Build and Run Charm++ and NAMD on the Cray XC with CrayPat

What is the purpose of this document?

This document is intended to provide information and guidance on how to build Charm++ so that programs which use it can be profiled with CrayPat.

An application which uses Charm++, NAMD, is provided as one example.

It is not possible to test the source on every combination of software used at Cray customer sites. However, this file should provide a basis from which building and running on your own site should be significantly easier.

What is Charm++?

“Charm++ is a message-passing parallel language and runtime system. It is implemented as a set of libraries for C++, is efficient, and is portable to a wide variety of parallel machines. Source code is provided, and non-commercial use is free.”

See: http://charm.cs.illinois.edu/manuals/html/faq/manual.html

What is NAMD?

“NAMD, recipient of a 2002 Gordon Bell Award and a 2012 Sidney Fernbach Award, is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. Based on Charm++ parallel objects, NAMD scales to hundreds of cores for typical simulations and beyond 500,000 cores for the largest simulations.”

See: http://www.ks.uiuc.edu/Research/namd/

What version of Charm++ is required?

Porting Charm++ to the Cray XC was based off of charm-v6.7.1.

Changes were made to the 6.7.1 source and submitted to the Charm++ developers in October 2016. It is believed these changes will be incorporated into the next release of Charm++, either 6.7.2 or 6.8.0. Changes can also be obtained by checking out the latest GIT source at: http://charm.cs.uiuc.edu/software

What version of NAMD is required?

Porting NAMD to the Cray XC30 was based off NAMD v2.11.

A few minor changes might be required in order for you to build NAMD on the Cray XC, but otherwise should build without issue. You can download NAMD here:

http://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=NAMD

What Cray software is required?

The testbed for the porting work was a Cray XC running CLE/6.0 and PBSPro/12.2.

The port is expected to work on any Cray XC system running CLE/5.2 or CLE/6.0. However, additional changes might be required for XC systems running CLE/5.2; or systems using workload managers other than PBSPro/12.2.

The following Cray Programming Environments were tested:

PrgEnv-cray/6.0.4	**PrgEnv-intel/6.0.4**	**PrgEnv-gnu/6.0.4**
with	with	with
cce/8.5.4	**intel/16.0.3.210**	**gcc/6.1.0**
cce/8.5.5	**intel/17.0.0.028**	**gcc/6.2.0**

Earlier versions of these compilers are not supported with this port.

In addition, the following cray modules were used:

```
craype-network-aries
craype-ivybridge
craype-hugepages-8M
craype/2.5.8.2 or later
cray-mpich/7.4.4 or later
cray-libsci/16.0.9.1 or later
perftools-base/6.4.4 or later
```

Using earlier versions of these modules may lead to build- or run-time problems, and are not supported. Perftools is only supported with Charm++/NAMD as of 6.4.4 or later.

Many other Cray-provided modules were used, but are considered part of the system software and are not included here.

Build Charm++

Before building Charm++, you must choose which compiler you wish to build with. **It is required that you use the same compiler to build both Charm++ and NAMD**. Mixed-compiler combinations may work but are not supported.

Building for Perftools

Do not load perftools or perftools-lite when building Charm++ (perftools-base can be safely loaded prior to building). Instead, ensure frame pointers are not omitted in your conv-mach.sh file (this is the new default for Cray XC builds). These options are provided in the new conv-mach.sh file submitted to the Charm++ developers

Common for all Compilers

Do not set this env variable when building charm++. This will cause the build to fail. On some systems, dynamic linking is the default. In such a case, you will need to unset this env variable.

```
# DO NOT SET: setenv CRAYPE_LINK_TYPE dynamic

# DO NOT SET: export CRAYPE_LINK_TYPE=dynamic
```

Load initial modules

```
% module load craype-ivybridge     # craype-haswell,
                                   # craype-broadwell not tested
% module load craype-hugepages8M
% module load rca
```

Load Programming Environment modules and build

Choose a compiler, and then choose either uGNI or MPI.

CRAY COMPILER (CCE) BUILD

```
% module load PrgEnv-cray          # typically default
% module swap cce cce/8.5.4        # cce/8.5.4 or later required
```

You must add '`craycc`' to your build options or the build will fail.

uGNI build

```
% ./build charm++ gni-crayxc craycc smp
```

MPI build

```
% ./build charm++ mpi-crayxc craycc smp
```

INTEL® COMPILER BUILD

```
% module swap PrgEnv-cray PrgEnv-intel
% module swap intel intel/16.0.3.210   # intel/16.0.3.210,
                                       # intel/17.0.0.098 tested
```

Do not add '`icc`' to your build options or the build will fail.

uGNI build

```
% ./build charm++ gni-crayxc smp -optimize
```

MPI build

```
% ./build charm++ mpi-crayxc smp -optimize
```

GNU COMPILER BUILD

```
% module swap PrgEnv-cray PrgEnv-gnu
% module swap gcc gcc/6.2.0        # gcc/6.1.0, 6.2.0 tested
```

Do not add '`gcc`' to your build options or this will fail to build.

uGNI build

```
% ./build charm++ gni-crayxc smp -optimize
```

MPI build

```
% ./build charm++ mpi-crayxc smp -optimize
```

Verify your Charm++ build

Modules

Except as noted below, do not change your loaded modules between building the Charm++ libraries and any subsequent Charm++ tests or applications. **Swapping modules to different versions after building Charm++ is not supported**.

