How to Build Various Common Applications and Libraries on Beagle2

While many applications and libraries are already provided by the Beagle support staff, you may find you need to compile these yourself with local modifications or requirements. Below we provide how we compiled the various applications we provide. This document assumes you are already familiar with how to compile code on Beagle. If you are not, please make sure you understand the Beagle programming environment before moving on.

- BAMTOOLS
- BLAT
- Charm++
- CGA tools
- Cufflinks
- Gromacs
  - Single precision, No MPI
  - Single precision, With MPI
  - Double precision, No MPI
  - Double precision, With MPI
- Gromacs 4.5.5 single precision with MPI: custom installation
- Hoard lib
- Sun java
- Swift
- GSL (GNU Scientific Library)
- LAGAN
  - GNU Compiler
  - PGI Compiler
- LAMMPS
  - GNU Compiler
- METIS
- NAMD
  - GNU Compiler
  - PGI Compiler
- NCBI Blast+
  - GNU Compiler
- Netlib CBLAS
- PCRE
  - GNU compiler
- Python
  - Biopython
  - NumPy
  - PIL
  - SciPy
- R
  - GNU Compiler
- SuiteSparse
- SGA -- String Graph Assembler
  - GNU Compiler
- SKAT
- SWIG
  - GNU Compiler
- Valgrind
  - GNU Compiler
- VARIANT TOOLS

BAMTOOLS

Currently only compiles with GNU compiler.

git clone git://github.com/pezmaster31/bamtools.git

module switch PrgEnv-cray PrgEnv-gnu

module load cmake

mkdir build

CD "$BUILD_DIR"

XTPE_LINK_TYPE=dynamic
CC="cc -dynamic" CXX="CC -dynamic" BamTools_SOURCE_DIR="/lustre/beagle/<bamtools_src>" cmake ..

make

Edit makefile to add "-dynamic" to the the last step, building the binaries:

make install DESTDIR=<you_dest_dir>
Move following libraries into the path, as the installation doesn't seem to work properly:

#move files to the library path of bamtools (for some reason they did not get copied)
cp /lustre/beagle2/denovo/src/bamtools/lib/ibamtools-utils.so.2.1.0 <your_dest_dir>/lib

cp /lustre/beagle2/denovo/src/bamtools/lib/libjsoncpp.so.1.0.0 <your_dest_dir>/lib/

**BLAT**

Currently only compiles with GNU compiler.

Setup your environment:

```
module switch PrgEnv-cray PrgEnv-gnu
```

Apply the patch file and compile:

```
export BLAT_INSTALL=/soft/blat/gnu/3.4/bin
mkdir -p ${BLAT_INSTALL}
export MACHTYPE=x86_64
make BINDIR=${BLAT_INSTALL} CC=cc CFLAGS=-O3
```

**Charm++**

Charm++ currently only compiles with the PGI and GNU compilers and there is no difference in how you compile with either. Below is an example using the PGI compiler.

NOTE: for the time being we do not have PGI compiler, it might become available in the future.

Setup your environment:

```
module load rca petsc
```

Configure and compile:

```
env MPICXX=CC MPICC=cc ./build \
   charm++ \n   mpi-crayxt \n   --no-build-shared \n   --with-production \n   -j4 \n   -O optimize \n   -DCMK_OPTIMIZE=1
env MPICXX=CC MPICC=cc ./build \
   AMPI \n   mpi-crayxt \n   --no-build-shared \n   --with-production \n   -j4 \n   -O optimize \n   -DCMK_OPTIMIZE=1
env MPICXX=CC MPICC=cc ./build \
   FEM \n   mpi-crayxt \n   --no-build-shared \n   --with-production \n   -j4 \n   -O optimize \n   -DCMK_OPTIMIZE=1
```

Test compilation:

```
cd tests
make
```

# Verify there are no errors when compiling the tests
CGA tools

NOTE: for the time being we do not have PGI compiler, it might become available in the future.

