Compiling and using Python on BlueGene/Q

Bertrand Brelier

SOSCIP

May 28, 2014

What is Soscip

SOSCIP
SOUTHERN ONTARIO SMART
COMPUTING INNOVATION PLATFORM

- Research collaboration involving :
  - Seven southern Ontario universities
  - IBM Canada
  - Dozens of small and medium-sized enterprises in Ontario
- Provide HPC access for simulation / data mining using :
  - An advanced cloud and analytic platform
  - An agile computing environment (FPGAs)
  - IBM Blue Gene/Q

Shared Libraries on BlueGene/Q

- Blue Gene/Q offers the possibility to create shared libraries and dynamic executables.
- Shared Libraries in C :
  - Compiler option "-qpic" to generate position independent code.
  - Use the options "-qmkshrobj" and "-qnostaticlink" to generate the shared library
  - example :
    mpixlc -c -qpic library_function.cmpixlc -qmkshrobj -qnostaticlink -o libmys_c.so library_function.o
- Dynamic Executables in C :
  - compile the code with -qpic
  - link it with the shared libraries using the "-qnostaticlink -qnostaticlink=libgcc" options.
  - Use the linker option "-rpath" to specify the directory where the shared libraries or objects will be located during execution of the code. (can set LD_LIBRARY_PATH)
  - example :
    mpixlc -c -qpic dyn_c.cmpixlc -o dyn_c.x dyn_c.o -qpic -Wl,-rpath=path-to-shared-lib -Lpath-to-shared-lib -lmys_c -qnostaticlink -qnostaticlink=libgcc

Compiling Python on BlueGene/Q

- By using the scripts provided by :
  /bgsys/drivers/V1R2M0/ppc64/tools/python/genbldsrc.sh
  /bgsys/drivers/V1R2M0/ppc64/tools/python/cfgandbuild.sh
- Version : 2.6.7, 2.6.8, 2.7.3, 3.2.2, or 3.2.3
- Can be built with GNU or XL compilers
- With the GNU compiler powerpc64-bgq-linux-gcc
  - Apply the patch:
    ./genbldsrc.sh --version 2.7.3 (--blddir)
  - Build and install :
    ./cfgandbuild.sh --version 2.7.3
- With the XL compiler bgxlc_r
  - Apply the patch:
    ./genbldsrc.sh --version 2.7.3 --compiler XL
  - Build and install :
    ./cfgandbuild.sh --version 2.7.3 --compiler XL

Compiling Python on BlueGene/Q

- works also with non-default version of a compiler :
- example with gcc 4.8.1 :
- replace /bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc with the location of powerpc64-bgq-linux-gcc 4.8.1 in file cfgandbuild.sh

Running python on BlueGene/Q

- To run python on BG/Q, the full path of python, the path to the library, the home directory have to be provided :

-
  ```
  runjob --np 1 --ranks-per-node=1 : python Test.py
  could not start job: job failed to start
  Opening application executable failed, errno 2 No such file or directory
  ```

-
  ```
  runjob --np 1 --ranks-per-node=1 :
  /scinet/bgq/tools/Python/python2.7.3-20131205/bin/python Test.py
  error while loading shared libraries: libpython2.7.so.1.0: cannot open shared object
  file: No such file or directory
  ```

-
  ```
  export LD_LIBRARY_PATH=/scinet/bgq/tools/Python/python2.7.3-20131205/lib
  runjob --np 1 --ranks-per-node=1 --envs LD_LIBRARY_PATH=$LD_LIBRARY_PATH :
  /scinet/bgq/tools/Python/python2.7.3-20131205/bin/python Test.py
  KeyError: 'getpwuid(): uid not found: 11062'
  ```

-
  ```
  runjob --np 1 --ranks-per-node=1 --envs HOME=$HOME LD_LIBRARY_PATH=$LD_LIBRARY_PATH :
  /scinet/bgq/tools/Python/python2.7.3-20131205/bin/python Test.py
  2.7.3
  ```

mpi4py

- Using mpich2/gcc (module compiled with /bgsys/drivers/ppcfloor/comm/gcc/bin/mpicc)
  - `python setup.py build`
  - `python setup.py install --prefix=...`
- Test.py:

