Reconciling high-performance computing with the use of third-party libraries?

Seminar in the Datamove team
May 20, 2022

Emmanuel Agullo, Marek Felšöci
emmanuel.agullo@inria.fr, marek.felsoci@inria.fr

hiepacs objectives

- Design of numerical algorithms
- Parallel implementation (`MPI+threads+Cuda` vs `task-based` programming)
- Mostly (or multi-linear) linear algebra
- Application to numerical simulation and (more recently) data analysis
- Composability: tentative of the upcoming `concace` team

A few codes

Currently

- `chameleon`: dense solver (collaboration with UTK and KAUST)
- `qr_mumps` (led by A. Buttari @ CNRS/IRIT) / `pastix`: sparse direct solvers
- `fabulous`: subspace incremental solvers (*aka* iterative methods)
- `maphys`: hybrid solver (domain decomposition methods)
- `scalfmm`: fast multipole method

Four-years objective (within `concace`)

- `compose`: re-visit the core algebraic, combinatorial and numerical concepts and turn that into a composable HPC software

Close interaction with other Inria (mainly BSO) HPC teams

Runtime support

- `starpu` (`storm`): task-based runtime for heterogeneous machines (read "*PU")
- `newmadeleine` (`tadaam`): communication engine (alternative to `openmpi` ... and `mpi`)
- `hwloc` (`tadaam` and `storm`): hardware locality

Partitioner

- `scotch` (`tadaam`): graph partitioner

Applications (example)

- `hou10ni` (`makutu`): wave propagation

Some definitions

Reproducibility (*wikipedia*):

The findings of a study to be reproducible means that results obtained by an experiment or an observational study or in a statistical analysis of a data set should be achieved again with a high degree of reliability when the study is replicated.

Reproducible environment (*reproducible-builds.org*)

Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code.

Not (far from ?) a consensus

Enthusiasm (*softwareheritage.org*)

Software Heritage and GNU Guix join forces to enable long term reproducibility.

Skepticism (*from liste calcul*)

Dans de nombreux domaines scientifique, la reproductibilité au bit près n'a pas d'intérêt. C'est même sclérosant pour les codes !

Typical issue a team like ours is facing

Using a large number of third-party libraries

- hybrid solver (*e.g.* `maphys`) using one/multiple direct solvers (*e.g.* `qr_mumps`, `mumps` or `pastix`) and iterative (*e.g.* `fabulous`) robust, optimized solvers relying on fully-featured execution engines (*e.g.* `starpu` and `newmadeleine`)
- this solver is itself embedded in an application (*e.g.* `hou10ni`)

Desired properties (for a team like ours) (1/2)

Producing a correct environment (!)

- Simply being able to *produce* such a complex software environment in a reasonable time!
- Work done once in the package definitions rather than when deploying.

Reliability of the deployment

- Ensuring a end-user may have a correct and fully-featured
- On two different machines? In continuous integration?
- In time?
- Pre-processing (definition of the experimental campaign) and the post-processing (figures, articles, website, . . .) also?

Desired properties (for a team like ours) (2/2)

Collaborative development (*e.g.* starpu issue #4):

```
STARPU_FXT_TRACE=1 STARPU_FXT_PREFIX=/tmp/teststarpu guix
↪  shell --pure --preserve=^STARPU --preserve=TZDIR
↪  chameleon openssh --with-branch==starpu=fxt -L
↪  /home/eagullo/soft/project/gitlab/guix-hpc/guix-hpc --
↪  chameleon_dtesting -o potrf -n 4000 --check | sed
↪  "s/;/|/g"
```

and . . . reproducible science

Producing and reproducing a study.