vers=1.5.0.31
vers=1.7.1
subvers=${vers}.5
wget http://sourceforge.net/projects/cgatools/files/${vers}/cgatools-${subvers}-source.tar.gz/download

tar -xvf cgatools-${subvers}-source.tar.gz

module load cmake
module swap PrgEnv-cray PrgEnv-gnu
module load boost

cd cgatools-${subvers}-source
   cd build
   # WARNING: check the current version and make sure that the
   # version of gcc used is compatible with cgatools and with the libraries
   # it is using (e.g., boost)
   module swap PrgEnv-gnu/4.1.40 PrgEnv-gnu/3.1.49A
   module swap gcc/gcc/4.7.2 gcc/gcc/4.6.1
   module swap xt-mpt xt-mpich2
   module swap xt-mpich2/5.6.1 xt-mpich2/5.3.2

cmake -DBOOST_ROOT=/$BOOST_ROOT \
   -DCMAKE_CXX_COMPILER=CC -DCMAKE_C_COMPILER=cc \
   -DCMAKE_INSTALL_PREFIX=/soft/cgatools/gnu/${vers} \
   -DCMAKE_BUILD_TYPE=Release \
   /lustre/beagle2/genomicsTools/soft/src/cgatools-${subvers}-source

   make -j8
   ctest -j8
   make install

cd /soft/modulefiles/applications/cgatools
   ln -s ../../templates/cgatools $vers

Cufflinks

http://cufflinks.cbcb.umd.edu/downloads/cufflinks-1.3.0.tar.gz

tar -xvf cufflinks-1.3.0.tar.gz

module load python
module load boost
module load samtools

(Needed to create include file on samtools, create a subdir bam there and copy there all the .h files)

CXX=CC cc=cc PYTHON=`which python` ./configure --prefix=/soft/cufflinks/gnu/1.3.0 --with-boost=$BOOST_ROOT --with-bam=$SAM_HOME

for x in *.cpp *.h; do sed 's/foreach/for_each/' $x > x; mv x $x; done

Replace line 24 of common.h

//#include <+boost/for_each.hpp>

#include <+boost/foreach.hpp>

# After because boost was made with 4.6.1
module swap gcc/gcc/4.6.1 gcc/gcc/4.5.2

make

make install

Gromacs
Gromacs can be built 4 different ways: single precision no MPI, single precision with MPI, double precision no MPI, double precision with MPI. There is no difference in how you compile Gromacs with the different compilers. Below are examples using the PGI compiler. You can find more details here: http://www.gromacs.org/Documentation/Installation_Instructions#Details_for_building_the_FFTW_prerequisite

Before building Gromacs, you need to setup your environment:

NOTE: for the time being we do not have PGI compiler, it might become available in the future.

```
module unload dmapp
module load fftw gsl
```

**Single precision, No MPI**

```
./configure \\nCC=cc \\
CXX=CC \\
F77=ftn \\
CFLAGS="${FFTW_INCLUDE_OPTS} ${GSL_INCLUDE_OPTS}" \\
--prefix=/soft/gromacs/pgi/4.5.3 \\
--disable-shared \\
--enable-fortran \\
--with-fft=fftw3 \\
--with-gsl \\
--disable-threads \\
--enable-all-static \\
--enable-static \\
--disable-mpi \\
--with-external-blas \\
--with-external-lapack \\
--without-xml
make
make install
```

**Single precision, With MPI**

```
./configure \\
CC=cc \\
CXX=CC \\
F77=ftn \\
CFLAGS="${FFTW_INCLUDE_OPTS} ${GSL_INCLUDE_OPTS}" \\
--prefix=/soft/gromacs/pgi/4.5.3 \\
--disable-shared \\
--enable-fortran \\
--with-fft=fftw3 \\
--with-gsl \\
--disable-threads \\
--enable-all-static \\
--enable-static \\
--enable-mpi \\
--program-suffix="_mpi" \\
--with-external-blas \\
--with-external-lapack \\
--without-xml
make
make install
```