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
print ‘‘Process number ‘‘+str(rank)+’’/’’+str(size)
```

- `runjob --np 4 --ranks-per-node=1 --envs HOME=$HOME LD_LIBRARY_PATH=$LD_LIBRARY_PATH :`

```
/scinet/bgq/tools/Python/python2.7.3-20131205/bin/python Test.py
Process number 2/4
Process number 1/4
Process number 0/4
Process number 3/4
```

Numpy with essl and lapack

- Copy site.cfg.example to site.cfg then add:

```
[DEFAULT]
library_dirs = /opt/ibmmath/essl/5.1/lib64
include_dirs = /opt/ibmmath/essl/5.1/include
libraries = lesslsmpbg, mass_simd, xl, xlopt, xlf90_r, xlfmath, xlsmp
search_static_first = true
```

- set variables :

```
export ldpath=/scinet/bgq/compilers/gcc-4.8.1/lib64/
export ldflags=''-Wl,--allow-multiple-definition -L/opt/ibmcmp/xlmass/bg/7.3/bglib64/
-L$SCINET_LAPACK_LIB''
export p=/scinet/bgq/tools/Python/python2.7.3-20131205/bin/python
c=''/scinet/bgq/compilers/gcc-4.8.1/bin/powerpc64-bgq-linux-gcc -DNO_APPEND_FORTRAN
-L/opt/ibmcmp/xlmass/bg/7.3/bglib64/ -L$SCINET_LAPACK_LIB''
```

- Now build and install numpy :

```
MATHLIB=''mass_simd'' LDFLAGS=''$ldflags -L/opt/ibmmath/essl/5.1/lib64/ -lesslsmpbg
-L/scinet/bgq/Libraries/lapack/lib/ -llapack -lrefblas
-L/opt/ibmcmp/xlf/bg/14.1/bglib64/ -lxlf90_r -lxlopt -lxl
-L/opt/ibmcmp/lib64/bg/bglib64/ -lxlsmp
-L/bgsys/drivers/ppcfloor/gnu-linux/powerpc64-bgq-linux/lib/ -lc -llapack''
LD_LIBRARY_PATH=''$ldpath:$LD_LIBRARY_PATH'' CC=''$c'' $p setup.py install
```

Numpy with the shared libraries

- Compiled with the shared library. When run :

```
>>> import numpy
File ''.../site-packages/numpy/linalg/linalg.py'', line 29, in <module>
from numpy.linalg import lapack_lite, _umath_linalg
ImportError:  /opt/ibmmath/essl/5.1/lib64/libesslsmpbg.so.1:  undefined symbol:
__xlf_omp_get_nested_i8
```

- The essl libraries were obviously compiled with xlf, xlf libraries needed during compilation
- I am wondering how well we can combine gcc and xlf modules, especially when these use OpenMP. We can do pure MPI instead but I do not know what is the best for numpy.

Scipy with essl and lapack

```
export BLAS=$SCINET_ESSL_LIB/libesslsmpbg.a
export LAPACK=$SCINET_LAPACK_BASE
export F90=/scinet/bgq/compilers/gcc-4.8.1/bin/powerpc64-bgq-linux-gfortran-4.8.1
export F77=/scinet/bgq/compilers/gcc-4.8.1/bin/powerpc64-bgq-linux-gfortran-4.8.1
/scinet/bgq/tools/Python/python2.7.3-20131205/bin/python setup.py build
/scinet/bgq/tools/Python/python2.7.3-20131205/bin/python setup.py install
```

Scipy with gfortran

- I had the following problem with Scipy with gfortran :

```
File ``.../site-packages/scipy/sparse/linalg/isolve/iterative.py'', line 7, in
<module>
from .  import _iterative
ImportError:  .../site-packages/scipy/sparse/linalg/isolve/_iterative.so:  undefined
symbol:  slamch_
```

- Only slamch is defined in our compiled lapack library :

