Concilier le calcul haute performance avec l'utilisation de bibliothèques tierces ?

Emmanuel Agullo (Inria `hiepacs`)

Atelier reproductibilité des environnements logiciels - 17 et 18 mai 2021

Outline

Presentation agenda

hiepacs

hiepacs

- Conception d'algorithmes numériques
- Mise en oeuvre parallèle (MPI+threads+Cuda vs programmation à base de tâches)
- Essentiellement de l'algèbre linéaire et (ou multi-linéaire)
- Application à la simulation numérique et (plus récemment) à l'analyse de données

Quelques codes

- `chameleon`: solveur dense
- `pastix`: solveur creux direct
- `fabulous`: solveur itératif
- `maphys`: solveur hybride (direct + itératif)
- `scalfmm`: méthodes multipoles rapides

Interaction forte avec l'écosystème HPC d'Inria Bordeaux Sud Ouest

Supports d'exécution

- `starpu` (`storm`): moteur d'exécution pour machines hétérogènes ("*PU")
- `newmadeleine` (`tadaam`): moteur de communication (alternatif à openmpi ... et mpi)
- `hwloc` (`tadaam` et `storm`): découverte et exposition de la topologie

Partitionneur

- `scotch` (`tadaam`): partitionneur de graphes

Applications (exemple)

- `hou10ni` (magique3d): propagation d'ondes acoustiques

Presentation agenda

Définitions

Reproductibilité (*Larousse*)

Reproductibilité: caractère de ce qui peut être reproduit.

Environnements reproductibles (*reproducible-builds.org*)

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

Un sujet de discorde

Enthousiasme (*softwareheritage.org*)

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

Scepticisme (*Extrait de la liste calcul*)

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

Question typique pour une équipe comme hiepacs

Utilisation d'un grand nombre de bibliothèques tierces

- solveur hybride (*e.g.* maphys) utilisant un/des solveurs directs (*e.g.* pastix ou mumps) et itératifs (*e.g.* fabulous) robustes, optimisés et exploitant des supports d'exécutions performants (*e.g.* starpu et newmadeleine)
- ce solveur est lui-même embarqué dans une application (*e.g.* hou10ni)

Intérêts pour une équipe comme hiepacs (1/2)

Produire un environnement correct (!)

- Être en mesure tout simplement de *produire* un environnement logiciel aussi complexe en un temps acceptable !
- Travail fait (une fois) dans la définition des paquets plutôt que lors du déploiement.

Fiabilité du déploiement

- Faire en sorte que ça marche pour un utilisateur applicatif.
- Sur deux machines différentes ? En intégration continue ?
- Dans le temps sur la même machine ?
- Pour le pré-traitement (définition de campagnes expérimentales) et le post-traitement (figures, articles, site web, annexes) aussi ?

Intérêts pour une équipe comme hiepacs (2/2)

Développement collaboratif (*e.g.* starpu issue #4):

```
STARPU_FXT_TRACE=1 STARPU_FXT_PREFIX=/tmp/teststarpu guix
↪  environment --ad-hoc --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"
```

. . . et bien sûr faire de la science reproductible

Produire et reproduire une étude.

Discussion

Sensibilité numérique *vs* bit-reproducibilité?

- analyse d'erreurs,
- quantification d'incertitudes, ...

Compression *vs* bit-reproducibilité?

- compresseurs (`sz`, ...)
- représentation (matricielle, tensorielle) de rang faible (svd tronquée, ...)

Performance *vs* bit-reproducibilité?

- programmation à base de tâches
- parallélisme vs séquentiel
- architectures hétérogènes
- commutativité et associativité

Presentation agenda

Définition de maphys dans 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)
    # ...
```

Définition de maphys dans 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'),
        ])
# ...
```

Remarques sur cette définition spack

- Définition élégante et compacte de variantes (+mumps)
- Définition compacte de multiple versions (1.0, 0.9.8.3, 0.9.8.2, . . .)

