JÜLICH
FORSCHUNGSZENTRUM

CUBE 4.3.4 – Installation Guide

Generic Display for Application Performance Data

March 19, 2016

The Scalasca Development Team
scalasca@fz-juelich.de

Copyright

Contents

1 CUBE Installation Guide

1.1 Introduction

CUBE (CUBE Uniform Behavioral Encoding) is a presentation component suitable for displaying a wide variety of performance data for parallel programs including MPI and OpenMP applications. CUBE allows interactive exploration of the performance data in a scalable fashion. Scalability is achieved in two ways: hierarchical decomposition of individual dimensions and aggregation across different dimensions. All metrics are uniformly accommodated in the same display and thus provide the ability to easily compare the effects of different kinds of program behavior.

The CUBE package currently consists of five components:

- A simple CUBE writer library written in C.
- A simple CUBE reader library written in Java.
- The full-featured C++ CUBE library for reading and writing CUBE files.
- A set of command-line tools to explore and modify CUBE files.
- A graphical user interface based on the Qt application framework.

The remainder of this document describes the generic installation procedure for the different CUBE components (i.e., the C and C++ libraries, the command-line tools, as well as the graphical user interfaces).

1.2 Availability & License

CUBE is available as a source-code distribution for UNIX or UNIX-like platforms. It can be downloaded from `http://www.scalasca.org`. Besides the standalone distribution, CUBE is also distributed as part of the Scalasca performance analysis toolset available via the same web page.

The CUBE software is free, but by downloading, installing and using it you agree to comply with the license agreement. Please read the file `LICENSE` in the distribution's top-level directory for precise wording.

1.3 Prerequisites

Depending on the components of CUBE that are to be built, various tools and packages are required to be available on the build system. Common requirements—needed to build any of the components—are:

- GNU make
  http://www.gnu.org/software/make/
- zlib (CUBE can be build without zlib support if zlib is not available)
  http://www.zlib.net/

Both are typically already installed on most systems. For zlib, you also need the development headers to be installed (might be provided by a separate package, e.g., on Linux systems).

For the graphical user interface an additional library/framework is required:

- Qt Framework (version 4.6 or higher)
  http://www.qt.io/

1.4 Installation

This section describes the general procedure to build and install the CUBE software. Before proceeding with the instructions given below, **please make sure to also read the Qt installation notes in Appendix 'qt_notes' as well as the platform- and compiler-specific notes in Appendix 'platform' .**

To configure and install the CUBE package, the following steps are usually required:

1. Unpack the sources

   ```
   gunzip -c cube-4.x.x.tar.gz | tar xvf -
   ```

2. Change into the CUBE source code directory

   ```
   cd cube-4.x.x
   ```

3. Create a `VPATH` directory, where do you build the CUBE, and change into it

   ```
   mkdir -p vpath; cd vpath
   ```

4. Run the `configure` script

   ```
   ../configure
   ```

   The configure script tries to determine whether the requirements for building CUBE are fulfilled and sets up the build configuration. The configure script provides a set of various options. The full list is displayed when invoking:

```
../configure --help=recursive
```

A list of main options:

-prefix=*dir* Specifies the installation directory (default: `/opt/cube`)

-with-nocross-compiler-suite=*(gcc|ibm|intel|pathscale|pgi|studio|open64|clang)* The compiler suite to build this package in non cross-compiling environments with. Needs to be in `$PATH` [gcc].

-with-frontend-compiler-suite=*(gcc|ibm|intel|pathscale|pgi|studio|open64|clang)* The compiler suite to build the frontend parts of this package in cross-compiling environments with. Needs to be in $PATH [gcc].

-enable-debug One configures and builds Cube with the debug output. Currently only derived metrics engine has a debug output, which is helpful while developing a remapping specification file or developing a derived metric. An environment variable *CUBEPL_VERBOSE_METRICS* lists metrics, which should be verbose. Possible values are:

a) all - every metric is verbose

b) name - only metric with the unique name "name" is verbose.

Values can be listed using ","

An environment variable *CUBE_DEBUG* defines the verbosity level

a) CUBEPL_EXECUTION - whole CubePL execution is verbose

b) CUBEPL_VARIABLES - only handling of CubePL variables is verbose

-with-cubelib | **-without-cubelib** Enables (default) or disables building and installation of the cube c++ library.

-with-tools | **-without-tools** Enables (default) or disables building and installation of cube tools.

-with-gui | **-without-gui** Enables (default) or disables building and installation of the Cube GUI.

-with-plugin-example | **-without-plugin-example** Enables (default) or disables building and installation of the Cube GUI plugin "↩ Example".

-with-cwriter | **-without-cwriter** Enables (default) or disables building and installation of the C cube writer.

-with-java-reader | **-without-java-reader** Enables (default) or disables the compilation, build and installation of the cube java reader

-with-xerces-path=*path* Specifies the path to the xerces.jar (default↩ : `/usr/share/java`)

-with-xerces-name= ***Name.jar*** Specifies the name of the jar file with the xerces xml parser (default: `xerces.jar`)

-with-frontend-zlib=*"path to frontend zlib"* Defines the zlib library, used by cube on frontend

-with-backend-zlib=*"path to backend zlib"* Defines the zlib library, used by cube on backend

-with-compression=full|ro|none Enables or disables zlib compression support in cube:

full All parts of the framework create compressed cubes and read compressed and uncompressed cube files.

ro Read only configuration. Created cube files are uncompressed, but tools and GUI can open and operate with compressed cube files. This is default value.

none Zlib library is not used in this case and compressed cube files are not supported.