Definition of maphys in spack (1/2)

```
from spack import *

class Maphys(CMakePackage):
    """a Massively Parallel Hybrid Solver."""

    homepage = "https://gitlab.inria.fr/solverstack/maphys/maphys"
    url      = homepage
    git      = url + ".git"

    version('master' , branch='master', submodules=True)
    version('develop', branch='develop', submodules=True)

    version(
        '1.0', '4e524e28402d81511e322636e1fc6c72',
        url='http://morse.gforge.inria.fr/maphys/maphys-1.0.0.tar.gz',
        preferred=True
    )
    # ...
```

Definition of maphys in spack (2/2)

```
# ...
variant('mumps', default=True, description='Enable MUMPS direct solver')
# ...
depends_on("mumps+mpi", when='+mumps')
# ...
def cmake_args(self):
    # ...
    args.extend([
        # ...
        "-DMAPHYS_SDS_MUMPS=%s" % ('ON' if spec.satisfies('+mumps') else
                                   'OFF'),
        ])
# ...
```

Remarks regarding this `spack` definition

- Elegant and compact definition of variants (`+mumps`)
- Compact definition of multiple versions (1.0, 0.9.8.3, 0.9.8.2, . . .)

Definition of maphys in guix-hpc (1/3)

```
(define-public maphys
  (package
    (name "maphys")
    (version "1.0.0")
    (home-page "https://gitlab.inria.fr/solverstack/maphys/maphys")
    (source
     (origin
      (method git-fetch)
      (uri
       (git-reference
        (url home-page)
        (commit version)
        ;; We need the submodule in 'cmake_modules/morse'.
        (recursive? #t)))
      (file-name (string-append name "-" version "-checkout"))
      (sha256
       (base32
        "0pcwfac2x574f6ggfdmahhx9v2hfswyd3nkf3bmc3cd3173312h3"))))
    (build-system cmake-build-system)
    ;; ...
```

Definition of maphys in guix-hpc (2/3)

```
'(#:configure-flags '("-DBUILD_SHARED_LIBS=ON"
                      "-DMAPHYS_BUILD_TESTS=ON"
                      "-DMAPHYS_SDS_MUMPS=ON"
                      "-DMAPHYS_SDS_PASTIX=ON"
                      "-DCMAKE_EXE_LINKER_FLAGS=-lstdc++"
                      "-DMAPHYS_ITE_FABULOUS=ON"
                      "-DMAPHYS_ORDERING_PADDLE=ON"
                      "-DMAPHYS_BLASMT=ON"
                      )

  #:phases
  (modify-phases
   %standard-phases
   ;; ...
   (add-before
    'check
    'prepare-test-environment
    (lambda _
      (setenv "OMPI_MCA_rmaps_base_oversubscribe" "1") #t)))))
```

Definition of maphys in guix-hpc (3/3)

```
    (inputs `(("hwloc" ,hwloc "lib")
              ("openmpi" ,openmpi)
              ("ssh" ,openssh)
              ("scalapack" ,scalapack)
              ("openblas" ,openblas)
              ("scotch" ,pt-scotch)
              ("mumps" ,mumps-openmpi)
              ("pastix" ,pastix-6.0.3)
              ("fabulous" ,fabulous)
              ("paddle", paddle)
              ("metis" ,metis)))
    (native-inputs `(("gfortran" ,gfortran)
                     ("pkg-config" ,pkg-config)))
))
```

Remarks on this `guix` definition

- Confidence on the deployment of the package with *all* its dependencies! (out-of-reach – for us – without a robust tool ensuring a bit-wise reproducible build)
- *variants* (`spack` terminology) / parametrized packages (`guix` terminology) are thus less important (but would still help)

Example `org-mode/spack` (Louis Poirel's PhD) (1/2: local)

```
git clone https://github.com/spack/spack.git
source ./spack/share/spack/setup-env.sh
git clone https://gitlab.inria.fr/solverstack/spack-repo.git
cat << EOF > ./spack/etc/spack/repos.yaml
repos:
- `pwd`/spack-repo
EOF
spack install -vj 4 maphys +pastix +mumps -paddle
↪  ^flex@2.6.0 ^mumps -scotch
+metis ^pastix -scotch
```

