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

JCAD 2024

November 4-6, 2024, Bordeaux

Emmanuel Agullo
joint work with

Ludovic Courtès, Marek Felšöci, Gilles Marait, Florent Pruvost

Outline

Separation of concerns and HPC

Our quest (`cmake`, `spack`, and now `guix`)

Deployment on supercomputers

Conclusion

References

Separation of concerns (Soc) (Dijkstra, 1982)

Separation of concerns (Wikipedia)

In computer science, *separation of concerns* is a design principle for separating a computer program into distinct sections. Each *section* addresses a *separate concern*, a set of information that affects the code of a computer program.

Modularity (Wikipedia)

A program that embodies SoC well is called a *modular program*.

Separation of concerns (Soc) (Dijkstra, 1982)

Opportunities (Wikipedia)

- When concerns are well-separated, there are more opportunities for module *upgrade*, *reuse*, and *independent development*.
- Use third-party libraries?

HPC

- "Yes, but I want to have full control to ensure I deliver high performance"
- "Yes, but one more issue <for users> when deploying my software"

Grid (/ cloud) computing

- Even harder
- Concern tackled with even more care

`concace` **objectives**

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

A few codes

Currently

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

Four-years objective of `concace`

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

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

About bit-wise reproducibility

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.

Outline

Separation of concerns and HPC

Our quest (`cmake`, `spack`, and now `guix`)

Deployment on supercomputers

Conclusion

References

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 still useful)

Outline

Separation of concerns and HPC

Our quest (`cmake`, `spack`, and now `guix`)

Deployment on supercomputers

Conclusion

References

Method

`guix` is there

- The perfect experience: smooth transition from laptop to supercomputers

`guix` is not there (yet!)

- `guix pack`!
- We consider `singularity` in this presentation

Resources

- https://hpc.guix.info/
- https://guix-hpc.gitlabpages.inria.fr/guix-hpc-tutorial/

Plafrim ("PlaFRIM: Plateforme fédérative pour la recherche en informatique et mathématiques", n.d.)

Homogeneous experiments (`bora` nodes)

- 36 cores per node (two Intel Cascade Lake 6240 @ 2.6 GHz 18-cores processors)
- 192 GB RAM per node
- Omni-Path 100 Gb/s interconnect

Heterogeneous experiments (`sirocco14-16` nodes)

- 32 cores per node (two Intel Skylake 6142 @ 2.6 GHz 16-cores processors)
- 384 GB of memory per node
- 2 GPUs NVIDIA V100 (16GB) per node
- Omni-Path 100 Gb/s interconnect

Channels I

```
guix describe -f channels > guix-channels-acmrepro.scm
```

```
(list (channel
        (name 'guix)
        (url "https://git.savannah.gnu.org/git/guix.git")
        (branch "master")
        (commit
          "89a8d213292ab99a4af67d9767743f47d6a1dc3f")
        (introduction
          (make-channel-introduction
            "9edb3f66fd807b096b48283debdcddccfea34bad"
            (openpgp-fingerprint
              "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA")))))
```

Channels II

```
(channel
  (name 'guix-hpc-non-free)
  (url "https://gitlab.inria.fr/guix-hpc/guix-hpc-non-free.git")
  (branch "master")
  (commit
    "14c842c82c14d3e520ed115b301fb852b8aefab0"))
(channel
  (name 'guix-hpc)
  (url "https://gitlab.inria.fr/guix-hpc/guix-hpc.git")
  (branch "master")
  (commit
    "2a264f59a2f7bd408840d2a85484bac3eb546e14")))
```

Homogeneous set up

Manifest

```
guix shell --export-manifest chameleon maphys++ \
--with-input=mumps-openmpi=mumps-mkl-openmpi \
--with-input=openblas=mkl \
bash coreutils emacs gawk grep inetutils \
intel-mpi-benchmarks openmpi openssh \
sed slurm time vim which \
> guix-manifests-acmrepro.scm
```

Singularity set up (local machine)

```
SINGULARITY_ACMREPRO=`\
guix time-machine -C guix-channels-acmrepro.scm \
-- pack -f squashfs -m guix-manifests-acmrepro.scm \
-S /bin=bin --entry-point=/bin/bash`
cp $SINGULARITY_ACMREPRO acmrepro.gz.sif
```