Presentation agenda

Définition de maphys dans 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
                "0pcwfac2x574f6ggfdmahhx9v2hfswyd3nkf3bmc3cd3173312h
    (build-system cmake-build-system)
    ;; ...
```

Définition de maphys dans 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-environme
                                      (lambda _
                                        (setenv "OMPI_MCA_rmaps_base
```

Définition de maphys dans 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)))
))
```

Remarques sur cette définition guix

- Confiance sur le déploiement du paquet avec *toutes* ses dépendances ! (inimaginable par défaut – pour nous – sans un outil assurant une reproducibilité bit-à-bit de l'environnement)
- Importance de disposer de *variantes* (terminologie spack) / paquets paramétrés (terminologie guix) donc largement réduite (. . . mais non nulle).

Presentation agenda

Exemple org-mode/spack (PhD Louis Poirel) (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
```

Exemple org-mode/spack (PhD Louis Poirel) (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-pa
 ↪  ckages/
```

Toute la complexité de la gestion des dépendances embarquée dans la définition (*e.g.* maphys spack)

Remarques sur l'exemple `org-mode/spack`

Compacité, flexibilité et fiabilité

- Compacité: *vs* installer tous les composants un par un
- Flexibilité: modules spécifiques à la machine
- Fiabilité: modules fournis par les administrateurs (*e.g.* module `mpi` assurément inter-opérable avec le démon `slurm` de la plate-forme)

Limites ?

- Ajustement nécessaire:
  - par plate-forme
  - dans le temps (évolution des modules fournis)
- Garanties sur la compatibilité des modules (*e.g.* version de `parmetis`) fournis par les administrateurs avec la pile logicielle déployée par via `spack` (*e.g.* version de `mumps`)?

Exemple org-mode/guix

Cf. présentation de Marek Felsoci.

Presentation agenda

Conclusion

- Gain de productivité énorme
- Très grande confiance dans le déploiement sur les plate-formes où `guix` est supporté
- Un continuum du déploiement à la reproductibilité

Merci

- Merci aux développeurs et à la communauté `guix` ainsi qu'aux administrateurs qui l'ont déployé (en particulier en ce qui nous concerne à l'équipe `plafrim`) !
- Merci aux organisateurs de ces ateliers !

Presentation agenda

1. Capacité à tourner sur les principaux super-calculateurs

- `~/gnu/store` ? (pour les machines où les administrateurs n'ont pas encore franchi le pas)
- Preneur de retours consolidés sur l'utilisation d'images (`singularity`, `rr`) vs exécution native. `mpirun`, `slurm`, `singularity exec`, ...
- Structuration d'une communauté autour de `guix` pour nous tourner vers les centres de calcul de manière cohérente

2. Feuille de route pour une interaction sereine avec `slurmd`

État des lieux (à ma connaissance seulement)

- Difficulté d'appréhension autour de slurm, mpi et guix
- Revue des détails bas niveau (`mpi` behind the scene)
- Mise à disposition d'un paquet `openmpi` optimisé et portable

Questions (peut-être déjà résolues ?)

- Comment savoir si le client `slurm` déployé par `guix` est compatible avec le démon `slurmd` fourni par les administrateurs systèmes (*a priori* autrement que via `guix`)?
- Faut-il fournir toutes les versions des `slurm` ? (Problème du nombre de binaires à construire . . .)
- Se baser sur des outils tiers (pmix ?, . . .)?

3. Disponibilité des binaires (1/2)

Up-to-date vs réactivité

- Dans certains cas (très souvent je dirais pour les utilisateurs finaux), mieux vaut *largement* un `pull` en retard de deux jours mais dont la construction de ses paquets cibles est garantie et donc les binaires disponibles.
- Forte dépendance à l'Internet: quid de serveurs haute-disponibilité ?

4. Disponibilité des binaires (2/2)

Note sur spack

mirror

spack/etc/spack/defaults:

```
mirrors:
  spack-public:
  ↪  https://spack-llnl-mirror.s3-us-west-2.amazonaws.com/
```

build cache

The Extreme-scale Scientific Software Stack (E4S) project proposes a build cache, which can be enabled has follows:

```
spack mirror add E4S https://cache.e4s.io
wget https://oaciss.uoregon.edu/e4s/e4s.pub
spack gpg trust e4s.pub
```

4. Paquets paramétrés `guix` / variantes `spack`

- Rendre disponibles les "variantes" communément utilisées (*e.g.* `--with-input=openmpi=madmpi`), avec une combinatoire raisonnable à déterminer
- paquets paramétrés: une première version par paquet (le temps de consolider la réflexion sur l'expression de paramétrisation sur le DAG).

5. Homogénéisation

- `blas/lapack` en particulier

6. Reproducibilité à long terme

- Quelle est la politique d'intégration *guix/softwareheritage.org* dans un cadre multi-dépôts?

7. IDE reproductible

- Produire et reproduire est déjà une grande avancée.
- Permettre de rejouer et analyser dans le même environnement que les auteurs, ce serait un pas de plus très intéressant !

Merci pour tout ce travail nous fournissant un environnement stimulant !