Example `org-mode/spack` (Louis Poirel's PhD) (2/2: `occigen`)

```
source $SHAREDSCRATCHDIR/test_spack/spack/share/spack/setup-env.sh

module load intel/17.0
module load hwloc/1.11.0
module load intelmpi/2017.0.098
module load cmake/3.5.2
module load parmetis/4.0.3-real64

spack load maphys
spack load python
spack load pastix

export PYTHONPATH=${SHAREDSCRATCHDIR}/libs/lib/python2.7/site-packages/
```

Most of the complexity embedded in the package definition (*e.g.* maphys spack)

Remarks on this `org-mode/spack` example

Compactness, flexibility et reliability

- Compactness: *vs* step by step install
- Flexibility: modules spécifiques à la machine
- Reliability: modules provided by the administrators (*e.g.* `mpi` module) are inter-operable with the `slurm` daemon

Limits ?

- Some adjustment is required
  - per platform
  - in time (modules evolve, . . .)
- Which guarantee of the compatibility of the modules (*e.g.* version of `parmetis`) provided by the administrators with the software stack deployed via `spack` (*e.g.* version of `mumps`)?

Example `org-mode/guix`

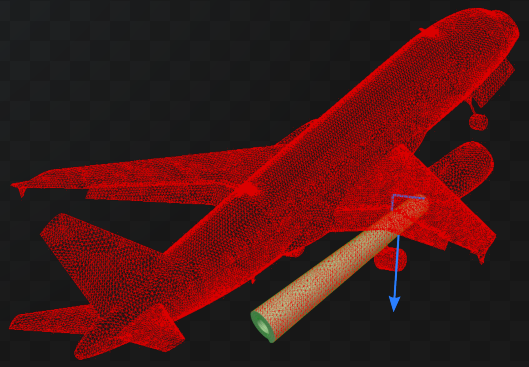
PhD thesis of Marek Felsoci.

Fast solvers for high-frequency aeroacoustics

AIRBUS RÉGION Nouvelle-Aquitaine

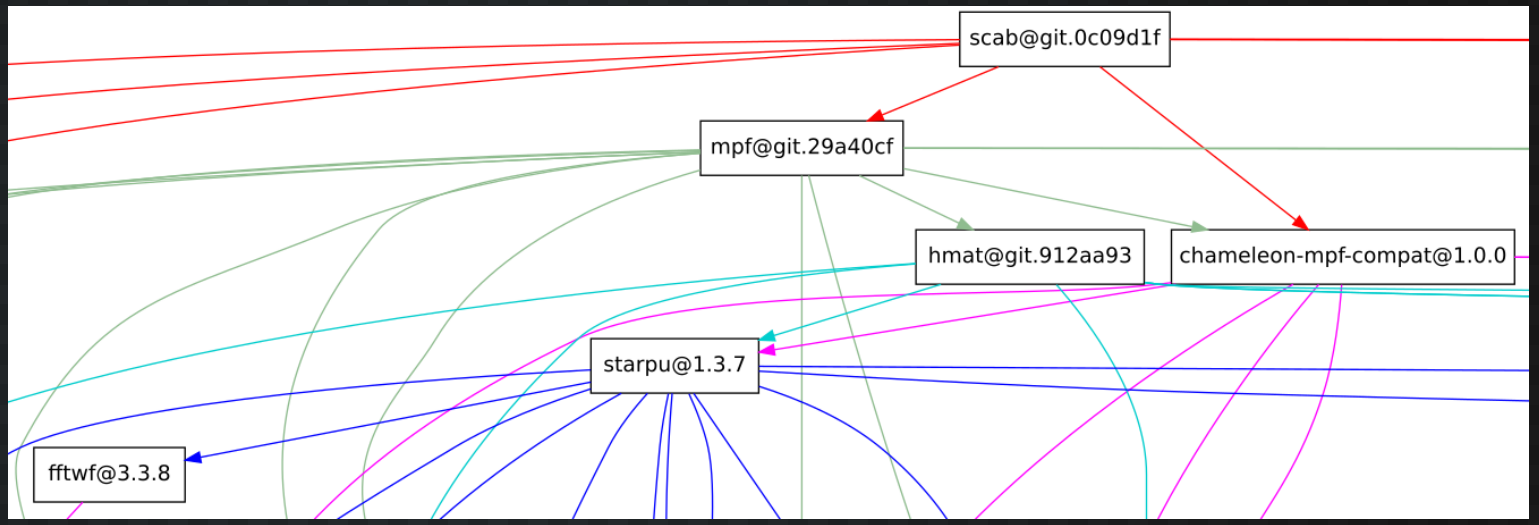
- numerical simulations for studying the propagation of sound waves emitted by an aircraft
- solving large coupled sparse/dense linear systems