**Double precision, No MPI**
Double precision, With MPI

Gromacs 4.5.5 single precision with MPI: custom installation

Before you start installation make sure to have following in your module list:
Currently Loaded Modulefiles:
1) modules/3.2.6.7
2) nodestat/2.2-1.0502.53712.3.109.gem
3) sdb/1.0-1.0502.55976.5.27.gem
4) alps/5.2.1-2.0502.9072.13.1.gem
5) lustre-cray_gem_s/2.5_3.0.101_0.31.1.1_0.0502.8394.10.1-1.0502.17198.8.50
6) udreg/2.3.2-1.0502.9275.1.25.gem
7) ugni/5.0-1.0502.9685.4.24.gem
8) gni-headers/3.0-1.0502.9684.5.2.gem
9) dmapp/7.0.1-1.0502.9501.5.211.gem
10) xpmem/0.1-2.0502.55507.3.2.gem
11) hss-1lm/7.2.0
12) base-opts/1.0.2-1.0502.53325.1.2.gem
13) craype-network-gemini
14) craype/2.2.1
15) craype-abudhabi
16) ci
17) moab/6.1.1
18) torque/2.5.7
19) cmake/2.8.4
20) gcc/4.9.1
21) totalview-support/1.2.0.3
22) totalview/8.14.1
23) cray-libsci/13.0.1
24) pml/5.0.6-1.0000.10439.140.3.gem
25) atp/1.7.5
26) PrgEnv-gnu/5.2.40
27) cray-mpich/7.0.5
28) fftw/3.3.0.4
29) gsl/1.15

export CPPFLAGS="-I/opt/fftw/3.3.0.4/abudhabi/include"
export LD_LIBRARY_PATH=/opt/fftw/3.3.0.4/abudhabi/lib:$LD_LIBRARY_PATH
export CRAY_LIBSCI_PREFIX_DIR=/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:/opt/cray/libsci/13.0.1/CRAY/83/x86_64/lib64:
export LDFLAGS=""
export CFLAGS="$(GSL_INCLUDE_OPTS) -static-libgcc -static-libstdc++ -static-libgfortran"
export LIBTOOL="/usr/bin/libtool --tag=CC --mode=link gcc -all-static"

./configure \
CC="cc" \
CXX="CC" \
P77="f77" \
--prefix=/lustre/beagle2/<your_installation_directory> \ 
--disable-shared \
--disable-fortran \
--with-fftw=fftw3 \
--with-gsl \
--disable-threads \
--enable-all-static \
--enable-static \
--enable-mpi \
--program-suffix="_mpi" \
--with-external-blas \
--with-external-lapack \
--without-xml
make
make install

To run Gromacs 4.5.5 on Beagle2, put this in your PBS script:

#!/bin/bash
#PBS -N <job_name>
#PBS -j oe
#PBS -l walltime=48:00:00
#PBS -l mppwidth=64  #this number can be any multiple of 32, since there is 32 cores/node, and this number must match aprun -n 32 (in this case)
source /opt/modules/default/init/bash
cd $PBS_O_WORKDIR
module load fftw/3.3.0.4
module load gromacs/4.5.5-single
aprun -n 1 -N 1 /soft/gromacs/gnu/4.5.5-single/bin/grompp -f tip3pBox60_equi.mdp -c tip3pBox60_preequil.gro -n tip3pBox60.index -p tip3pBox60.top -o tip3pBox60_equi.tpr
aprun -n 64 /soft/gromacs/gnu/4.5.5-single/bin/mdrun -s tip3pBox60_equi.tpr -o tip3pBox60_equi.trr -x tip3pBox60_equi.xtc -e tip3pBox60_equi.edr -g tip3pBox60_equi.log
To learn more about Gromacs installation please visit this page: http://www.gromacs.org/Documentation/Installation_Instructions_4.5

To learn more how to run Gromacs please visit this page: http://www.gromacs.org/@api/deki/files/213/=gromacs_parallelization_acceleration.pdf

Hoard lib

This is an alternate version of java other than the IBM java that is installed by default on Beagle.
http://www.cs.umass.edu/%7Eemery/hoard/hoard-3.8/source/hoard-38.tar.gz

module swap PrgEnv-cray PrgEnv-gnu

replace all g++ with CC in makefile

create lib and move .so there

create include and move .h there.

Sun java

This is an alternate version of java other than the IBM java that is installed by default on Beagle.

Setup your environment:

module load sun-java

Swift

The Swift parallel scripting framework.

Setup your environment:

module load swift

GSL (GNU Scientific Library)

There is one slight difference in how you compile GSL with the PGI and Cray from GNU compilers. With the PGI and Cray compilers, you need to do the following immediately after running configure:

sed 's;#define HAVE_INLINE 1;#undef HAVE_INLINE;g' -i config.h

With GNU this is not required.

Below is an example using the PGI compiler.

NOTE: for the time being we do not have PGI compiler, it might become available in the future.

export GSL_VERSION=1.14
./configure \
 CC=cc \
 --prefix=/soft/gsl/pgi/$GSL_VERSION \
 --disable-shared
sed 's;#define HAVE_INLINE 1;#undef HAVE_INLINE;g' -i config.h
make
make install
LAGAN
LAGAN compiles with bot PGI and GNU compilers.

