Publish a reproducible result

Journées du réseau français de recherche reproductible

May 19, 2026 @ Bordeaux

Marek Felšöci
marek.felsoci@uvsq.fr

UNIVERSITÉ DE VERSAILLES – ST. QUENTIN EN YVELINES

$ whoami

Computer science studies (Université de Strasbourg)

- Bachelor's and Master's degrees

Ph.D. (Inria, Bordeaux) ← The Epoch (for me)

- solvers for large sparse/dense linear systems in aeroacoustics

Post-doc (Inria, Strasbourg)

- automatic task-based parallelization

Post-doc (LIP6, Paris)

- mixed precision computation

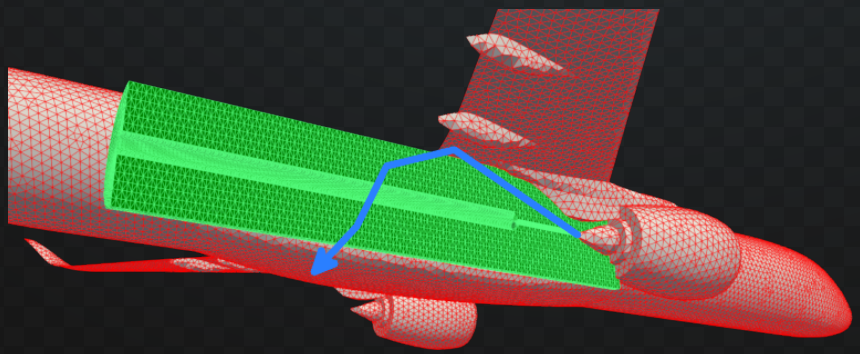
Associated professor (UVSQ, LI-PaRAD, current position)

- composability of high-performance computing applications

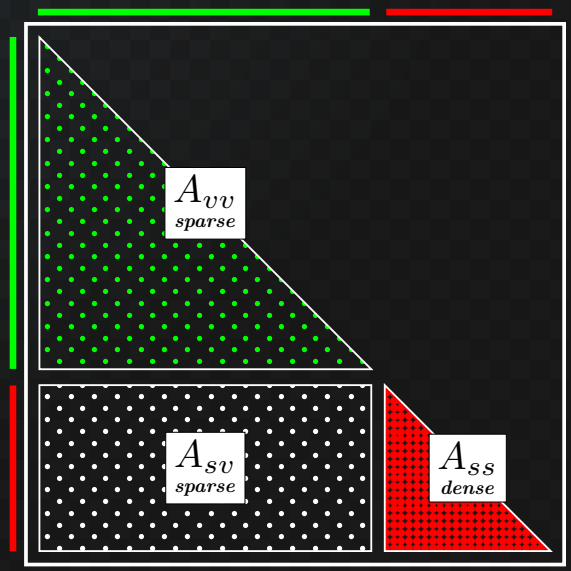
Scientific context – Sparse/dense solvers

AIRBUS Région Nouvelle-Aquitaine

- simulating propagation of soundwaves around an aircraft in flight
- solving large coupled sparse/dense linear systems
- software stack: linear solvers, linear algebra libraries, parallel processing and communication libraries, runtimes
- languages: C, C++, Fortran



A discrete numerical 3D/2D model.



sparse/dense coefficient matrix A of the corresponding linear system.

Scientific context – Automatic parallelization

```
int pivot;
partition(&pivot, data, size);
sort(&data[0], pivot);
sort(&data[pivot + 1], size - (pivot + 1));
```

- take advantage of increasingly complex parallel architectures
- task-based model
- automatically infer dependencies and find an efficient decomposition into tasks
- software stack: compilation frameworks, graph libraries, performance modelizers
- languages: OCaml, C, C++, Python

```
int pivot;
partition(&pivot, data, size);
#pragma omp task depend(inout : data[0]) ... if(cost > cutoff)
sort(&data[0], pivot);
#pragma omp task depend(inout : data[pivot + 1]) ... if(cost > cutoff)
sort(&data[pivot + 1], size - (pivot + 1));
```