A real-life example [7].



A discrete numerical model.

Challenges



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

Guix and Org mode

1. Guix [2] package manager
   - efficient management of multiple environments
   - reproducibility across different computing platforms

```
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
```

Guix and Org mode

2 Org mode for Emacs [4]

- literate programming [6]
- detailed documentation of environments and experiments

```
#+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)

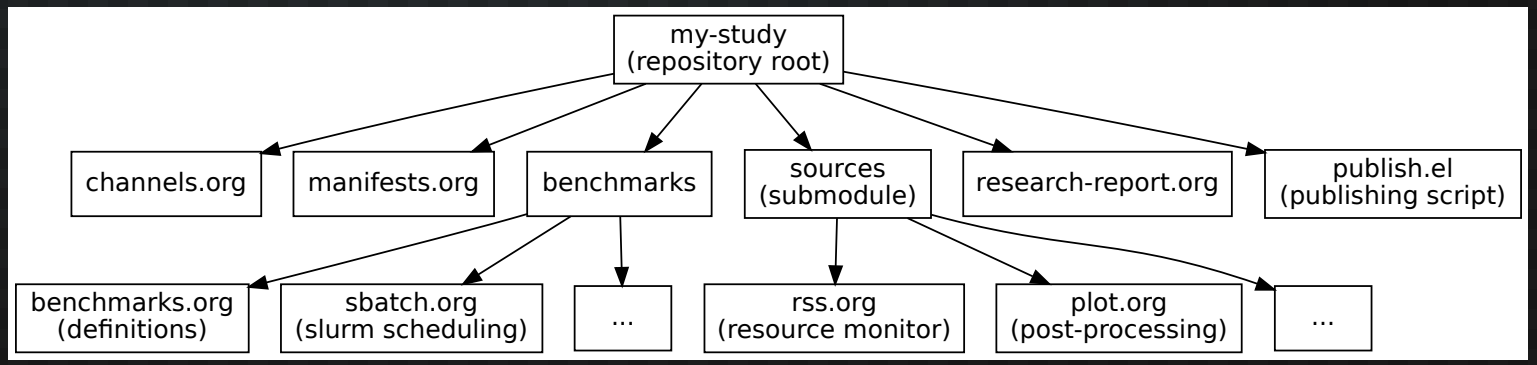
```
(define transform1
  (options->transformation
    '((with-input . "openblas=mkl")
      ;; ...
      (with-input . "slurm=slurm@19"))))
(packages->manifest
  (list (transform1 (specification->package "scab"))
        (transform1 (specification->package "gcvb-minimal-felsocim"))
        ;; ...
  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