Building with Perftools

After the build of Charm++ is complete,

```
% module load perftools-base
% module load perftools       # or perftools-lite, for example
# Build your application.
#   Ensure you have added -save to your CHARMC line.
#   See the "megatest" example, below.
% pat_build < . . . >         # as desired
```

Note: There are trace groups designed to help you study the performance of Charm++ or converse calls. Use "`pat_build -gcharm++`" or "`pat_build -gconverse`" to use this functionality. See "`man pat_build`".

Build and run megatest

"megatest" is a sanity test provided by Charm++ and is included in the download of the source. This test was chosen as a sanity test because it mimics many of the Charm++ calls that NAMD uses. There are many other Charm++ tests included in the source, and many of them can be run with similar directions:

```
% module swap craype-ivybridge craype-broadwell   # if desired
% module load perftools-base
% module load perftools                           # if desired
% cd tests/charm++/megatest
% vi Makefile
      # point CHARMC to the correct path
      # Add -save for perftools, if desired
      #     e.g.:  /your/path/bin/charmc -save
% make clean
% make
% pat_build -Oapa pgm   # for example
```

Run megatest

This test was run on four Cray XC30 compute nodes containing Intel® Broadwell 36-core chips. There are many different ways to run this test, but these aprun settings were found to be large enough to exercise Charm++ to ensure functionality of the libraries.

```
% setenv OMP_NUM_THREADS 1
% aprun -n 4 -N 2 -S 1 -d 4 ./pgm+pat +ppn2
```

Failures in megatest

Failures seen during the execution of “megatest” will almost guarantee failures for any other application built against your Charm++ libraries.

Therefore, it is highly recommended that you resolve any failures seen with “megatest” before proceeding to build NAMD.

Pat_report for megatest

If you ran “megatest” with perftools, “-s aggr_th” shows work for non-main threads, and is required for seeing data for Charm++ threads.

% pat_report -s aggr_th=avg -o rpt pgm+pat.xf

Build TCL

If all “megatest” tests passed successfully, you can proceed to build NAMD.

Note that **TCL is required to build NAMD**. You’ll need to build TCL on your own, as this library is not provided by Cray.

It is highly recommended that you use the same Programming Environment to build TCL that you used to build Charm++.

The following example shows one of the ways TCL can be built:

```
% ./configure --prefix=/your/path/tcl/tcl8.5.18 --enable-threads
--disable-shared
% make
% make install
```

Point NAMD’s TCL path to your TCL build

```
% cd /your/path/namd/source
% vi arch/CRAY-XC.tcl
      # point TCLDIR to your newly-created TCL directory, e.g.:
      # TCLDIR=/your/path/tcl/tcl8.5.18/install
```

Build NAMD

It is required that you use the same Programming Environment to build NAMD that you used to build Charm++. Mixed-compiler combinations may work, but they are not supported.

Common for all compilers

For NAMD, it does not matter whether you built Charm++ for uGNI or MPI. The build process is the same for NAMD for both options.

Make sure to **set this environment variable** when building NAMD. **You must set this or the build will fail**. This is the opposite of what is required for the Charm++ build.

```
      setenv CRAYPE_LINK_TYPE dynamic
or    export CRAYPE_LINK_TYPE=dynamic
```

If you build NAMD in the same shell that you built Charm++, you will not need to reload many of the Programming Environment modules. However, if you start a new shell, you will need to load all of the modules required to build Charm++ for your chosen compiler. Go back to the Charm++ section and reload any required modules listed there.

Load additional modules

The build of NAMD requires additional modules:

```
% module load fftw
% module load pmi-lib
```

Building for Perftools

Load perftools or perftools-lite when building NAMD. This is required to obtain performance data for NAMD.

```
% module load perftools-base
% module load perftools-lite       # or perftools,
                                   # or perftools-lite-events
```

All options available to `pat_build` are supported for NAMD. In addition, new options are available to study the effect of Charm++ or Converse in your application:

```
% pat_build -gcharm++
% pat_build -gconverse
```

The "`-save`" option passed to "`--charm-opts`" is required when building with perftools. It does not affect non-perftools builds, other than a few temporary files left around after the build.

CRAY COMPILER (CCE) BUILD

uGNI/MPI build

```
% ./config CRAY-XC-cce --charm-arch <your-charm-build> --with-
fftw3 --charm-opts -save
% cd CRAY-XC-cce
% make -j4
```

INTEL® COMPILER BUILD

uGNI/MPI build

```
% ./config CRAY-XC-intel --charm-arch <your-charm-build> --with-
fftw3 --charm-opts -save
% cd CRAY-XC-intel
% make -j4
```

GNU COMPILER BUILD

uGNI/MPI build

```
% ./config CRAY-XC-gnu --charm-arch <your-charm-build> --with-
fftw3 --charm-opts -save
% cd CRAY-XC-gnu
% make -j4
```

Run NAMD

If your build was successful, your resulting binary should be called "`namd2`". You should now be able to run NAMD as normal.

For example, the sample input provided by NAMD called "`apoa1`" could be run like the following example on two Cray XC Broadwell 36-core nodes (place inside your PBS, Moab, or Slurm script):

```
% setenv OMP_NUM_THREADS 1
% aprun –n12 –N6  -d6 -cc depth ./namd2 apoa1.conf +ppn4
```