```
nm -g $SCINET_LAPACK_LIB/liblapack.a |grep slamch |grep D
0000000000000000 D slamch
```

- Most fortran compilers add an '_' to the names of routines and variables
- The XL compilers do not do this, and since our lapack was compiled with bgxlf then it would not have those symbols.

h5py

- The h5py package is a Pythonic interface to the HDF5 binary data format.
- The module is installed with :
```
python setup.py build
python setup.py install --prefix=...
```
- When I tried to test it, I had the following error :
```
h5py/_errors.so:  undefined symbol:  SZ_encoder_enabled
```
- Typically this error means that h5py was mistakenly compiled against a "static" version of the HDF5 library. For h5py to work correctly, HDF5 must be installed as a dynamic library.

Compiling hdf5 with the dynamic libraries

- I have tried to compile hdf5 with cmake but I have the following problem :

```
CMake Warning at CMakeFilters.cmake:46 (FIND_PACKAGE):
Could not find a package configuration file provided by ''ZLIB'' with any of the
following names:
ZLIBConfig.cmake
zlib-config.cmake
```

- Installation using configure, make, make install
- during the configure :

```
as: symbol lookup error: /bgsys/drivers/V1R2M0/ppc64/comm/sys/lib/libpami.so:
undefined symbol: _cnkspi_MemoryRegionCacheLastAccessedElementNumber
```

  The problem came from the mpi compiler used to compile the zlib (should have used the serial gcc compiler)

- configure with :

```
CC=mpicc ./configure --prefix=...  --with-zlib=.../zlib-1.2.7-gcc4.8.1/
--with-szlib=.../szip-2.1-gcc4.8.1/ --enable-shared --enable-parallel
```

h5py with the hdf5 shared libraries

- Once the hdf5 shared libraries have been compiled :
```
export CPPFLAGS=''-L$SCINET_HDF5_LIB''
python setup.py build --mpi
python setup.py install --prefix=...
```
- The parallel features of HDF5 are mostly transparent. To open a file shared across multiple processes, use the mpio file driver. Example program which opens a file, creates a single dataset and fills it with the process ID:
```
from mpi4py import MPI
import h5py
rank = MPI.COMM_WORLD.rank
f = h5py.File('parallel_test.hdf5', 'w', driver='mpio', comm=MPI.COMM_WORLD)
dset = f.create_dataset('test', (4,), dtype='i')
dset[rank] = rank
f.close()
```
- `runjob --np 4 --ranks-per-node=1 --envs HOME=$HOME LD_LIBRARY_PATH=$LD_LIBRARY_PATH : /scinet/bgq/tools/Python/python2.7.3-20131205/bin/python Test.py`
- `h5dump parallel_test.hdf5`
```
HDF5 ''parallel_test.hdf5'' {
GROUP ''/'' {
   DATASET ''test'' {
      DATATYPE H5T_STD_I32BE
      DATASPACE SIMPLE { ( 4 ) / ( 4 ) }
      DATA {
      (0):  0, 1, 2, 3
      }
```

Problem with variable length string

One of the astrophysics team working on our BlueGene noticed that they were not able to read variable length string from their hdf5 files.
The same python script was working fine on a x86 cluster.
The HDF5 library is big-/little-endian transparent.
I wrote 2 scripts to write and read an hdf5 file with variable length strings:

```
import h5py
file = h5py.File('vlstring_BGQ.h5','w')
str_type = h5py.new_vlen(str)
dataset = file.create_dataset(''DSvariable'',(4000,), dtype=str_type)
data=()
for i in range(1000):
data += (''Parting'', '' is such'', '' sweet'', '' sorrow...'')
dataset[...]  = data
file.close()
```

and read the file :

```
import h5py
file = h5py.File('vlstring_BGQ.h5', 'r')
dataset = file['DSvariable']
data_out = dataset[...]
for i in range(4):
print ''DSvariable[''',i,'']'', '''''+data_out[i]+'", ''has length'', len(data_out[i])
print data_out

file.close()
```

Problem with variable length string

- Submitted the job on BlueGene : no issue
- did the same on a x86 cluster : no issue

File read / File produced	BlueGene	x86 cluster
BlueGene	No problem	Failed to find converter for 8 -> PYTHON:OBJECT
x86 cluster	Failed to find converter for 8 -> PYTHON:OBJECT	No problem

- Both files have the same size but differ from 3 to 4 bytes.
- No such problem with C code, contacted the h5py developers : https://github.com/h5py/h5py/issues/428
  Possible byte order glitch related to opaque types, still under investigation.

Cython

- helloworld.pyx :