Remote machine (supercomputer)

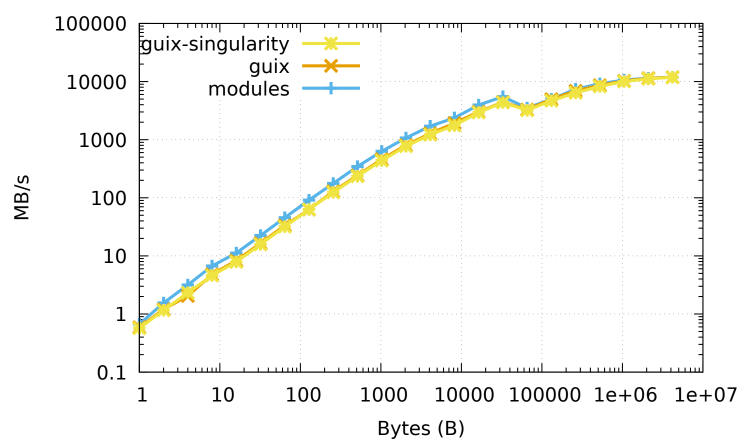
Host "vanilla" MPI

```
tar xJf $OMPI_TARBALL
cd openmpi-4.1.4/
OMPI_DIR=$PWD/install
./configure --with-slurm --prefix=$OMPI_DIR
make -j5 install
```

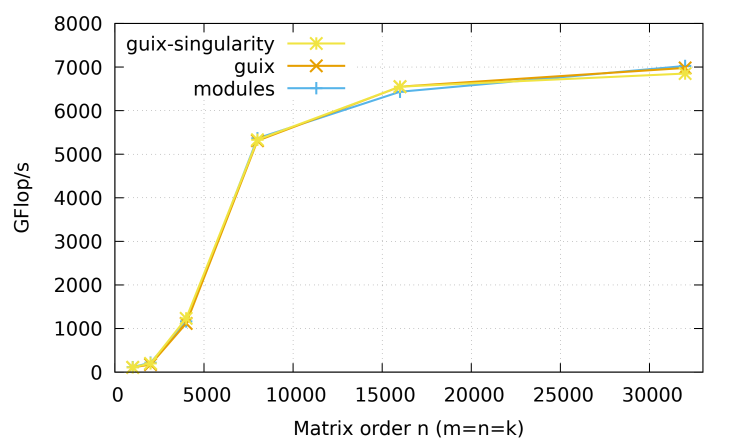
Run

```
$OMPI_DIR/bin/mpiexec singularity exec acmrepro.gz.sif IMB-MPI1 Pingpong
```

Intel-MPI-Benchmark PingPong - 2 nodes



`chameleon` homogeneous SGEMM - 2 nodes

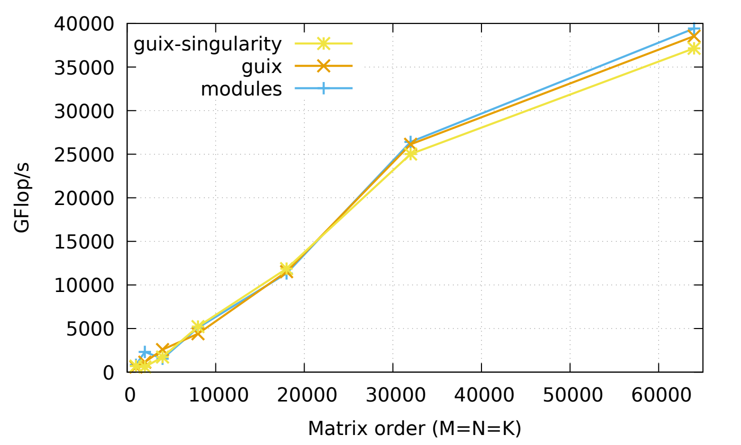


Heterogeneous set up

Manifest

```
guix shell --export-manifest chameleon-cuda \
--with-input=openblas=mkl \
bash coreutils emacs gawk grep inetutils \
intel-mpi-benchmarks openmpi openssh \
sed slurm time vim which \
> guix-manifests-acmrepro-cuda.scm
```

`chameleon` heterogeneous SGEMM - 2 nodes



Jean Zay ("Institut du développement et des ressources en informatique scientifique: calculateur Jean Zay", n.d.)

