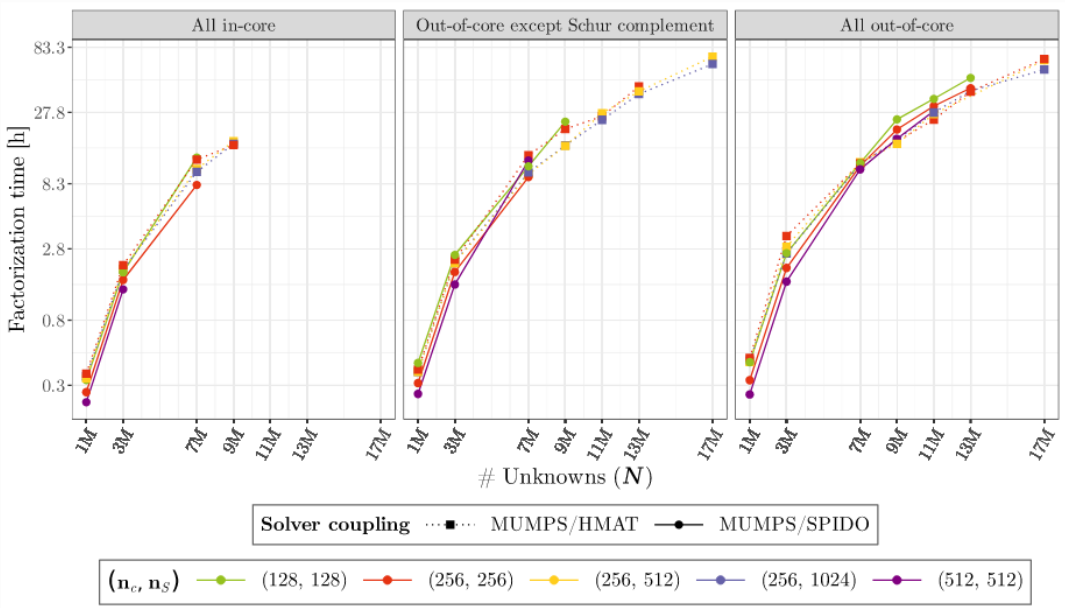


*Figure 7: Computation times of **multi-solve** on coupled FEM/BEM linear systems of varying number of unknowns for both of the solver couplings MUMPS/HMAT and MUMPS/SPIDO and for varying values of n_c and n_S. We test 3 different configurations of the out-of-core feature: completely disabled, enabled **except** for the Schur complement matrix or enabled **including** for the Schur complement matrix. Parallel runs on single `miriel` node.*

Example A.3: research report in PDF

Inria

Parallel distributed coupled solvers for large sparse/dense FEM/BEM linear systems implementing low-rank compression and out-of-core computation

Emmanuel Agullo, Marek Felšöci, Guillaume Sylvand

RESEARCH REPORT
N° ????
Fromulary 3014
Project-Team CONCACE

ISSN 0249-6399 ISRN INRIA/RR--????-FR+ENG

solvers and exploit their most advanced features such as compression techniques [9, 3, 10, 1] in an effort to lower the memory footprint and potentially reduce the computation time so as to process larger problems. In this section, we present the main algorithmic steps of both these methods. The objective is neither to motivate them nor to describe them in details (we refer the reader to [2] for that) but to provide a high-level view of the steps and their nature (such as whether they involve dense, sparse or compressed computation). Both methods must assemble the following dense matrix $S = A_{ss} - A_{sv}A_{vv}^{-1}A_{sv}^T$ associated with the A_{ss} block and referred to as the Schur complement.

2.2.1 Multi-solve algorithm.

Most sparse direct solvers do not provide an API to handle coupled sparse/dense systems and can process exclusively sparse systems. The multi-solve approach accommodates with this constraint by delegating only the A_{vv} block to the sparse direct solver. Using the latter, the A_{vv} block is factorized through a so-called *sparse factorization*. The A_{ss} block is handled by the dense direct solver. Because this block may not fully fit in memory, it is split into multiple vertical slices (see Fig. 3) which are assembled one by one, all the processing units tackling the same slice i at the same time. To compute such a slice S_i of A_{ss}, a slice A_{sv_i} is first processed through a *sparse solve* step of the sparse direct solver, yielding a dense temporary slice Y_i. The latter is multiplied by the sparse A_{sv} block. Then, we perform a final assembly $(A_{ss_i} - A_{sv}Y_i)$ to produce the dense S_i slice.