Scientific context – Multi-precision computation

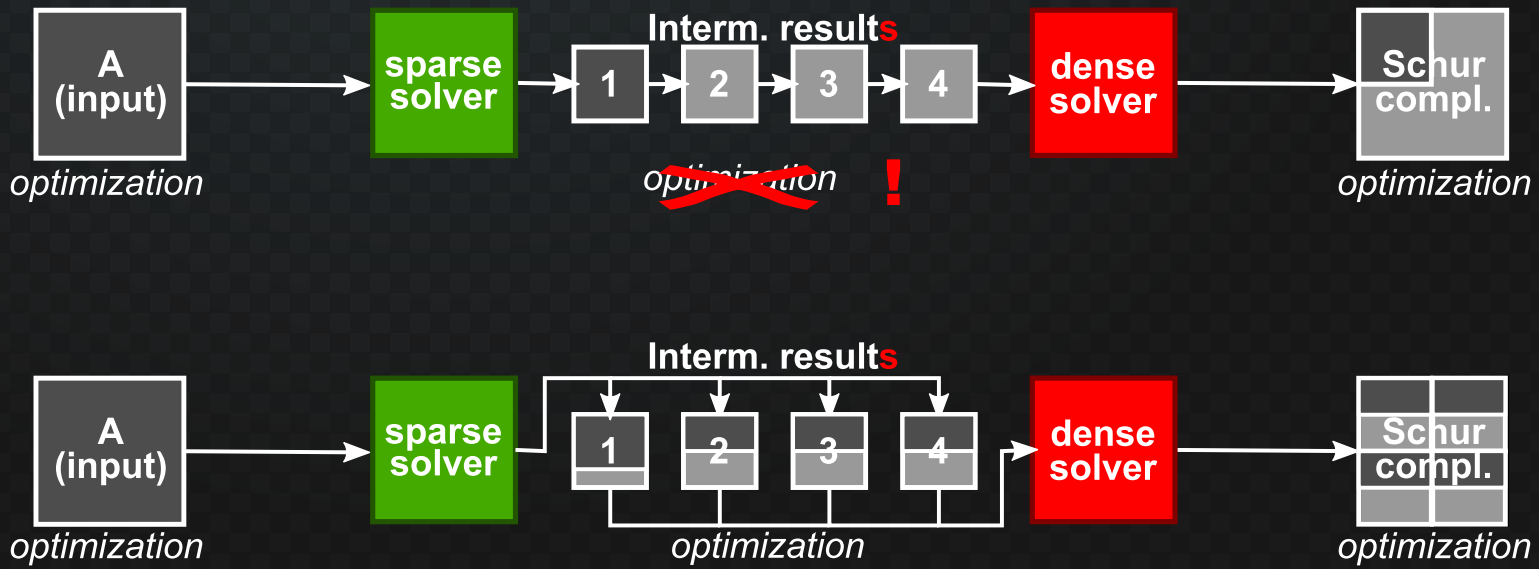
- transition from double into multi-precision computation within a acoustic numerical processing block while ensuring numerical stability of results → more performance, optimize for accelerators
- software stack: FFT (Fast Fourier transform) libraries, numerical error debuggers, precision optimizers
- languages: C, C++

```
int main(void) {
  cadna_init(-1);
  double_st a = 1.99999999999998,
            b = 1.99999999999999, c;
  a = a + 0.00000000000001;
  printf(
    "a = %s with %d correct digits\n",
    strp(a), a.nb_significant_digit()
  );
  c = a - b;
  printf(
    "c = %s with %d correct digits\n",
    strp(c), c.nb_significant_digit()
  );
  cadna_end();
  return 0;
}
```

```
------------------------------------
CADNA_C_HALF 3.1.12 software
...
------------------------------------
a =  0.199999999999999E+001 with 15
↪  correct digits
c =  @.0 with 0 correct digits
------------------------------------
CADNA_C_HALF 3.1.12 software
There is  1 numerical instability
1 LOSS(ES) OF ACCURACY DUE TO
↪  CANCELLATION(S)
------------------------------------
```

Scientific context – Composability of HPC applications

- composition of existing well-optimized HPC applications into more complex solutions
  - data exchange, numerical precision, data placement, resources sharing
- software stack: linear solvers, linear algebra libraries, parallel processing and communication libraries, runtimes, . . .
- languages: C, C++, Fortran, . . .