NOTE: for the time being we do not have PGI compiler, it might become available in the future.

**GNU Compiler**
Setup your environment:

```bash
module switch PrgEnv-cray PrgEnv-gnu
```

Unzip the source and move the resulting directory to the installation destination:

```bash
mkdir -p /soft/lagan/gnu
cd /soft/lagan/gnu
tar zxvf lagan20.tar.gz
mv lagan20 2.0
```

Apply the `src/Makefile` patch and `src/local/Makefile` patch and compile:

```bash
rm -f prolagan
make
```

**PGI Compiler**
Setup your environment:

NOTE: for the time being we do not have PGI compiler, it might become available in the future.

```bash
module switch PrgEnv-cray PrgEnv-gnu
```

Unzip the source and move the resulting directory to the installation destination:

```bash
mkdir -p /soft/lagan/pgi
cd /soft/lagan/pgi
tar zxvf lagan20.tar.gz
mv lagan20 2.0
```

Apply the `src/Makefile` patch and `src/local/Makefile` patch and compile:

```bash
rm -f prolagan
make
```

**LAMMPS**
LAMMPS, currently, only compiles with the GNU compiler.

**GNU Compiler**
Setup you environment:

```bash
module switch PrgEnv-cray PrgEnv-gnu
module load fftw/2.1.5.2
```

- Install `atc Makefile` in `lib/atc`
• Install `meam` Makefile in `lib/meam`
• Install `poems` Makefile in `lib/poems`
• Install `reax` Makefile in `lib/reax`
• Install `Beagle` Makefile in `src/MAKE`

Compile LAMMPS:

cd src
make yes-all
make no-gpu

cd ../lib/atc
make -f Makefile-atc.beagle

cd ../meam
make -f Makefile-meam.beagle

cd ../poems
make -f Makefile-poems.beagle

cd ../reax
make -f Makefile-reax.beagle

cd ../../src
sed -i -e 's/fftw.h/dfftw.h/g' fft3d.h
make beagle

METIS

Download either the **PGI Makefile** or **GNU Makefile**. The following example assumes PGI:

NOTE: for the time being we do not have PGI compiler, it might become available in the future.

```
rm -f Makefile.in; ln -sf Makefile-pgi.in Makefile.in
make
mkdir -p /soft/metis/pgi/4.0.1/bin
mkdir -p /soft/metis/pgi/4.0.1/lib
cp graphchk mesh2dual oemetis partdmesh pmetis kmetis mesh2nodal onmetis partnmesh /soft/metis/pgi/4.0.1/bin/
cp libmetis.a /soft/metis/pgi/4.0.1/lib/
```

NAMD

**GNU Compiler**

Setup you environment:

```
module switch PrgEnv-cray PrgEnv-gnu
module load rca petsc charm++ tcl fftw/2.1.5.2
```

**Edit** `Make.charm` **and make sure** `CHARMBASE` **points to the install of Charm++**:

```
CHARMBASE = $(CHARM_DIR)
```

**Edit** `arch/CRAY-XT.fftW` **so it looks like**:

```
FFTDIR=$(FFTW_DIR)
FFTNCL=$(FFTW_INCLUDE_OPTS)
FFTLIB=$(FFTW_POST_LINK_OPTS) -lfftw -ldfftw
FFTFLAGS=-DNAMD_FFTW
FFT=$(FFTNCL) $(FFTFLAGS)
```

**Edit** `arch/CRAY-XT.tcl` **so it looks like**:
Edit arch/CRAY-XT-g++.arch so it looks like:

NAMD_ARCH = CRAY-XT
CHARMARCH = mpi-crayxt

# The GNU compilers produce significantly faster NAMD binaries than PGI.
# You must run the following to switch CC/cc to the GNU compiler
# environment before building either Charm++ or NAMD:
# module swap PrgEnv-cray PrgEnv-gnu
# Users of psfgen might also need to do 'module remove acml' in order for
# the the psfgen compilation to succeed.
#
CXX = CC -DNO_SOCKET -DDUMMY_VMDSOCK -DNOHOSTNAME -DNAMD_NO_STDOUT_FLUSH -DNAMD_NO_O_EXCL
CXXOPTS = -O3 -ffast-math -fexpensive-optimizations -fomit-frame-pointer
CC = cc
COPTS = -O3 -ffast-math -fexpensive-optimizations -fomit-frame-pointer

