SCons

Software Construction tool

Pierre Navaro

IRMA Strasbourg

2 décembre 2011

Présentation de Scons

- Utilitaire de construction logiciel utilisant python comme langage de base.
- Alternative solide aux utilitaires historiques make et autoconf.
- Permet de produire des programmes par transformation ou compilation.
- Assure une mise à jour correcte en analysant les dépendances.
- Très bonne rapidité
- Des valeurs par défaut sont disponibles en fonction de la plateforme afin de minimiser le travail du programmeur.
- Reconnait un très grand nombre de langages.
- Disponible sur Unix, Windows, Mac OS X...

Principes de base

- Le script à la racine du projet doit être nommé **SConstruct**. Il peut produire des fichiers ou appeler des scripts se trouvant dans des sous-répertoires.
- Langage déclaratif : Ne pas préjuger de l'ordre d'exécution.
- Exécution dans le répertoire où est SConstruct

```
scons
```

- Compilation parallèle

```
scons -j 2
```

- Nettoyage des fichiers internes

```
scons -c
```

- Exécution peu bavarde

```
scons -Q
```

Compiler plusieurs fichiers C :

- Les fichiers C sont les éléments d'une liste python

```
files = ['main.c', 'sub1.c','sub2.c' ]
files = Split("main.c sub1.c sub2.c")
files = Glob("*.c")
```

- Définition du constructeur pour générer le programme exécutable.

```
Program("exe",files)
```

```
$ scons
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
gcc -o sub1.o -c sub1.c
gcc -o sub2.o -c sub2.c
gcc -o main.o -c main.c
gcc -o exe main.o sub1.o sub2.o
scons: done building targets.
```

Compiler des bibliothèques

- Bibliothèque statique de fichiers F90

```
Library('user',['initial.f90','vitesse.f90'])
```

```
$ scons -Q
gfortran -o initial.o -c initial.f90
gfortran -o vitesse.o -c vitesse.f90
ar rc libuser.a initial.o vitesse.o
ranlib libuser.a
```

- Librairie dynamique C++

```
SharedLibrary('go++',Glob("*.cpp"),CPPATH='.')
```

```
$ scons -Q
g++ -o XFunction.os -c -fPIC -I. XFunction.cpp
g++ -o approx_grad.os -c -fPIC -I. approx_grad.cpp
g++ -o condlim.os -c -fPIC -I. condlim.cpp
g++ -o discret.os -c -fPIC -I. discret.cpp
g++ -o libgo++.dylib -dynamiclib  discret.os  condlim.os \
              XFunction.os approx_grad.os
```

Exemple d'un programme F90 lié avec des libraries

- SConstruct

```
Program('exe', Glob('*.f90'), F90FLAGS=['-O3'],
               LIBPATH='/usr/local/lib',
               LIBS=['hdf5', 'siloh5','z','gfortran'],
               F90PATH='/usr/local/include',
               LINK='g++',
               LINKFLAGS=['-Wl,-rpath','-Wl,/usr/local/lib'])
```

- Compilation

```
$ scons -Q
gfortran -o initial.o -c -O3 -I/usr/local/include initial.f90
gfortran -o sorties.o -c -O3 -I/usr/local/include sorties.f90
gfortran -o main.o -c -O3 -I/usr/local/include main.f90
gfortran -o vitesse.o -c -O3 -I/usr/local/include vitesse.f90
g++ -o exe -Wl,-rpath -Wl,/usr/local/lib \
    initial.o main.o sorties.o vitesse.o \
   -L/usr/local/lib  -lhdf5 -lsiloh5 -lz -lgfortran
```

Notion d'environnement

Un environnement est une collection de variables définissant les mécanismes de compilation et d'édition de liens.