```
print ''Hello World''
```

- setup.py :

```
from distutils.core import setup
from Cython.Build import cythonize
setup(
   ext_modules = cythonize(''helloworld.pyx'')
)
```

- compile with :

```
python setup.py build_ext --inplace
```

```
powerpc64-bgq-linux/bin/ld:  cannot find -lpython2.7
```

- compile with :

```
python setup.py build_ext --inplace -L$SCINET_PYTHON_LIB
running build_ext
building 'helloworld' extension
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -pthread
-fno-strict-aliasing -g -O2 -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC
-I/scinet/bgq/tools/Python/python2.7.3-20131205/include/python2.7 -c helloworld.c -o
build/temp.linux-ppc64-2.7/helloworld.o
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -pthread -shared
build/temp.linux-ppc64-2.7/helloworld.o
-L/scinet/bgq/tools/Python/python2.7.3-20131205//lib/ -L. -lpython2.7 -o helloworld.so
```

Cython

- Test.py :

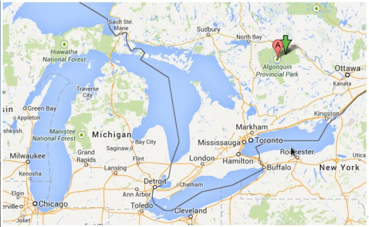
  ```
  import helloworld
  ```

- `runjob --np 4 --ranks-per-node=1 --envs HOME=$HOME LD_LIBRARY_PATH=$LD_LIBRARY_PATH :`