HPE SGI 8600 machine

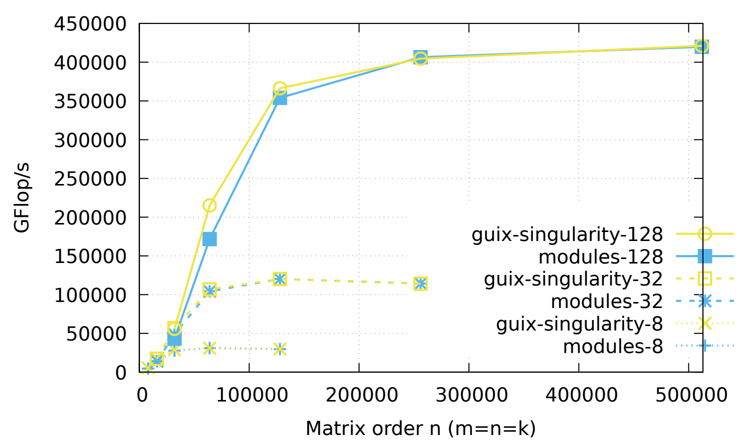
- 40 cores per node (two 20 cores Cascade Lake 6248 @ 2.5 GHz processors)
- 192 GB RAM per node
- Omni-Path 100 Gb/s interconnect

modules

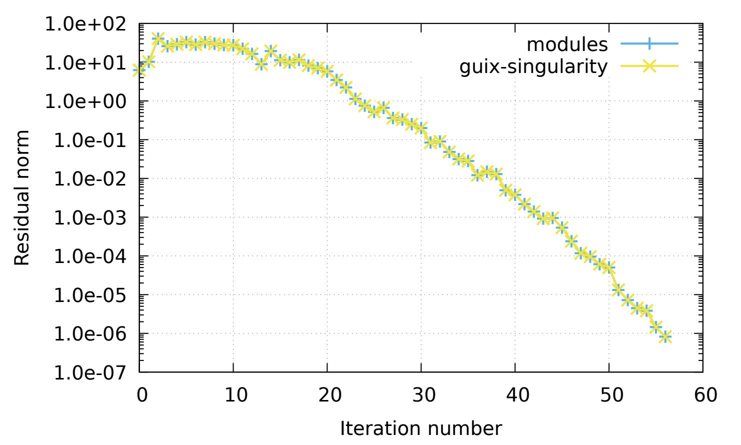
- Intel 2020.4 suite with `intel-compilers/19.1.3`,

`intel-mkl/2020.4` and `intel-mpi/2019.9`.

`chameleon` single-precision GEMM - 128 nodes



`composyx` CG convergence - 40 nodes - 40 subdomains



Neither `guix` nor `singularity` available

Local machine (laptop)

```
scp `guix pack -RR hwloc -S /bin=bin` supercomputer:hwloc.tar.gz
```

Remote machine (supercomputer)

```
mkdir -p ~/.local
(cd ~/.local; tar xf ~/hwloc.tar.gz)
~/.local/bin/lstopo
```

Outline

Separation of concerns and HPC

Our quest (`cmake`, `spack`, and now `guix`)

Deployment on supercomputers

Conclusion

References

Conclusion

Today (this presentation)

- We can *already* do *HPC* with `guix` today *on supercomputers*
  - `guix pack`
  - robustness
  - performance

Tomorrow (hope)

- `guix` also directly available on supercomputers
  - enhanced transition from laptop to supercomputers
  - more reliable deployment
  - composability
  - reproducibility

Thank you!

Thank you for your attention!

Outline

Separation of concerns and HPC

Our quest (`cmake`, `spack`, and now `guix`)

Deployment on supercomputers

Conclusion

References

References I

- Dijkstra, E. W. (1982).On the role of scientific thought. *Selected writings on computing: a personal perspective*, 60–66.
- Institut du développement et des ressources en informatique scientifique: calculateur Jean Zay. (n.d.).
- PlaFRIM: Plateforme fédérative pour la recherche en informatique et mathématiques. (n.d.).