Numerical context

BASIC protocol

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

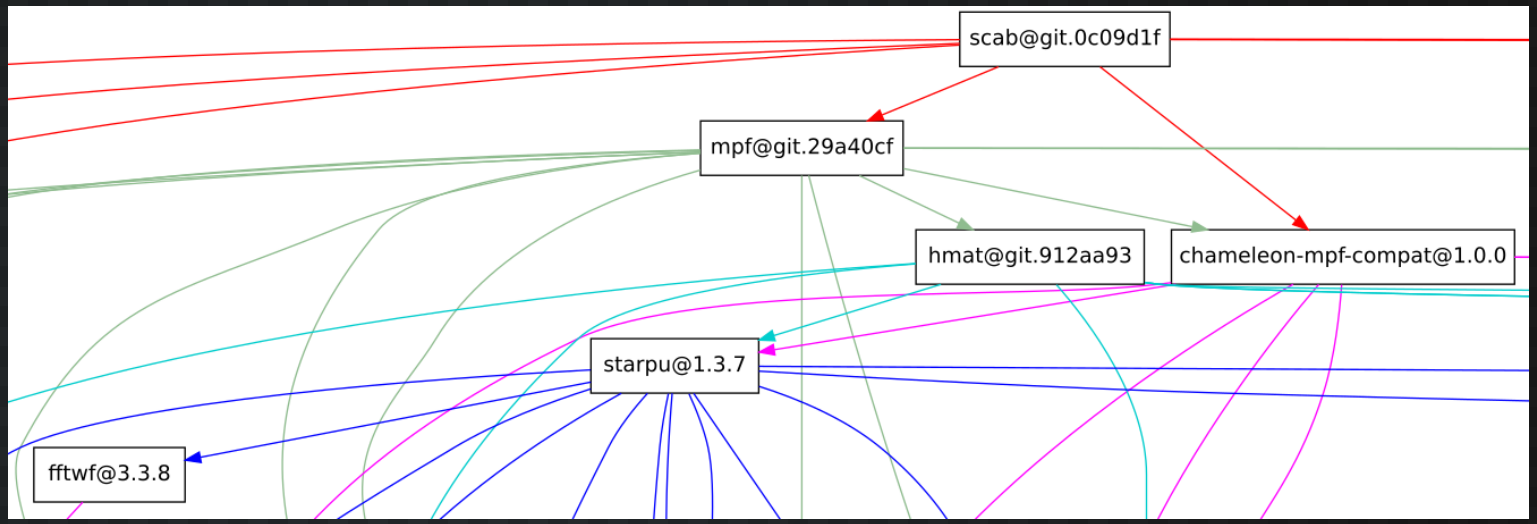
3 reproducibility of the experimental study itself

4 long-term conservation of our work

Today's tutorial

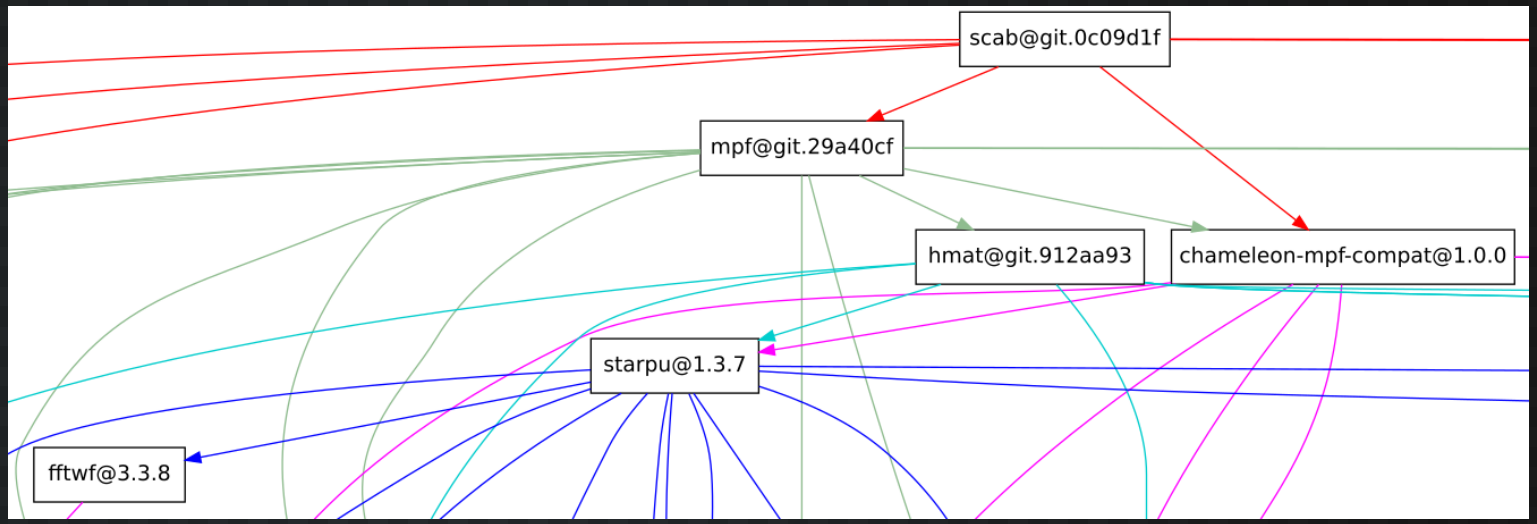
1. reproducibility of the hardware environment
2. reproducibility of the software environment ← hands-on session
3. reproducibility of the experimental study itself
4. long-term conservation of our work