- Construire un nouvel environnement.

```
import os
env = Environment(ENV = os.environ)
```

- Interrogation d'un environnement.

```
print "Compilateur C = ", env['CC']
print env.Dump() # dump tous les paramètres
```

- Utiliser un environnement pour compiler

```
env.Library ('user', Split ('user1.f90 user2.f90'))
env.Program ( 'exe' ,  'main.f90', LIBS = [ 'user' ] , LIBPATH = [ '.' ] )
```

- Modification d'un environnement.

```
debug = Environment(F90FLAGS=['-g'])
opt = debug.clone()
opt.Replace(F90FLAGS=['-O3'])  #pour une compilation optimisée
debug.Append(F90FLAGS=['-pg']) #pour analyser avec gprof
```

Exemple d'un programme C++ avec des librairies

```
SRCSGOPP = Split("""
        geom.cpp maillage.cpp discret.cpp ish.cpp condlim.cpp
        interpolation.cpp XFunction.cpp approx_grad.cpp hamiltonian.cpp
        application.cpp timesolveur.cpp sortie_resMTV.cpp sortie_resGNUPLOT.cpp
""")
SRCSHYPECOLPP = Split("""
        geomNS.cpp interp2D.cpp metric2D.cpp maillageNS.cpp
        FunctionMinimisation.cpp Optimisation.cpp MeshSmoothingNS.cpp discretNS.cp
        state2D.cpp thermo_equil2D.cpp thermo_hors_equil2D.cpp
        condlim2D.cpp state_interface2D.cpp state_node2D.cpp Gradients.cpp riemann
        flux2D_hyperbolique.cpp  flux2D_eulerien.cpp flux2D_lagrangien.cpp
        InterfaceReconstructionNS.cpp bilan_flux2D.cpp projection2D.cpp
        applicationNS.cpp timesolveurNS.cpp DiscontinuousALENS.cpp gpc.c
""")
SRCSDIFFUSION= 'DiffusionBuildGlobalMatrix.cpp'
APPLI= 'goppCEMRACS_DIF.cpp'

env = Environment(LIBPATH='.',CPPPATH='.',CPPFLAGS='-O3')
env.Library('go++',SRCSGOPP)
env.Library('hycl++',SRCSHYPECOLPP)
env.Library('diff++', SRCSDIFFUSION)
env.Program(APPLI, LIBS=['go++', 'hycl++', 'diff++'])
```

Production hiérarchique

Les fichiers de la bibliothèque sont dans src et les programmes sont dans le répertoire test.

- fichier Sconstruct

```
import os
env = Environment( ENV = os.environ, CPPDEFINES='G95')
SConscript(['src/SConscript'],['env'])
SConscript(['test/SConscript'],['env'])
```

- SConscript dans les répertoires src et test

```
Import('env')
env.Library('fish90',['fish.F']+Glob('*.f'))

Import('env')
env.Program('tpois3d.exe',['tpois3d.f'],LIBPATH=['../src'],LIBS=['fish90'])
```

```
$scons
ar rc src/libfish90.a src/fish.o src/blktri.o src/cblktri.o ...
ranlib src/libfish90.a
gfortran -o test/tpois3d.o -c -DG95 test/tpois3d.f
gfortran -o test/tpois3d.exe test/tpois3d.o -Lsrc -lfish90
```

Séparation sources et compilation

- fichier Sconstruct pour compiler dans lib

```
import os
env = Environment( ENV = os.environ, CPPDEFINES='G95')
SConscript(['src/SConscript'],['env'],variant_dir='lib')
```

- Autre possibilité, dans le fichier SConstruct

```
import os
env = Environment( ENV = os.environ, CPPDEFINES='G95')
VariantDir('lib','src')
SConscript(['lib/SConscript'],['env'])
VariantDir('bin','test')
env.Program('tpois3d.exe',['bin/tpois3d.f'],LIBPATH=['lib'],LIBS=['fish90'])
```

Attention SCons copie src dans lib $\rightarrow$ risque de fichiers écrasés.

```
VariantDir('bin','test',duplicate=0)  #pas de copie du repertoire.
```

des soucis avec les fichiers d'entête peuvent apparaitre dans ce cas.