-with-qt=*path* Forces to look for Qt in the `path` in first place. If this option is omitted the configure script searches for a Qt installation using the following sequence:

a) `$QT_DIR`

b) `$PATH`

c) `/usr/local/Trolltech/`

d) `/opt/local/libexec/qt4-mac/` (to support Mac)

e) `/opt/lib/qt4`

f) `/usr/lib/qt/bin`

g) `/usr/lib32/qt4`

h) `/usr/lib32/qt`

i) `/usr/lib64/qt4`

j) `/usr/lib64/qt`

k) `/opt/qt`

l) `/opt/qt4`

m) `/usr/local/qt4`

n) `/usr/local/qt`

-with-qt-specs=*spec* Defines a specs for Qt (Default is "default")

-with-vampir | **-without-vampir** Enables (default) or disables support of TraceBrowser connection with Vampir.

```
The support of TraceBrowser is only possible if DBUS headers and DBUS library are available
The configure script tries to find them automatically. If it fails, you have to specify the
<tt>dbus/dbus.h</tt> and <tt>dbus/dbus-arch-deps.h</tt>. You can do it like

../configure --prefix=/opt/software/cube \
                    CPPFLAGS='-I/usr/include/dbus-1.0  \
                              -I/usr/lib64/dbus-1.0/include'"
```

-with-paraver | -without-paraver Enables support of TraceBrowser connection with Paraver.

See the note above.

-with-paraver-cfg=FILE Uses FILE as a configuration file for Paraver.

-with-buffersize=N Use N bytes buffer for I/O operations (default N=1↩ Mb)

-disable-shared Disables building of a shared CUBE C++ library. (Build with shater libraries is supported only for some compilers/platforms)

-with-strategy=keepall|preload|manual|lastn Selects a default data loading strategy.

a) **keepall** - data is loaded on demand and kept in memory to the end of lyfecycle of the Cube object.

b) **preload** - all data is loaded during the metric initialization and kept in memory to the end of lyfecycle of the Cube object.

c) **manual** - Application should request and drop the data sets explicitly. No correctness check is performed. Therefore one has to use this strategy with care.

d) **lastn** - Only `N` last used data rows are kept in memory. `N` is specified via command line option `-with-nrows=N`

The `configure` script assumes that the GNU C/C++ compilers should be used to compile CUBE. You can select a different compiler using one of the options above e.g.

```
../configure - -with-nocross-compiler-suite=intel
```

If you want to use alternative compilers or compiler optimization options, which are not listed in the description of the options, you can do so by specifying appropriate variables in the command line when invoking `configure`, e.g.,

```
../configure --prefix=/opt/software/cube CC=xlc CFLAGS="-g -O2"
```

For a list of the recognized variables, please inspect the output of the command

```
./configure - -help=recursive
```

5. Start the build process

   ```
   make
   ```

   or

   ```
   make -j N
   ```

6. You can invoke a test suite of the CUBE framework

   ```
   make check
   ```

   If all tests pass, everything looks good. If some tests do not pass, please report it to scalasca@fz-juelich.de and attach the file `configure.log`

7. Install the software

   ```
   make install
   ```

1.5 Configuration

CUBE provides the option of displaying an online description for entries in the metric tree via a context menu. By default, it will search for the given HTML description file on all the mirror URLs specified in the CUBE file. In case there is no Internet connection, the Qt-based CUBE GUI can be configured to also search in a list of local directories for documentation files. These additional search paths can be specified via the environment variable `CUBE_DOCPATH` as a colon-separated list of local directories, e.g.,

```
CUBE_DOCPATH=/opt/software/doc:/usr/local/share/doc
```

Note that this feature is only available in the Qt-based GUI and **not** in the older wx↩ Widgets-based one.

To prevent CUBE from trying to load the HTML documentation via HTTP or HTTPS mirror URLs (e.g., in restricted environments were outbound connections are blocked by a firewall and the timeout is taking very long), the environment variable `CUBE_DISABL↩ E_HTTP_DOCS` can be set to either `1`, `yes` or `true`.

To prevent CUBE from trying to load the HTML documentation via HTTP or HTTPS mirror URLs (e.g., in restricted environments were outbound connections are blocked by a firewall and the timeout is taking very long), the environment variable `CUBE_DISABL↩ E_HTTP_DOCS` can be set to either `1`, `yes` or `true`.

CUBE C++ library allows to control the way it loads the data using the environment variable `CUBE_DATA_LOADING`. Following values are possible:

1. **keepall** - data is loaded on demand and kept in memory to the end of lyfecycle of the Cube object.
2. **preload** - all data is loaded during the metric initialization and kept in memory to the end of lyfecycle of the Cube object.
3. **manual** - Application should request and drop the data sets explicitly. No correctness check is performed. Therefore one has to use this strategy with care.
4. **lastn** - Only `N` last used data rows are kept in memory. `N` is specified via environment variable `CUBE_NUMBER_ROWS`

1.6 Support

If you have any questions or comments you would like to share with the CUBE developers, please send an e-mail to scalasca@fz-juelich.de.

2 Bibliography

Message Passing Interface Forum: *MPI: A Message Passing Interface Standard — Version 2.2*, September, 2009, http://www.mpi-forum.org

OpenMP Architecture Review Board: *OpenMP Fortran Application Program Interface — Version 3.0*, May, 2008, http://www.openmp.org