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
```

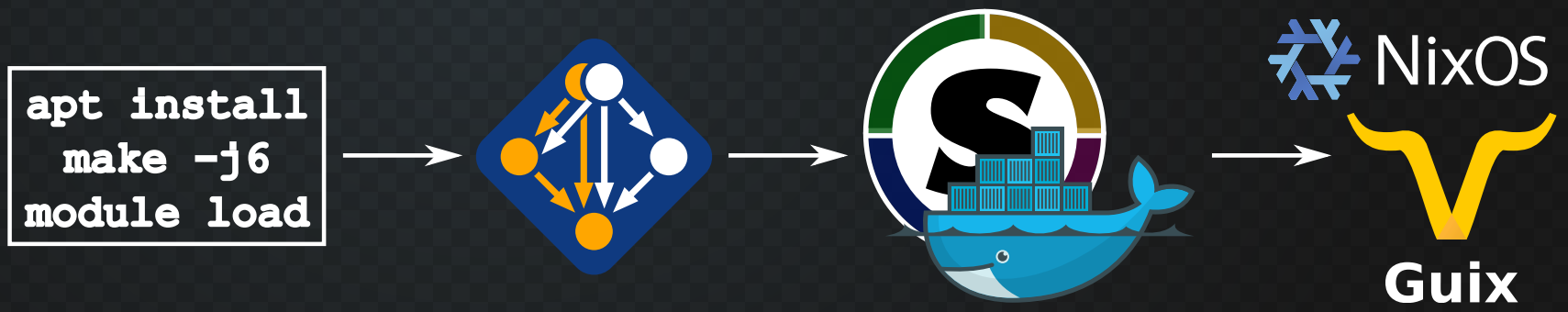
Tool palette

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

Tool palette

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

Tool palette



Let us dive into it!

Companion document

`https://p2r.felsoci.sk`

Guix

Familiarization

```
guix shell --container cowsay -- cowsay -f tux "Keep calm, use Guix"
```

Guix

Familiarization

```
guix shell --container cowsay -- cowsay -f tux "Keep calm, use Guix"
```

```
 ----------------------
< Keep calm, use Guix >
 ----------------------
   \
    \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
    /'\_   _/`\
    \___)=(___/
```

Guix

Familiarization

```
guix shell --container cowsay
```

Guix

Familiarization

```
guix shell --container cowsay
```

```
guix shell --container bash coreutils which \
     python python-pandas python-matplotlib \
     texlive-scheme-basic texlive-collection-fontsrecommended \
     texlive-type1cm texlive-underscore texlive-dvipng texlive-babel-english \
     texlive-latexmk texlive-wrapfig texlive-ulem texlive-capt-of \
     texlive-listings texlive-fancyvrb texlive-upquote texlive-lineno \
     texlive-biblatex texlive-biber texlive-xcolor \
     minisolver
```

Guix

Familiarization