Configuration et vérification de dépendances

```
import os
env=Environment(ENV=os.environ,F90FLAGS=['-O3'],
 LIBPATH='/usr/local/lib', FRAMEWORKS='veclib',F90PATH='/usr/local/include',
 LIBS=['hdf5', 'siloh5', 'z','gfortran'],LINK='g++',
 LINKFLAGS=['-Wl,-rpath','-Wl,/usr/local/lib'])
conf = Configure(env)
if not conf.CheckLib('hdf5'):
   print "HDF5 doit etre installe"
   Exit(1)
if not conf.CheckLibWithHeader('siloh5','silo.h','c'):
   print "SILO doit etre installe"
   Exit(1)
env = conf.Finish()
env.Program('exe', Glob('*.f90'))
```

```
$ scons
scons: Reading SConscript files ...
Checking for C library hdf5... (cached) yes
Checking for C library siloh5... (cached) yes
scons: done reading SConscript files.
scons: Building targets ...
```

Installation dans des répertoires différents

On veut installer la bibliothèque dans le répertoire lib et l'exécutable dans le répertoire bin.

```
import os
env = Environment( ENV = os.environ, CPPDEFINES='G95')
fish_lib = env.Library('fish90',['src/fish.F']+Glob('src/*.f'))
env.Install('lib',fish_lib)
prog_test = env.Program('tpois3d.exe',['test/tpois3d.f'],
                        LIBPATH=['lib'],LIBS=['fish90'])
env.Install('bin',prog_test)
env.Depends('prog_test','fish_lib') #facultatif
```

```
$ scons
ar rc libfish90.a src/fish.o src/blktri.o src/cblktri.o src/cmgnbn.o ...
ranlib libfish90.a
Install file: "libfish90.a" as "lib/libfish90.a"
gfortran -o tpois3d.exe test/tpois3d.o -Llib -lfish90
Install file: "tpois3d.exe" as "bin/tpois3d.exe"
```

Options en ligne de commande

- SetOption

```
import os
num_cpu = int(os.environ.get('NUM_CPU', 2))
SetOption('num_jobs', num_cpu)
print "running with -j", GetOption('num_jobs')
```

- GetOption

```
if GetOption( 'help'):
    print 'Voici la liste des options possibles'
```

```
$ export NUM_CPU=4; scons -h
Voici la liste des options possibles
running with -j 4
usage: scons [OPTION] [TARGET] ...
```

Variables en lignes

- Scons stocke ces variables définies en ligne de commande dans un dictionnaire ARGUMENTS.

```
import os
env = Environment(ENV = os.environ)
fcompiler = ARGUMENTS.get('fcompiler', 'gfortran')
env.Replace(FORTRAN = fcompiler)
```

```
$scons fcompiler=sunf95
```

- On peut aussi utiliser la liste de nuplets ARGLIST.

```
cppdefines = []
for key, value in ARGLIST:
    if key == 'define':
        cppdefines.append(value)
env = Environment(CPPDEFINES = cppdefines)
env.Object('fish.F')
```

```
$ scons -Q define=G95
gfortran -o fish.o -c -DG95 fish.F
```

Malheureusement il n'y a pas d'aide disponible pour l'utilisateur.

Variables en lignes (suite)

- Pour obtenir l'aide en ligne on utilise le constructeur Variables

```
vars = Variables()
vars.Add('HAVE_FFTW', 'Set to 1 to build with FFTW library', 0)
env = Environment(variables = vars,
                  CPPDEFINES={'HAVE_FFTW' : '${HAVE_FFTW}'})
Help(vars.GenerateHelpText(env))
```

```
$scons -Q HAVE_FFTW=1
cc -o fft.c -c -DHAVE_FFTW=1 fft.c
$scons -Q -h
HAVE_FFTW: Set to 1 to build with FFTW library
    default: 0
    actual: None
```

- Dans un fichier :

```
vars = Variables ('custom.py')
```

```
$cat custom.py
HAVE_FFTW=1
```

Créer une règle de compilation

- Ecrire un "Builder".

```
import os
env = Environment(ENV=os.environ)
env.PrependENVPath('PATH','/usr/local/cuda/bin')
cudaBuilder = Builder(action= 'nvcc -m64 -c $SOURCE',src_suffix = '.cu')
env.Append(BUILDERS = { 'Cuda' : cudaBuilder})
env.Cuda('c_kernels')
```

```
$ scons -Q
nvcc -m64 -c c_kernels.cu
```

