Guix and Org mode, a powerful association for building a reproducible research study

Tuto Techno - GUIX HPC
June 14, 2022

Marek Felšöci
marek.felsoci@inria.fr

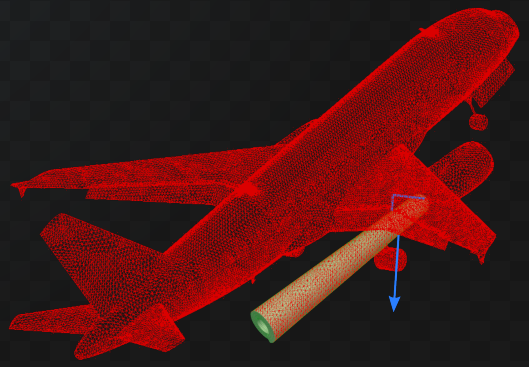
Fast solvers for high-frequency aeroacoustics

AIRBUS

RÉGION Nouvelle-Aquitaine

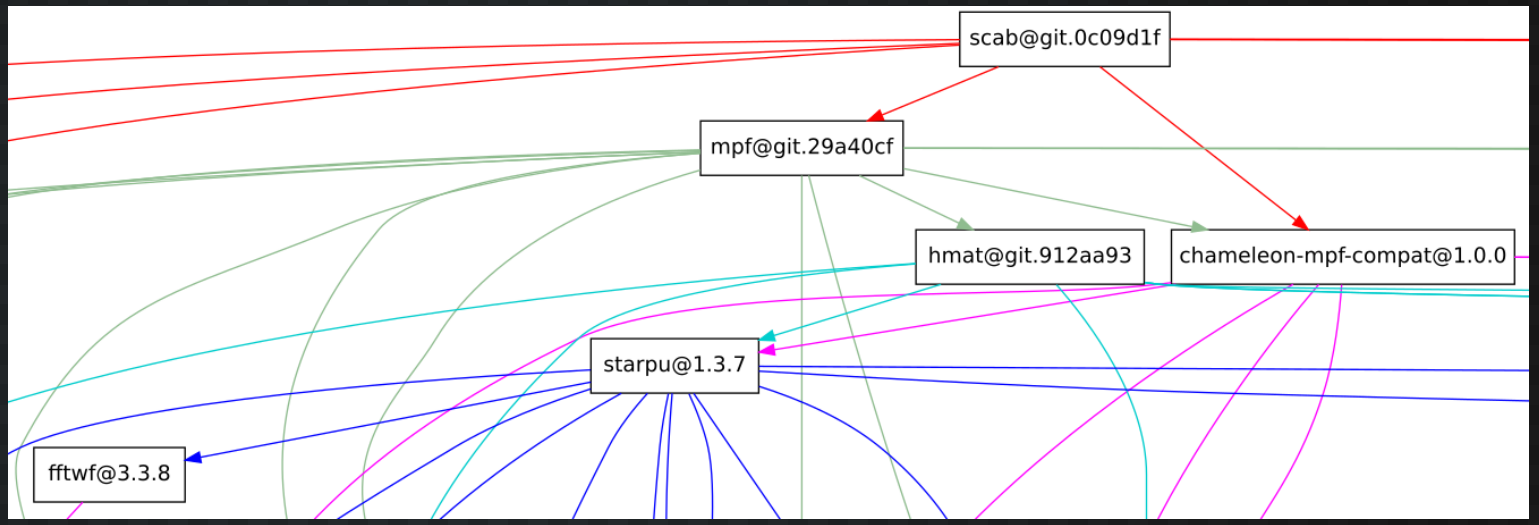
- numerical simulations for studying the propagation of sound waves emitted by an aircraft
- solving large coupled sparse/dense linear systems

A real-life example [8].



A discrete numerical model.

Challenges



1. rich software environment
   - lot of dependencies
   - multiple versions
   - different computing platforms
2. benchmark campaigns
   - easily reproducible and extensible
   - automatized

Choosing the right tools

1. reproducible software environment
   - efficient management of multiple environments
   - reproducibility across different computing platforms
   - Modules, Spack, Singularity . . . GNU Guix [2]

```
guix shell --pure --with-input=pastix-5=pastix-5-mkl \
     --with-input=mumps-scotch-openmpi=mumps-mkl-scotch-openmpi \
     --with-input=openblas=mkl --with-git-url=gcvb=$HOME/src/gcvb \
     --with-commit=gcvb=40d88ba241db4c71ac3e1fe8024fba4d906f45b1 \
     --preserve=^SLURM bash coreutils inetutils findutils grep sed \
     bc openssh python python-psutil gcvb slurm@19 openmpi scab
```

Choosing the right tools

2 reproducible experiments

- literate programming [7]
- detailed documentation of environments and experiments
- Org mode for Emacs [4]

```
#+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.

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