```
guix shell --container cowsay
```

```
guix shell --container bash coreutils which \
     python python-pandas python-matplotlib \
     texlive-scheme-basic texlive-collection-fontsrecommended \
     texlive-type1cm texlive-underscore texlive-dvipng texlive-babel-english \
     texlive-latexmk texlive-wrapfig texlive-ulem texlive-capt-of \
     texlive-listings texlive-fancyvrb texlive-upquote texlive-lineno \
     texlive-biblatex texlive-biber texlive-xcolor \
     minisolver
```

`--container` or `--pure`?

Guix

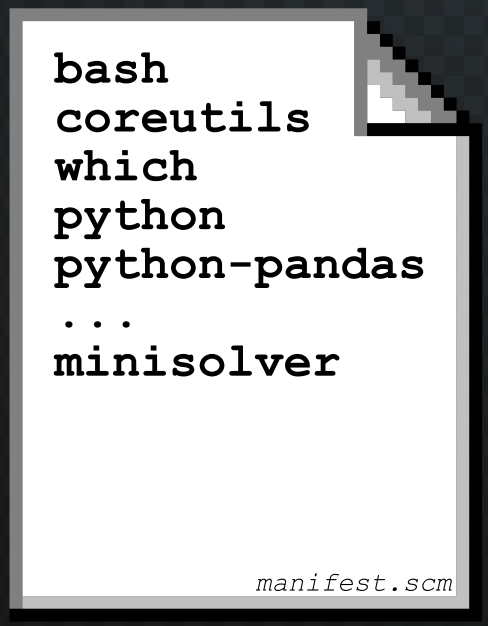
Manifests

```
guix shell --container bash coreutils which \
     python python-pandas python-matplotlib \
     texlive-scheme-basic texlive-collection-fontsrecommended \
     texlive-type1cm texlive-underscore texlive-dvipng texlive-babel-english \
     texlive-latexmk texlive-wrapfig texlive-ulem texlive-capt-of \
     texlive-listings texlive-fancyvrb texlive-upquote texlive-lineno \
     texlive-biblatex texlive-biber texlive-xcolor \
     minisolver
```

Guix

Manifests

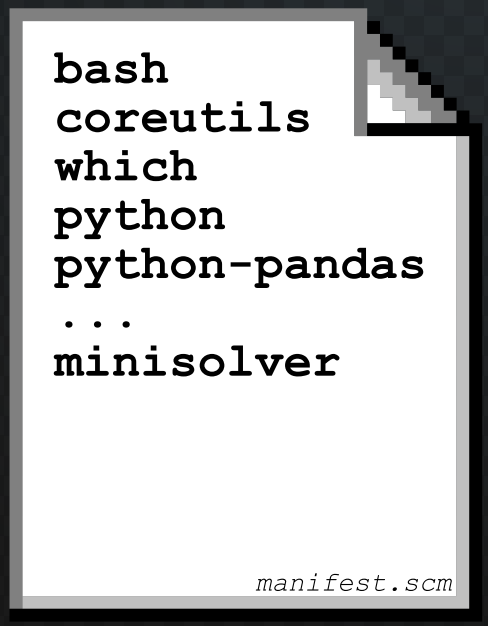
```
guix shell --container bash coreutils which \
     python python-pandas python-matplotlib \
     texlive-scheme-basic texlive-collection-fontsrecommended \
     texlive-type1cm texlive-underscore texlive-dvipng texlive-babel-english \
     texlive-latexmk texlive-wrapfig texlive-ulem texlive-capt-of \
     texlive-listings texlive-fancyvrb texlive-upquote texlive-lineno \
     texlive-biblatex texlive-biber texlive-xcolor \
     minisolver
```



Guix

Manifests

```
guix shell --container bash coreutils which \
     python python-pandas python-matplotlib \
     texlive-scheme-basic texlive-collection-fontsrecommended \
     texlive-type1cm texlive-underscore texlive-dvipng texlive-babel-english \
     texlive-latexmk texlive-wrapfig texlive-ulem texlive-capt-of \
     texlive-listings texlive-fancyvrb texlive-upquote texlive-lineno \
     texlive-biblatex texlive-biber texlive-xcolor \
     minisolver
```

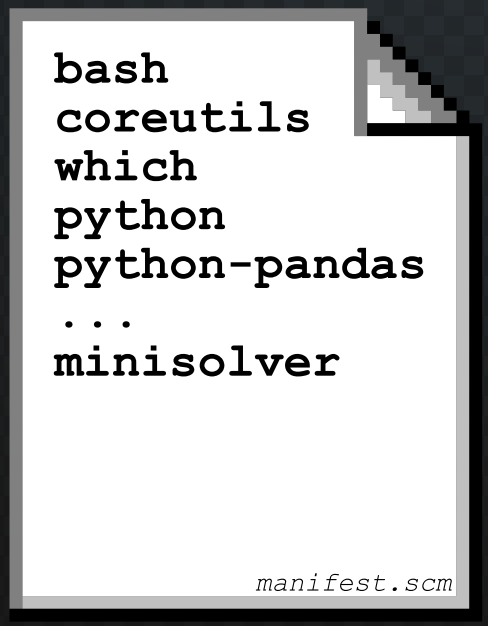