Compile NAMD:

./config CRAY-XT-g++.arch
cd CRAY-XT-g++
make -j4

PGI Compiler

Setup you environment:

NOTE: for the time being we do not have PGI compiler, it might become available in the future.
module load rca petsc charm++ tcl fftw/2.1.5.2

Edit Make.charm and make sure CHARMBASE points to the install of Charm++:

CHARMBASE = $(CHARM_DIR)

Edit arch/CRAY-XT.fftw so it looks like:

FFTDIR=$(FFTW_DIR)
FFTINCL=$(FFTW_INCLUDE_OPTS)
FFTLIB=$(FFTW_POST_LINK_OPTS) -lfftw -ldfftw
FFTFLAGS=-DNAMD_FFTW
FFT=$(FFTINCL) $(FFTFLAGS)

Edit arch/CRAY-XT.tcl so it looks like:

TCLDIR=$(TCL_DIR)
TCLINCL=$(TCL_INCLUDE_OPTS)
TCLLIB=$(TCL_POST_LINK_OPTS) -ldl
TCLFLAGS=-DNAMD_TCL
TCL=$(TCLINCL) $(TCLFLAGS)

Edit arch/CRAY-XT-pgcc.arch so it looks like:
NAMD_ARCH = CRAY-XT
CHARMARCH = mpi-crayxt

# The GNU compilers produce significantly faster NAMD binaries than PGI.
#

CXX = CC
CXXOPTS = -O
CXXNOALIASOPTS = -fast -Mnodepchk -Msafeptr=arg,global,local,static -Minfo=all -Mneginfo=loop
CC = cc
COPTS = -fast

Compile NAMD:

./config CRAY-XT-pgcc.arch
cd CRAY-XT-pgcc
make -j4

NCBI Blast+

We currently only have instructions for compiling Blast+ with the GNU compiler

GNU Compiler

Setup your environment:

module switch PrgEnv-cray PrgEnv-gnu

Configure and compile Blast+:

./configure \ 
  CC=cc \ 
  CXX=CC \ 
  --prefix=/soft/ncbi-blast+/gnu/2.2.24 \ 
  --without-dll \ 
  --with-static \ 
  --with-static-exe \ 
  --with-lfs \ 
  --with-64 \ 
  --without-debug
make
make install

Netlib CBLAS

You can only compile CBLAS using the PGI or GNU compilers. There is no difference in how you compile Netlib CBLAS, only which Makefile you link Makefile.in to. We provide the PGI Makefile and the GNU Makefile. The examples below use the PGI compiler.

NOTE: for the time being we do not have PGI compiler, it might become available in the future.

Setup your environment:

module load acml

Compile and install CBLAS:

rm -f Makefile.in
ln -sf cblas-Makefile.pgi Makefile.in
make all
mkdir -p /soft/cblas/sgi/3.0/lib
cp lib/libcblas.a /soft/cblas/sgi/3.0/lib
PCRE

GNU compiler

We compile PCRE for both shared library support (for SWIG) and for static library support (for Octave). Because compiles default to static libraries, we compile that last.

Setup your environment:

```bash
module switch PrgEnv-cray PrgEnv-gnu

export XTPE_LINK_TYPE=dynamic
```

Configure, build and install the shared library version:

```bash
export XTPE_LINK_TYPE=native

./configure \n  CC="cc -dynamic" \n  CXX="CC -dynamic" \n  --prefix=/soft/pcre/gnu/8.12 \n  --enable-shared \n  --disable-static
make
make install
```

Configure, build and install the static library version:

```bash
./configure \n  CC=cc \n  CXX=CC \n  --prefix=/soft/pcre/gnu/8.12 \n  --disable-shared \n  --enable-static
make
make install
```

Python

It only makes sense to compile Python dynamically since in almost all cases you will want to load Python modules dynamically. Because of this, you have to use Python in DSL mode. We also only compile Python with the GNU compiler.