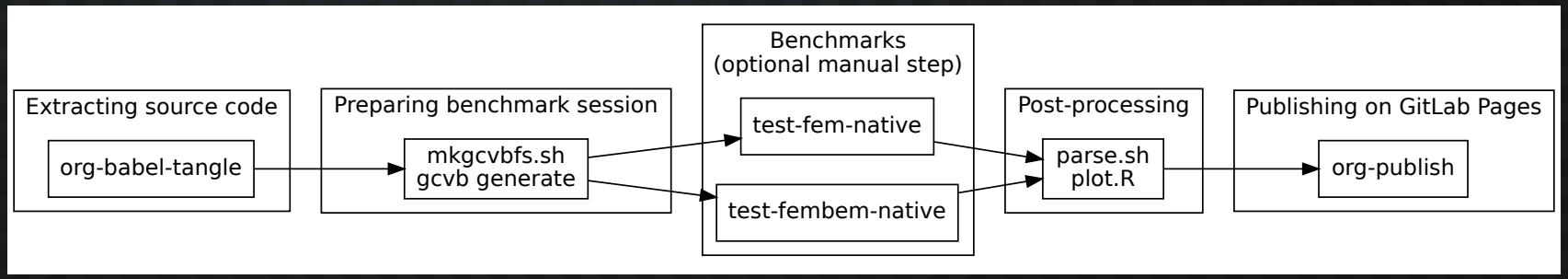
One study = one GitLab repository

- 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

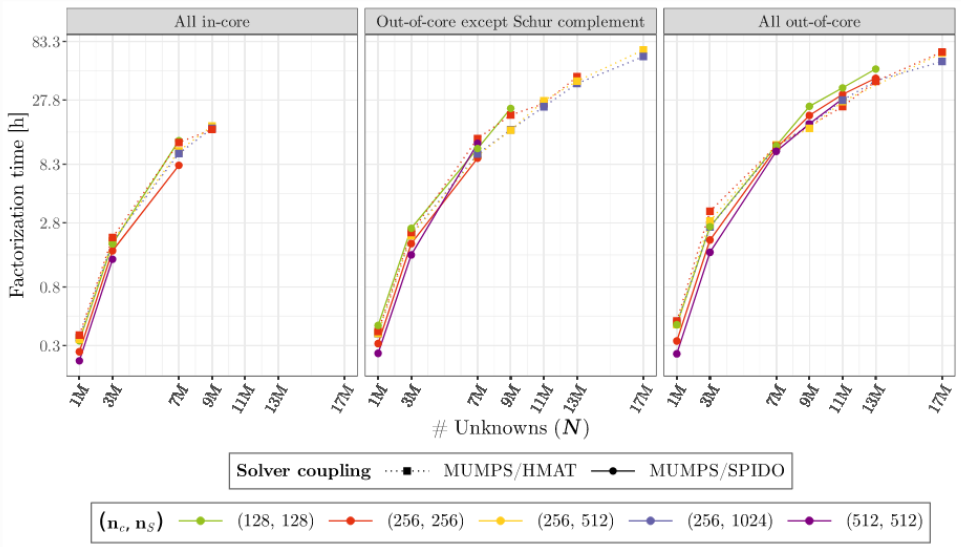


*Figure 7: Computation times of **multi-solve** on coupled FEM/BEM linear systems of varying number of unknowns for both of the solver couplings MUMPS/HMAT and MUMPS/SPIDO and for varying values of n_c and n_S. We test 3 different configurations of the out-of-core feature: completely disabled, enabled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

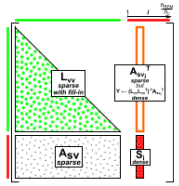


FIGURE 3: *baseline multi-solve.*

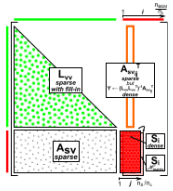


FIGURE 4: *compressed Schur multi-solve.*

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

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:
```

Summary

- 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

Personal experience

- share of my adventures with Guix and Org mode : `https://felsoci.sk/blog/posts.html`
- my own Guix channel (QtCreator IDE, svgfix utility and some attemps in the `unstable` branch) : `https://gitlab.inria.fr/mfelsoci/guix-extra`
- translation of Guix into my native Slovak language:

  preložené 100%

  `https://translate.fedoraproject.org/projects/guix/guix/sk/`

Further joining literate and reproducible studies?

```
:cache yes / guix / nix / gwl
```

Thank you!

- *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/.
- 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.