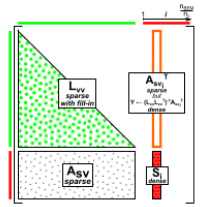


FIGURE 3: *baseline multi-solve.*

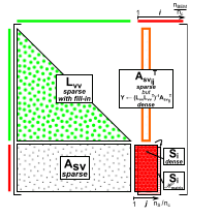


FIGURE 4: *compressed Schur multi-solve.*

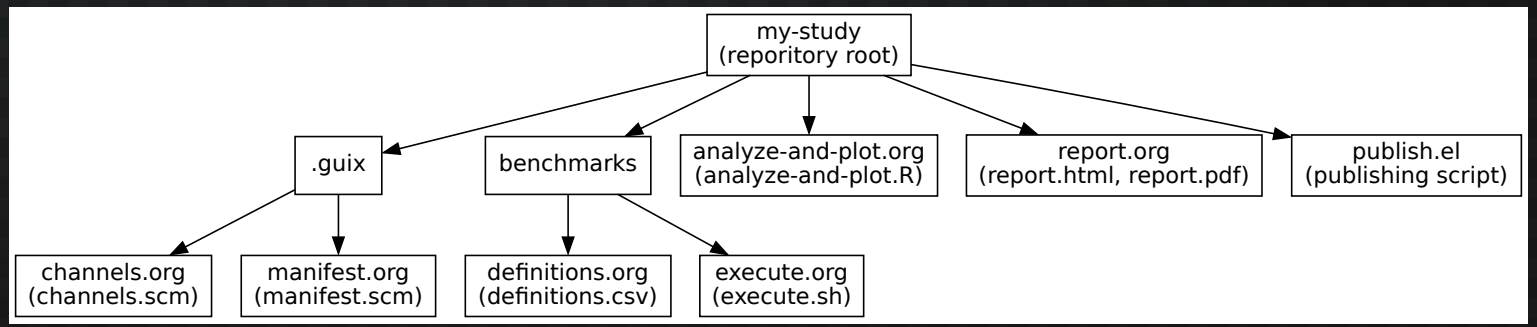
In the *baseline multi-solve* case, the block S_i is kept dense. Conversely, in the *compressed Schur multi-solve* variant, it is compressed (through hierarchically low-rank techniques). Note that A_{ss_i} is initially compressed, but this operation implies a recompression of the block at each iteration of the loop on i. This is why this variant allows for computing multiple (typically 4 in the experiments below) slices S_i before compressing and assembling them (see Fig. 4).

2.2.2 Multi-factorization algorithm.

The multi-factorization algorithm is based on a more advanced usage of sparse direct methods consisting in delegating also the management of the dense A_{ss} block to the sparse direct solver. Only supported by a few fully-featured sparse direct solvers, this functionality (referred to as Schur) has the advantage of efficiently handling off-diagonal blocks thanks to the advanced combinatorial (such as management of the fill-in), numerical (such as low-rank compression) and computational (such as level-3 BLAS usage) features of modern sparse direct solvers when processing the off-diagonal A_{sv}^T and A_{sv} sparse-dense coupling parts (see [2] for more details). The computation of the Schur complement S in the *baseline multi-factorization* algorithm is not anymore computed by vertical slices but tile-wise. Computing a tile S_{ij} (see Fig. 5) amounts

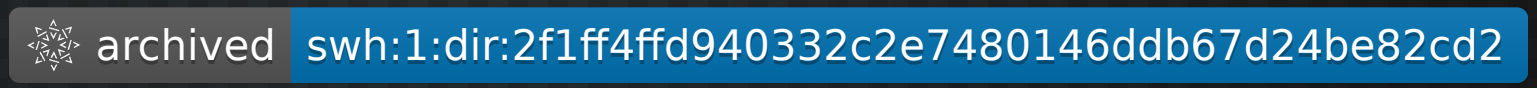
1 study = 1 git repository

- software environment specification
  - channels and manifests
- experiments
  - validation, performance, . . .
- related manuscripts
  - articles, abstracts, research reports, . . .
- can be published online (GitLab Pages)



Example B: an example study written in Org

https://gitlab.inria.fr/thesis-mfelsoci/dissertation/example-fembem [1]



[1] https://archive.softwareheritage.org/swh:1:dir:2f1ff4ffd940332c2e7480146ddb67d24be82cd2;origin=https://gitlab.inria.fr/thesis-mfelsoci/dissertation/example-fembem.git;visit=swh:1:snp:32a3c2dc1d7790103919355e603342d9e5192fad;anchor=swh:1:rev:01396c9149c59062eec2aeb0e5c21ea3b16824a2

Take-away

Reproducibility

1. reproducibility of the hardware environment
2. reproducibility of the software environment
   - → GNU Guix
3. reproducibility of the experimental study itself
   - → literate programming in Org mode
4. long-term conservation of our work

a solid basis for improving
the reproducibility of an experimental study

Today's tutorial

Améliorer la reproductibilité d'une étude grâce à Guix !

1. use Guix to manage the software environment
   - channels
   - manifest
2. do literate programming in Org mode
   - define some experiments
   - run the experiments
   - write a study featuring the results

References

[1] David Barrie. *First Airbus A350-900 XWB for Cathay Pacific - take off*. https://www.flickr.com/photos/curufinwe-xiane/27249076575. F-WZFX (MSN29 - to be registered as B-LRA) performing final flight before delivery - Toulouse, France (May 25, 2016).

[2] Carsten Dominik. *The Org Mode 9.1 Reference Manual*. 12th Media Services, 2018. ISBN: 9781680921656.

[3] Donald E. Knuth. “Literate Programming”. In: *Comput. J.* 27.2 (May 1984), pp. 97–111. ISSN: 0010-4620. DOI: 10.1093/comjnl/27.2.97. URL: https://doi.org/10.1093/comjnl/27.2.97.