  ```
  /scinet/bgq/tools/Python/python2.7.3-20131205/bin/python2.7 Test.py :
  Hello World
  Hello World
  Hello World
  Hello World
  ```

Very-long-baseline interferometry (VLBI)

- astronomical interferometry used in radio astronomy.
- Signal from an astronomical radio source collected at multiple radio telescopes on Earth.
- Distance between the radio telescopes is calculated using the time difference between the arrivals of the radio signal at different telescopes.
- Observations of an object made simultaneously by many radio telescopes to be combined, emulating a telescope with a size equal to the maximum separation between the telescopes.
- Pr. Ue-Li Pen team (University of Toronto) :
  Correlate pulsar signals from ARO (Algonquin Radio Observatory), CHIME (Penticton, BC), LOFAR (Netherlands), GMRT (India)



Very-long-baseline interferometry

Goal : combines multiple telescopes around the world to achieve milli arc second or better resolution.

Code reads in a time series of data sampled at a rate $\sim 10^6$, and process fft (Fast Fourier transform)
Observation from 3 telescopes which need to x-correlate: would take $\sim$ 648000 seconds of data to process, or roughly 180 hours of single-core processing.

- Thought processing might be disk-read limited since each chunk is $\sim$ Mb, and is processing a chunk takes $\sim$ 1s. But with 512 processes, managed to read $\sim$ 512 Mb/s.
- python for control flow, but the “hard” processing is shipped to fortran code, fftw through the pyfftw wrapper.

Very-long-baseline interferometry

- only run 4 mpi processes per node before hit memory issues. 4 openmp threads per process (in the fftw calls)
- data files opened with another python module, mpi4py.MPI
- hdf5 to store data products.

scalability :

Number of node boards	Number of MPI processes	Number of MPI per node	Number Thread per MPI	Time (in s)
64	32	4	4	6877.5
64	64	4	4	3472.2
64	128	4	4	1784.0
64	256	4	4	979.0
128	512	4	4	563.4

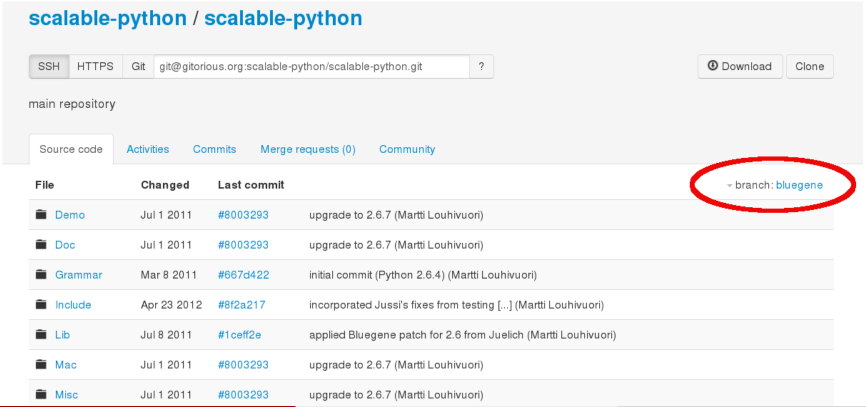
GPAW

GPAW is a density-functional theory (DFT) Python code based on the projector-augmented wave (PAW) method and the atomic simulation environment (ASE).
followed the instructions from :
https://wiki.fysik.dtu.dk/gpaw/install/BGQ/mira.html
Dependencies :

- Scalable Python, NumPy, ScaLAPACK, HDF5

scalable-python

https://gitorious.org/scalable-python :
Scalable Python is a modified version of Python geared towards high-performance computing (HPC) on massively parallel supercomputers. It tries to address major performance bottlenecks and scalability issues that limit Pythons applicability to HPC with as minimal modifications as possible to the Python code base.
There is a BlueGene branch in the repository :


scalable-python / scalable-python

File	Changed	Last commit	
Demo	Jul 1 2011	#8003293	upgrade to 2.6.7 (Martti Louhivuori)
Doc	Jul 1 2011	#8003293	upgrade to 2.6.7 (Martti Louhivuori)
Grammar	Mar 8 2011	#667d422	initial commit (Python 2.6.4) (Martti Louhivuori)
Include	Apr 23 2012	#8f2a217	incorporated Jussi's fixes from testing [...] (Martti Louhivuori)
Lib	Jul 8 2011	#1ceff2e	applied Bluegene patch for 2.6 from Juelich (Martti Louhivuori)
Mac	Jul 1 2011	#8003293	upgrade to 2.6.7 (Martti Louhivuori)
Misc	Jul 1 2011	#8003293	upgrade to 2.6.7 (Martti Louhivuori)

scalable-python

- To install scalable-python :

```
export CC=/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc
export CXX=/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-g++
export MPICC=mpicc
export CCSHARED=-fPIC
export LINKFORSHARED='-Xlinker -export-dynamic -dynamic'
export MPI_LDFLAGS_SHARED='-Xlinker -export-dynamic -dynamic'
./configure --prefix=...  --enable-mpi --disable-ipv6
make
make mpi
make install
make install-mpi
```

- NumPy 1.3.0 or later is recommended.A compiler must be explicitly specified.

```
export CC=/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc
export BASECFLAGS=''-fno-strict-aliasing''
export LD_LIBRARY_PATH=/bgsys/drivers/ppcfloor/gnu-linux/lib64
export PYTHONHOME=...
export PYTHON=$PYTHONHOME/bin/python
$PYTHON setup.py install
```

install GPAW

GPAW will build with the XL legacy MPI wrapper script. It is recommended that you statically link as many libraries as possible into GPAW to avoid potential performance bottlenecks in loading shared libraries at scale. This can be done with some modification of the stock GPAW config.py file config_mira.py.