```
(specifications->manifest
 (list
  "bash"
  "coreutils"
  "which"
  "python"
  "python-pandas"
  "python-matplotlib"
  "texlive-scheme-basic"
  ;; ...
  "minisolver"))
```

Guix

Manifests

```
guix shell --container bash coreutils which \
     python python-pandas python-matplotlib \
     texlive-scheme-basic texlive-collection-fontsrecommended \
     texlive-type1cm texlive-underscore texlive-dvipng texlive-babel-english \
     texlive-latexmk texlive-wrapfig texlive-ulem texlive-capt-of \
     texlive-listings texlive-fancyvrb texlive-upquote texlive-lineno \
     texlive-biblatex texlive-biber texlive-xcolor \
     minisolver
```




```
(specifications->manifest
 (list
  "bash"
  "coreutils"
  "which"
  "python"
  "python-pandas"
  "python-matplotlib"
  "texlive-scheme-basic"
  ;; ...
  "minisolver"))
```

```
guix shell --container --manifest=manifest.scm -- echo "Hello world"
```

Guix

Channels

- git repositories defining available software packages
- public or private (Airbus, Thalès, . . .)
- global, per-user, temporary

Guix

Channels

- git repositories defining available software packages
- public or private (Airbus, Thalès, . . .)
- global, per-user, temporary

Guix

Channels

- git repositories defining available software packages
- public or private (Airbus, Thalès, . . .)
- global, per-user, temporary


```
(list
 (channel
  (name 'guix)
  (url "https://git.guix.gnu.org/guix.git")
  (branch "master")
  (commit "5bb8ff2edd90725447352l3e385efba1c35c60ef")
  ;; ...
 (channel
  (name 'jrfrr-2026-p2r)
  (url "https://gitlab.com/jrfrr-2026-p2r/channel.git")
  (branch "main")
  (commit "d14fee41b8f536f089916df9a39432216cfec2cc")))
```

Guix

Reproducible environment specification

Guix

Reproducible environment specification


```
guix time-machine --channels=channels.scm -- \
     shell --container --manifest=manifest.scm -- \
     echo "Hello world"
```

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
   - experiences
   - results
   - publications

. . . with literate programming[1] in Org mode[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.

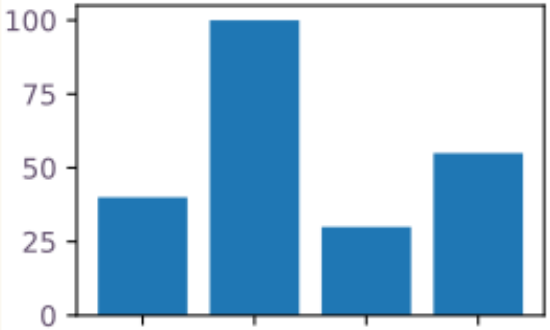
```
def rss(pid):
    with open("/proc/%d/statm" % pid, "r") as f:
        line = f.readline().split();
        VmRSS = int(line[1])
        return VmRSS
```

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

[1] Donald E. Knuth, 1984, Cumput. J., pp. 97 – 111, ISSN: 0010-4620
[2] Carsten Dominik, 2018, 12th Media Services, ISBN: 9781680921656

Example A: writing a paper in Org mode

```
Let us draw an illustrative chart of fruit supply using =matplotlib= for Python.
For this, we have to create a mock dataset, at first.

#+begin_src python :session fruit :results none
fruits = ['apple', 'kiwi', 'cherry', 'orange']
counts = [40, 100, 30, 55]
#+end_src

Now, we are ready to draw our chart and save it to a file, i.e. =fruit.svg=,

#+NAME: PlotFruit
#+HEADER: :var name="./figures/fruit.svg"
#+begin_src python :session fruit :results file link
import matplotlib.pyplot as plot
plot.figure(figsize=(3,2))
plot.bar(fruits, counts)
plot.savefig(name)
name
#+end_src

which we show in Fig. [[FigFruit]].