```python
#!/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
...
```

Constraints

- different environments
  - benchmark execution
  - result post-processing
- multiple versions of our solver stack
  (after implementing new algorithms or improvements)
- multiple platforms (with or without Guix)
  1. PlaFRIM (Inria Bordeaux) [5]
  2. Curta (Nouvelle Aquitaine) [1]
  3. Jean Zay (IDRIS) [3]

Software sources

Guix channels

- Git repositories defining available packages
- public or private (Airbus)
- defined globally, per-user or in a standalone Scheme file

```
(list (channel (name 'guix)
               (url "https://git.savannah.gnu.org/git/guix.git")
               (commit "1ac4959c6a94a89fc8d3a73239d107cfb1d96240"))
 (channel (name 'guix-hpc)
          (url "https://gitlab.inria.fr/guix-hpc/guix-hpc.git")
          (commit "9cc4593aaaaeaf17a602b620e9ab1974b5b82984"))
 (channel (name 'guix-hpc-airbus)
          (url "git@gitlab.inria.fr:mfelsoci/guix-hpc-airbus.git")
          (commit "bff0aa3f20140dbfda53d6882e25d3e9770e2911"))
...
```

Specifying and entering an environment

```
guix shell --pure option1 option2 package1 package2 ...
```

Manifests

- Scheme files containing definitions of software environments
- prevent long `guix shell` commands
- works with `guix pack` to create software bundles (Singularity)

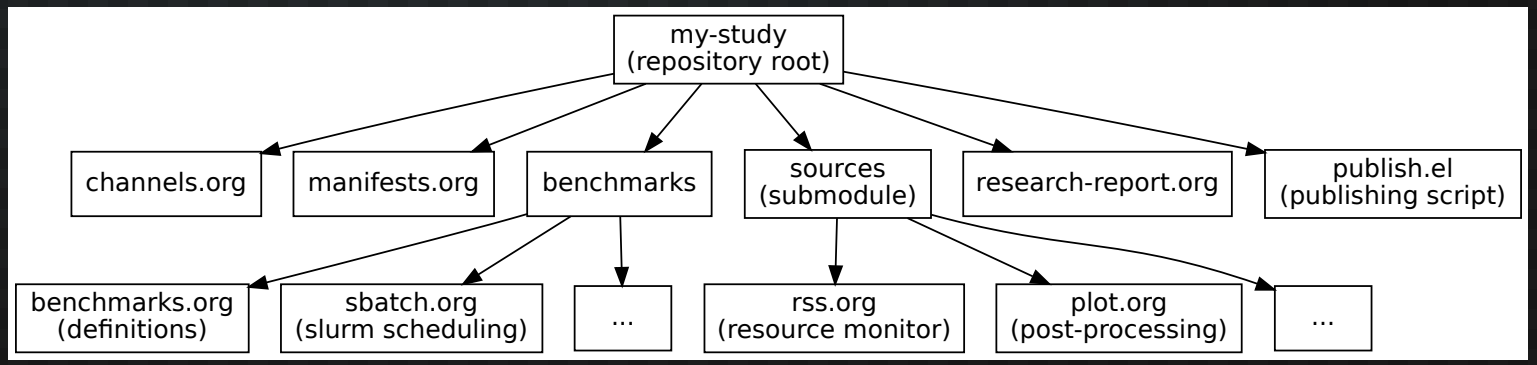
```
(define transform1
  (options->transformation
    '((with-input . "openblas=mkl")
      ;; ...
      (with-input . "slurm=slurm@19"))))
(packages->manifest
  (list (transform1 (specification->package "scab"))
        (transform1 (specification->package "gcvb-minimal-felsocim"))
        ;; ...
        (transform1 (specification->package "bash"))))
```

Going one step further...

- reproducible software environment
  - platform-independent (as much as possible)
- reproducible experiments
  - presentation of this software environment and its setup
  - exhaustive description of experiments
  - explanation of result post-processing
  - ...

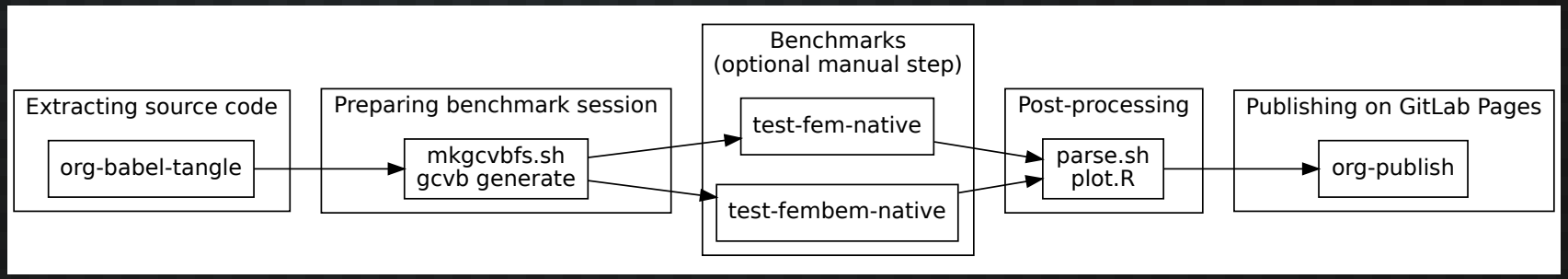
1 study = 1 repository

- environment specifications (channels and manifests)
- experiment definitions
- associated publications
- Org format preferred for text documents and source code
- everything published and shared online (GitLab Pages)



Continuous integration

- automatization of the entire process
  - environment setup, benchmark execution, result post-processing, . . .
- task sequence executed on each `git push`
- based on `guix shell` and/or Singularity bundles



Example

thesis-mfelsoci.gitlabpages.inria.fr/oocdst/

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

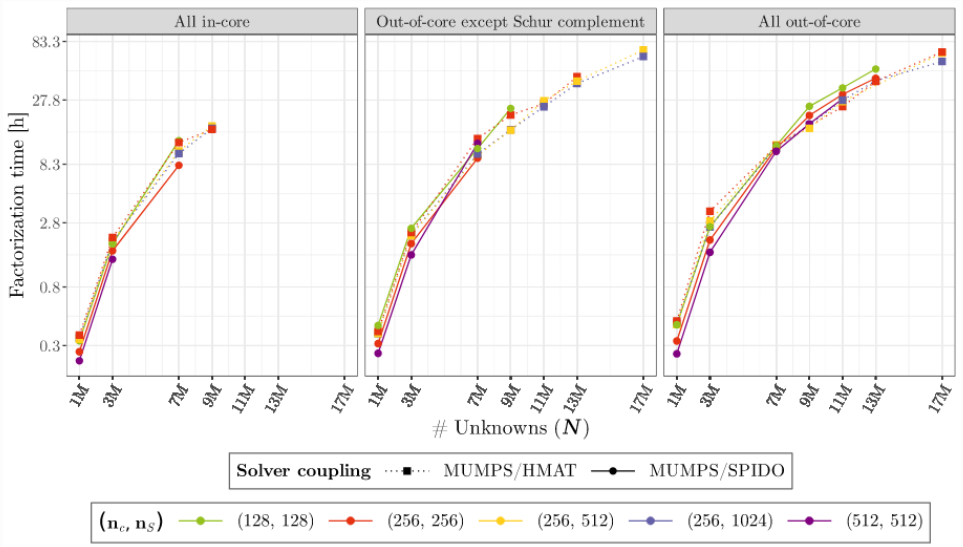


*Figure 7: Computation times of **multi-solve** on coupled FEM/BEM linear systems of varying number of unknowns for both of the solver couplings MUMPS/HMAT and MUMPS/SPIDO and for varying values of n_c and n_S. We test 3 different configurations of the out-of-core feature: completely disabled, enabled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

Example

thesis-mfelsoci.gitlabpages.inria.fr/oocdst/

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--7777--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_{sv} 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.

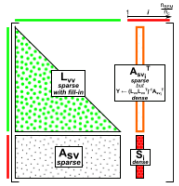


FIGURE 3: *baseline multi-solve.*

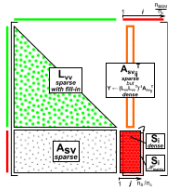


FIGURE 4: *compressed Schur multi-solve.*

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

Example

thesis-mfelsoci.gitlabpages.inria.fr/oocdst/

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:
```