```
python=...
$python setup.py build_ext --customize=customize_mira_xlc_mpi.py
```

customize_mira_xlc_mpi.py

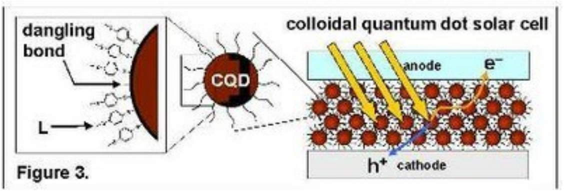
```
define_macros += [('GPAW_NO_UNDERSCORE_BLAS', '1')]
define_macros += [('GPAW_NO_UNDERSCORE_LAPACK', '1')]
define_macros += [('GPAW_NO_UNDERSCORE_CBLACS', '1')]
define_macros += [('GPAW_NO_UNDERSCORE_CSCALAPACK', '1')]
define_macros += [('GPAW_NO_UNDERSCORE_BLACS', '1')]
define_macros += [('GPAW_NO_UNDERSCORE_SCALAPACK', '1')]
define_macros += [('GPAW_ASYNC', 1)]
define_macros += [('GPAW_MPI2', 1)]
define_macros += [('GPAW_PERFORMANCE_REPORT',1)]
scalapack = True
hdf5 = True
libraries = [
           'scalapack',
           'lapack',
           'esslsmpbg',
           'xlf90_r',
           'xlopt',
           'xl',
           'xlfmath',
           'xlsmp',
            ]
import os
python_base = '/scinet/bgq/tools/Python/scalable-python-2.6.7-cnk-gcc'
library_dirs = [
           '/scinet/bgq/Libraries/lapack/lib',
           '/scinet/bgq/Libraries/lapack/lib',
           '/opt/ibmmath/essl/5.1/lib64',
           '/opt/ibmcmp/xlf/bg/14.1/bglib64',
           '/opt/ibmcmp/xlsmp/bg/3.1/bglib64',
           '/scinet/bgq/tools/Python/scalable-python-2.6.7-cnk-gcc/',
           '/scinet/bgq/Libraries/HDF5-1.8.12/mpich2-gcc4.8.1/lib/',
           ]
include_dirs += [
    '/scinet/bgq/tools/Python/scalable-python-2.6.7-cnk-gcc/lib/python2.6/site-packages/numpy/core/include/',
    '/scinet/bgq/Libraries/HDF5-1.8.12/mpich2-gcc4.8.1/include/'
    ]
```

customize_mira_xlc_mpi.py

```
mpi_libraries = [
    'hdf5',
    'mpich',
    'opa',
    'mpl',
    'pami',
    'SPI',
    'SPI_cnk',
    'stdc++',
    ]
mpi_library_dirs = [
    '/bgsys/drivers/ppcfloor/comm/xl.legacy/lib',
    '/bgsys/drivers/ppcfloor/comm/sys/lib',
    '/bgsys/drivers/ppcfloor/spi/lib',
    '/scinet/bgq/Modules/modulefiles/hpctw',
    ]
extra_link_args = ['-Wl,-export-dynamic']
compiler = "./bgq_xlc.py"
mpicompiler = "./bgq_xlc.py"
mpilinker = "./bgq_xlc_linker.py"
```

Renewable solar energy

Ted Sargent's research team (University of Toronto) uses colloidal quantum dots (cqds), semiconductor particles just a few manometers wide, to create a solar ink that can be painted onto almost any surface. this could allow the mass production of solar cells on a previously impossible scale and cost.



CQD solar cells have been around since 2005 and have made tremendous progress from 0.1% to 8.5% published efficiency. To further improve their efficiency, SOSCIP IBM Blue Gene/Q supercomputer used to run computationally intensive simulations of CQDs on an atomic scale. An efficiency of 10% will enable CQDs to compete compellingly with more costly and inflexible conventional technologies for generating energy.

Renewable solar energy

To reach this milestone, Ted Sargent's team will have to understand more about the properties of the materials used to produce the solar cells. Small defects at the atomic level within the cell can trap electronic charges instead of converting them into useful energy. Using the Blue Gene/Q to run simulations of a single quantum dot that his team has built atom-by-atom.
Determine the origin of these traps and remove them, improving the efficiency of the cell.

GPAW is used for electronic structure calculations to understand the properties of material at the quantum level.

Conclusions

- Python is a powerful tool on BlueGene
- many packages work on BlueGene : Numpy, Scipy, Cython, h5py, mpi4py, astropy ...
- would like to investigate openmp with Cython to develop hybrid python code (so far just tested pure mpi code on BGQ)
- waiting for the h5py fix which has an issue to store variable length strings (byte order glitch related to opaque types)