- Ne pas écrire un "Builder" et utiliser "Command".

```
f2py_module = env.Command('mylapack','dgemm.f',
   "f2py -m $TARGET -c $SOURCE --f90flags=-O3 -lapack" )
Depends(f2py_module, 'dgemm.pyf')

lib_user = env.Library('user',['f90routines.f90','f90wrap.f90'])
f2py_module = env.Command('f90mod','f90module.f90',
   "f2py -m $TARGET -c $SOURCE --f90flags=-O3 -luser" )
Requires(f2py_module, lib_user)
```

Un exemple de SConstruct pour PETSc

```
import os
env = Environment(ENV=os.environ)

PETSC_DIR  = os.getenv('PETSC_DIR')
PETSC_ARCH = os.getenv('PETSC_ARCH')

env.Replace(F90=PETSC_DIR+'/'+PETSC_ARCH+'/bin/mpif90')
env.Replace(F90FLAGS=['-O3','-m64'])
env.Append(LIBPATH=[ PETSC_DIR+'/'+PETSC_ARCH+'/lib'])
env.Append(LIBS=['petsc'])
env.Append(LINKFLAGS=['-Wl,-commons,use_dylibs'])
env.Append(F90PATH=[ PETSC_DIR+'include/finclude',
                     PETSC_DIR+'/'+PETSC_ARCH+'/include',
                     PETSC_DIR+'/include'])

env.Program('exe',  ['petsc_dmmg.F90'])
```

Gérer plusieurs serveurs connus avec SCons

```
import os, socket,distutils.util

platform = distutils.util.get_platform()
hostname = socket.gethostname()

env = Environment(ENV=os.environ,LIBS=['siloh5', 'z', 'hdf5'])

if platform[:6] == 'macosx':
        env.Replace(CPPDEFINES = 'MAC')
        env.Replace(FRAMEWORKS  = 'veclib',
                    LINK = 'g++',
                    LIBPATH= '/usr/local/lib',
                    F90PATH = ['/usr/local/include'],
                    F90FLAGS = '-cpp -D_MAC',)
        env.Append(LIBS = ['gfortran', 'dfftpack'])

if hostname[:8] == 'irma-hpc': #where Oracle studio compiler is available
        env.Replace(FORTRAN='sunf95')
        env.Prepend(FORTRANFLAGS = '-fpp')
        env.Replace(LIBPATH= '/usr/local/hdf5/lib',LINKFLAGS='-xlic_lib=sunperf')
        env.Append(LIBS = ['nsl', 'socket'])

env.Program('picsou', Glob('*.f90')+Glob('*.f'))
```

Un dernier exemple

Vérifier un numéro de version gfortran :

```
ff = os.popen('gfortran --version')
    ll = ff.readline()
    ff.close()
    fc_version =ll[18:21]
    if float(fc_version) < 4.5:
       print 'gfortran version number '+fc_version+' less than 4.5'
       exit();
```

Lancer des tests

```
def runUnitTest(env,target,source):
   import subprocess
   app = str(source[0].abspath)
   if not subprocess.call(app):
      open(str(target[0]),'w').write("PASSED\n")

for root,dirs,files in os.walk('.'):
    for file in files:
        env.Program(file)
        Command("unit_test in "+root[2:],file[:-4],runUnitTest)
```

Autres fonctionnalités de Scons

- Changer le type de détection pour la mise à jour. Par défaut, elle est basée sur la signature MD5.
- Gestion explicite des dépendances (Depend,Requires,Ignore)
- Ecrire ses propres constructeurs (CUDA).
- Très efficace pour les programmes JAVA.
- On peut utiliser des fonctions python pour la construction.
- Faiblesses
  - Produit jeune.
  - Quelques manques en Fortran
  - distutils reste plus performant pour créer des modules python.
  - Réputé plus lent que CMake, l'utilisation de MD5 est chère.
- Documentation et exemples
  - `http://www.scons.org/doc/2.0.1/HTML/scons-user.html`
  - `http://www.scons.org/wiki/SconsRecipes`