#+CAPTION: A nice plot of our fruit supplies.
#+NAME: FigFruit
#+RESULTS: PlotFruit
```



Example B.1: how to reproduce a study?

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 B.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

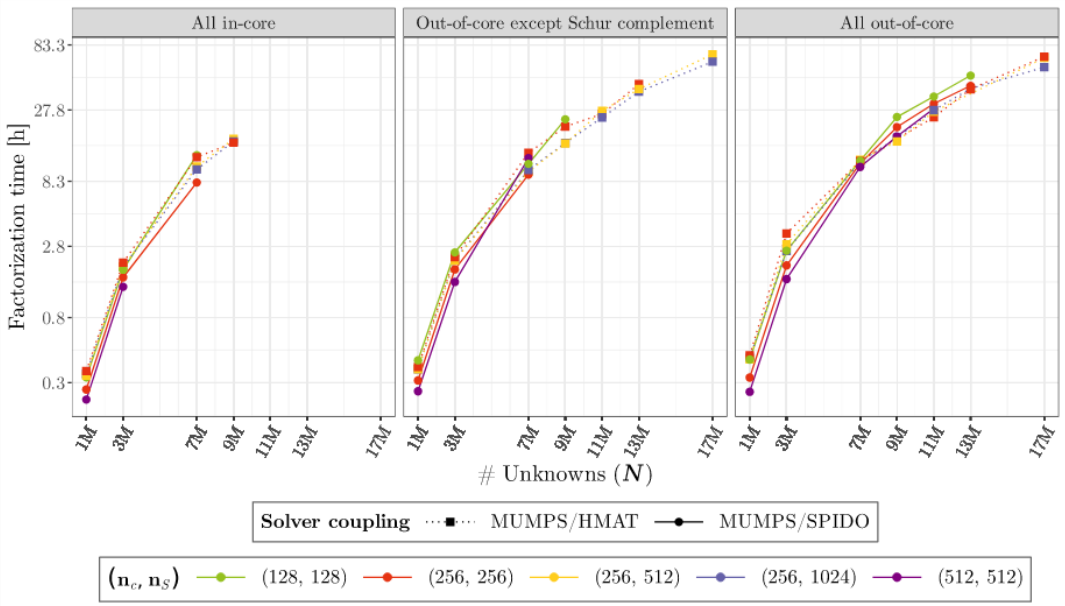


*Figure 7: Computation times of **multi-solve** on coupled FEM/BEM linear systems of varying number of unknowns for both of the solver couplings MUMPS/HMAT and MUMPS/SPIDO and for varying values of n_c and n_S. We test 3 different configurations of the out-of-core feature: completely disabled, enabled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

Example B.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.

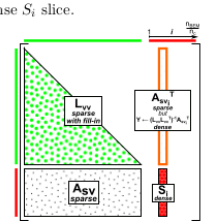


FIGURE 3: *baseline multi-solve.*

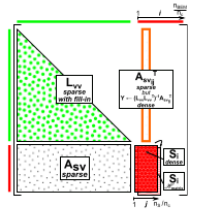


FIGURE 4: *compressed Schur multi-solve.*

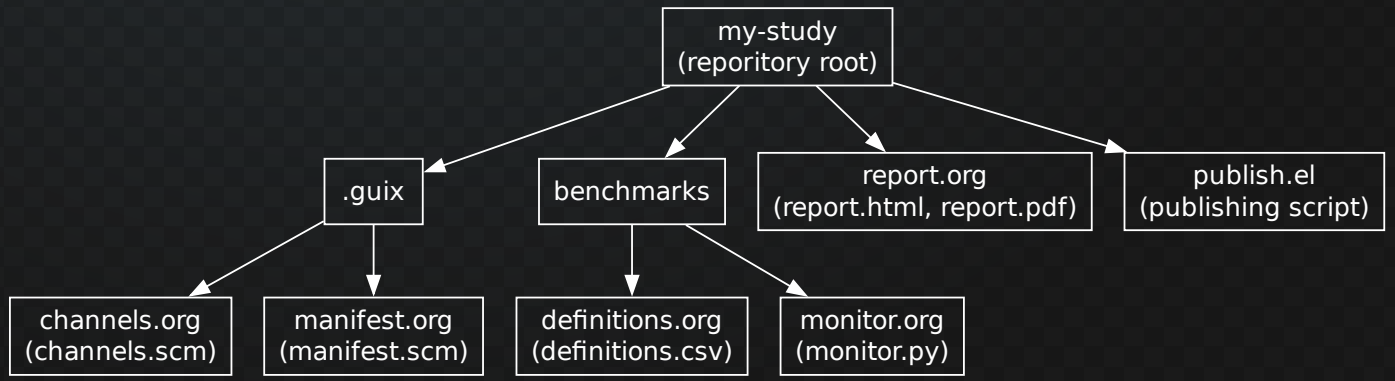
In the *baseline multi-solve* case, the block S_i is kept dense. Conversely, in the *compressed Schur multi-solve* variant, it is compressed (through hierarchically low-rank techniques). Note that A_{ss_i} is initially compressed, but this operation implies a recompression of the block at each iteration 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

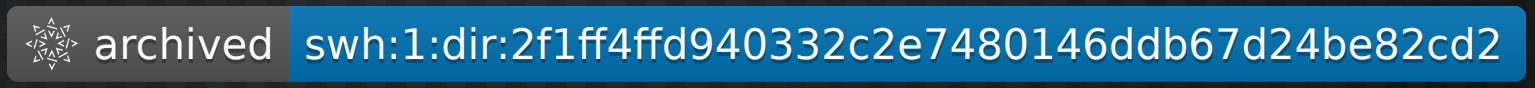
1 study = 1 git repository

- software environment specification
  - channels and manifests
- experiments
  - validation, performance, . . .
- manuscripts
  - articles, abstracts, research reports, . . .
- shareable (GitLab Pages)