Setup you environment:

```bash
module switch PrgEnv-cray PrgEnv-gnu
module unload xt-mpt pmi

Configure and compile:

```bash
./configure \n  CC="cc -dynamic" \n  CXX="CC -dynamic" \n  F77="ftn -dynamic" \n  XTPE_LINK_TYPE=dynamic \n  --prefix=/soft/python/2.7.1 \n  --enable-shared
make
make install
```

Biopython

Setup your environment:

```bash
module load python
export XTPE_LINK_TYPE=dynamic
export BIOPYTHON_ROOT=/soft/python/modules/biopython/1.56
```
Build Biopython:

```bash
mkdir -p ${BIOPYTHON_ROOT}/lib/python2.7/site-packages
env LDFLAGS="-L/soft/python/2.7/2.7.1/lib" python setup.py build PYTHONPATH=${PYTHONPATH}:${BIOPYTHON_ROOT}/lib/python2.7/site-packages python setup.py install --prefix ${BIOPYTHON_ROOT}
```

**NumPy**

```bash
module swap PrgEnv-cray PrgEnv-gnu

module load python/2.7.3-vanilla
module load acml
module loadmetis
module load Suitesparse
module load swig
export XTPE_LINK_TYPE=dynamic
export NUMPY_ROOT=/soft/python/2.7/2.7.3-vanilla/modules/numpy/1.7.0

python setup.py build --compiler=unix --fcompiler=gnu95
```

```bash
PYTHONPATH=${PYTHONPATH}:${NUMPY_ROOT}/lib/python2.7/site-packages python setup.py install --prefix ${NUMPY_ROOT}
```

**PIL**

**Setup your environment:**

```bash
module load python
export XTPE_LINK_TYPE=dynamic
export IMAGING_ROOT=/soft/python/modules/imaging/1.1.7

Apply the patch file and compile:

```bash
mkdir -p ${IMAGING_ROOT}/lib/python2.7/site-packages
env LDFLAGS="-L/soft/python/2.7/2.7.1/lib" python setup.py build PYTHONPATH=${PYTHONPATH}:${IMAGING_ROOT}/lib/python2.7/site-packages python setup.py install --prefix ${IMAGING_ROOT}
```

**SciPy**

```bash
module load pil
export XTPE_LINK_TYPE=dynamic
export SCIPY_ROOT=/soft/python/2.7/2.7.3-vanilla/modules/scipy/0.12.0
mkdir -p ${SCIPY_ROOT}/lib/python2.7/site-packages
export BLAS="/opt/cray/libsci/12.0.00/GNU/47/mc12/lib/libsci_gnu.a"
export LAPACK="/opt/cray/libsci/12.0.00/GNU/47/mc12/lib/libsci_gnu.a"
export ATLAS="/opt/cray/libsci/12.0.00/GNU/47/mc12/lib/libsci_gnu.a"
python setup.py build --compiler=unix --fcompiler=gnu95

repeat at eternum adding -shared to build shared libraries when the idiotic linker is called incorrectly. NEED TO FIND A WAY TO TELL TO THE INSTALLER HOW TO USE GCC TO LINK THE FORTRAN LIBRARIES.

export PYTHONPATH=${PYTHONPATH}:${SCIPY_ROOT}/lib/python2.7/site-packages python setup.py install --prefix ${SCIPY_ROOT}
```

**R**

It only makes sense to compile R dynamically since in almost all cases you will want to load R modules dynamically. Because of this, you have to use R in DSL or CCM mode.
**GNU Compiler**

This has been successfully tested with R up to version 2.13.1 (not all intermediate releases have been tested)

Setup your environment:

If using PrgEnv-gnu/3.1.49A or earlier:

```
module switch PrgEnv-cray PrgEnv-gnu/3.1.49A
```

This might be necessary because some of the later versions have issues with shared libraries. Otherwise

```
module switch PrgEnv-cray PrgEnv-gnu
```

is sufficient because cray-mpich is the current default (unless you changed it)

Configure and compile:

```
./configure \
  CC="cc -dynamic" \ 
  CXX="CC -dynamic" \ 
  FC="f77 -dynamic" \ 
  F77="/f77 -dynamic" \ 
  XTPR_LINK_TYPE="dynamic" \ 
  CPICFLAGS="-fPIC" \ 
  CPICFLAGS="-fPIC" \ 
  FPICFLAGS="-fPIC" \ 
  FCFCFLAGS="-fPIC" \ 
  SHLIB_LDFLAGS="-dynamic" \ 
  SHLIB_CXXLDFLAGS="-dynamic" \ 
  --prefix="/soft/R/gnu/2.12.1" \ 
  --enable-R-shlib \ 
  --enable-R-shared \ 
  --enable-blas="-lsci -lsci_mc12" \ 
  --with-lapack="-lsci -lsci_mc12" \ 
  --without-x
make
make install
```

Note that `--prefix="/soft/R/gnu/2.12.1"` should point to a local directory (probably under `/lustre/beagle/whoami/soft/R`) if you are building it for yourself.