Reproducibility

- reproducible software environments thanks to Guix
- exhaustive documentation in Org mode for redoing experiments
- automatized environment setup, benchmark execution and result post-processing using continuous integration
- solid basis for future reproducible research studies

Availability

- repositories may change of hosting
- software forges may be discontinued
  - Inria Forge, Gitorious, ...
- need for a durable hosting → Software Heritage [6]
  - initiated by Inria
  - lifetime storage of repositories
  - unique identifiers `swh:...` suitable for referencing in papers

Software Heritage
THE GREAT LIBRARY OF SOURCE CODE

Today's goal

Make an existing research study better reproducible using Guix and Org mode!

1. specify software environment with Guix
   - channels, packages
2. write some source code in Org mode
   - literate programming
3. reproduce the study within a Guix environment
4. document the instructions for reproducing the study
5. archive the repository on Software Heritage

References

- *Cluster Curta, Mésocentre de Calcul Intensif Aquitain.*
  https://www.mcia.fr/projects/cluster-curta.
- *GNU Guix software distribution and transactional package manager.*
  https://guix.gnu.org.
- *Institut du développement et des ressources en informatique scientifique: calculateur Jean Zay.*
  http://www.idris.fr/jean-zay/.
- *Org mode for Emacs.*
  https://orgmode.org/.
- *PlaFRIM: Plateforme fédérative pour la recherche en informatique et mathématiques.*
  https://plafrim.fr/.
- *Software Heritage.*
  https://www.softwareheritage.org/.
- D. E. Knuth, *Literate Programming*, Comput. J., 27 (1984), p. 97–111.
- Sebaso, *Jet engine airflow during take-off.*
  https://commons.wikimedia.org/wiki/File:20140308-Jet_engine_airflow_during_take-off.jpg.