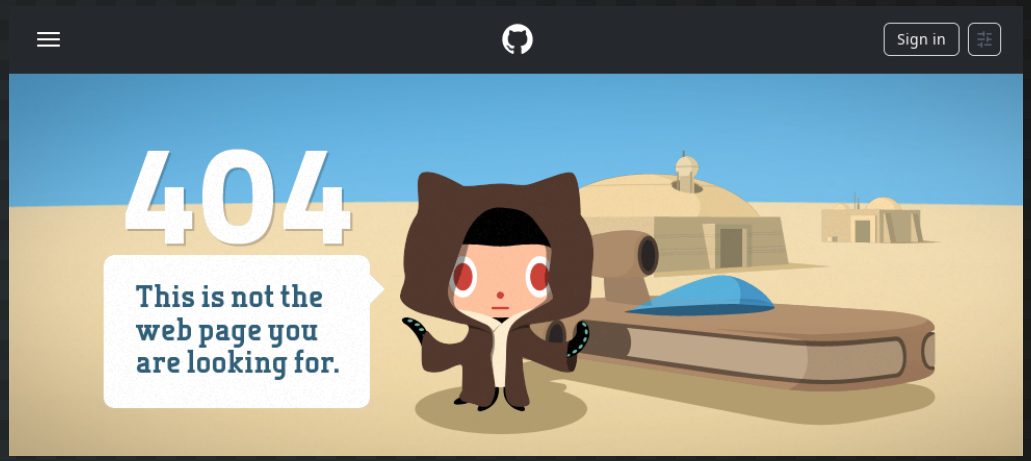
Example C: an example study written in Org

https://gitlab.inria.fr/thesis-mfelsoci/dissertation/example-fembem [3]



[3] 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

Reproducibility in time



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
   - repositories may change of hosting
   - software forges may be discontinued
     - Inria Forge, Gitorious, . . .
   - need for a durable hosting

Existing solutions

Software Heritage – `https://www.softwareheritage.org/`

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

- initiated by Inria, supported by many (CEA, Microsoft, CNRS, . . .)
- lifetime storage of repositories, focus on source code
- seamless integration with Guix
- unique identifiers `swh:...` suitable for referencing in papers

Software Availability We endeavor to make our experiments reproducible. A public companion[2] contains the instructions to reproduce our study.

[2] `https://gitlab.inria.fr/pswartva/paper-model-memory-contention-r13y`, archived on `https://www.softwareheritage.org/` with the ID `swh:1:snp:306f7c10cf69a5860587e5aad62b76070b798ecd`

Existing solutions

Zenodo – `https://zenodo.org/`

- storage at CERN Data Centre
- supported by CERN, OpenAIRE, European Commission
- storage of data, source code, artifacts 'as long as CERN exists'
- referencing in papers possible through DOI identifiers

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
   - → Software Heritage, Zenodo, . . .

a solid basis for improving
the reproducibility of an experimental study