**SuiteSparse**

SuiteSparse currently only builds with GNU compilers. Download the `UFconfig.mk` configuration file and install it as `/soft/R/UFconfig/UFconfig.mk`.

Setup your environment:

```
module switch PrgEnv-cray PrgEnv-gnu
module load metis
```

Copy the configured and built source tree of METIS to the source root of SuiteSparse as `metis-4.0` so that your SpareSuite root looks like:

```
[jurbanski@sandbox:/soft/build/SuiteSparse]$ ls
AMD/        CSparse_to_CXSparse*      MATLAB_Tools/       SuiteSparse_install.m
BTF/        CSparse/                 MATLAB_Tools.tar.gz
CAMD/       CSparse_newfiles/        MATLAB_Tools.tar.gz
CCHOLMOD/   CSparse_newfiles.tar.gz  SuiteSparse_install.m
SPARSLIB/   KLU/                     SuiteSparse_install.m
CHOLMOD/    LINFACTOR/               SuiteSparse_install.m
COLAMD/     LINFACTOR.tar.gz          SuiteSparse_install.m
SuiteSparse/ Makefile                 SuiteSparse_install.m
```

Build and install SuiteSparse:

```
make
make install
```
SGA -- String Graph Assembler

GNU Compiler

git clone git://github.com/jts/sga.git

module load sparsehash
module load bamtools
module load hoard

./autogen.sh

CXX=CC cc=cc ./configure --with-sparsehash=${SPARSEHASH_DIR} --with-bamtools=${BAMTOOLS_HOME} --with-hoard=${HOARD_DIR}

make install DESTDIR=<your_dest_sir>

SKAT

module load R/2.15.1-vanilla
vers=0.82
wget http://cran.r-project.org/src/contrib/SKAT_${vers}.tar.gz
gunzip SKAT_${vers}.tar.gz

# We try with CC, which is the wrapper for MPI and general system builds
# it is possible that for the package to work properly under Vanilla we
# will need to use CC=g++ cc=gcc
R CMD INSTALL --configure-args="--prefix=$R_HOME --build=X86_64  CC=CC cc=cc"  SKAT_${vers}.tar

SWIG

GNU Compiler

Because SWIG is used with languages such as Python that are dynamically linked, we build SWIG dynamically.

Setup your environment:

module switch PrgEnv-cray PrgEnv-gnu
module load pcre
module load python
export XTPE_LINK_TYPE=dynamic

Configure, build and install SWIG:

./configure 
CC="cc -dynamic"
CXX="CC -dynamic"
  --prefix=/soft/swig/gnu/2.0.2
make
make install

Valgrind

GNU Compiler

Valgrind is an instrumentation framework for building dynamic analysis tools. The version that ships from Cray will not work on the compute nodes since it requires /tmp to be writable. We will build a patched version that will honor a user’s $TMPDIR environment variable.

Setup your environment:

module switch PrgEnv-cray PrgEnv-gnu
Apply the patch to the Valgrind 3.6.1 source tree:

```bash
tar jcvf valgrind-3.6.1.tar.bz2
cd valgrind-3.6.1
cp /path/to/patch/file/tmpdir.patch .
patch -p1 < tmpdir.patch
```

Configure, build and install Valgrind, allowing it to find the default gcc:

```bash
./configure \      
   --prefix=/soft/valgrind/gnu/3.6.1
make
make install
```

To use Valgrind on the compute nodes you will need to set `TMPDIR` to something other than `/tmp`.

**VARIANT TOOLS**

```bash
wget http://sourceforge.net/projects/varianttools/files/1.0.4/variant_tools-1.0.4a.tar.gz/download
```

```bash
tar -xvf variant_tools-1.0.4a.tar.gz
```

```bash
module load python/2.7.3-vanilla
```

```bash
python setup.py install --install-platlib=/soft/variant_tools/gnu/1.0.4/python_lib --install-scripts=/soft/variant_tools/gnu/1.0.4/bin
```