READEX Tool Suite Installation Guide

August 1, 2018

Contents

1. About different compilers

During the development of the READEX tool suite it turned out that it is not obvious to use the same compiler for everything. Please be aware that there are some incompatibilities between different compilers. Specially for C++ this is a problem, as different compilers may have different ABI versions, which will lead to linking errors. Moreover, a Score-P version compiled for GCC will not work with the Intel compiler and vice versa.

Therefore, it is really important that you are using the same compiler for the entire READEX tool suite as well as for the applications on which you are going to use the tool suite. Again, please use the same compiler for Score-P, PTF, the RRL, the PCPs, the ATP library and your application.

2. Score-P

This section outlines how to build Score-P for READEX.

2.1. Requirements

The build procedure for the READEX version of Score-P requires the following tools to be already installed:

- Intel compiler version 2017.2.174/2018.1.163 or GCC (G++ and GFortran) version 6.3.0/7.1.0. Other Intel or GCC compiler versions can also be used, but have not been explicitly tested by the READEX developers.
- PAPI version 5.5.1.
- Bison version 3.0.4.

2.2. Download

Please download the version of Score-P for READEX from the following location and unpack it:

```
http://www.readex.eu/index.php/dissemination/software/ScoreP.tar.gz
tar -xzvf ScoreP.tar.gz
```

2.3. Preparing the Score-P directory

Please prepare the Score-P build directory as follows:

```
cd ScoreP
mkdir build
cd build
```

2.4. Configuring and installing Score-P

You may use the following naming scheme for the "--prefix" argument:

```
<Desired path for Score-P installation>/scorep/scorep_readex_1808_<mpi version>_<compiler
    version>

<mpi version>:          for example, intelmpi2017.2.174
<compiler version>:     for example, intel2017.2.174
```

To run configure please do:

```
../configure        '--prefix=<Desired_path_for_Score-P_installation>/scorep/
    scorep_readex_1808_<mpi_version>_<compiler_version>/' \
                    '--enable-backend-test-runs' \
                    '--with-nocross-compiler-suite=<gcc|intel>' \
                    '--with-mpi=<bullxmpi|intel3|...>' \
                    '--with-libcudart=<path_to_CUDA_installation>' \
                    '--with-pdt=<path_to_PDT_bin>' \
                    '--with-papi-header=<path_to_PAPI_include>' \
                    '--with-papi-lib=<path_to_PAPI_lib>' \
                    '--with-libbfd=no' \
                    '--disable-silent-rules' \
                    '--without-gui' \
                    '--enable-static' \
                    '--enable-shared' \
```

```
                        '--enable-debug'\
                        'CFLAGS=-g -O3 -fno-omit-frame-pointer' \
                        'CXXFLAGS=-g -O3 -fno-omit-frame-pointer'\
make -j
make install
```

For more details on installing Score-P, refer to Section 2.1 in the Score-P User Manual that can be downloaded from `https://silc.zih.tu-dresden.de/scorep-current.pdf`.

3. PTF

This section outlines how to build the Periscope Tuning Framework (PTF) for READEX.

3.1. Requirements

The build procedure for the READEX version of PTF requires the following tools to be already installed:

- Score-P extension for READEX as described in Section 2.
- Intel compiler version 2017.2.174/2018.1.163 or GCC (G++ and GFortran) version 6.3.0/7.1.0. Other Intel or GCC compiler versions can also be used, but have not been explicitly tested by the READEX developers.
- PAPI version 5.5.1.
- Boost version 1.62.0.
- Cereal version 1.2.1.
- Bison version 3.0.4.
- Python version 3.6 or higher.
- Ace version 6.3.3.
- Flex version 2.5.39.
- Score-P developer tools containing patched Libtool version 2.4.6, Autoconf version 2.69, Automake version 1.13.4, Doxygen version 1.8.10 and M4 version 1.4.16.

Please make sure that the Score-P version is also compiled with the same compiler as the one used for PTF.

NOTE: Please note that it may be necessary to load the compiler's environment (loading modules or adding directories/locations to environment variables) before loading the environment for the Boost library when installing PTF.

3.2. Download

Please download the READEX branch of PTF and Score-P development tools from the following locations, and unpack them:

```
http://www.readex.eu/index.php/dissemination/software/PTF.tar.gz
tar -xzvf PTF.tar.gz

http://www.readex.eu/index.php/dissemination/software/scorep-dev-06.tar.gz
tar -xzvf scorep-dev-06.tar.gz
```

3.3. Preparing the PTF directory

Please bootstrap PTF as follows:

```
cd PTF
./bootstrap
mkdir build
cd build
```

3.4. Configuring and installing PTF

Add the paths to the libraries and executables of the Score-P developer tools to the `PATH` and `LD_LIBRARY_PATH` environment variables:

```
export PATH=<path to Score-P developer tools>/bin:$PATH
export LD_LIBRARY_PATH=<path to Score-P developer tools>/lib:$LD_LIBRARY_PATH
```

You may use the following naming scheme for "--prefix":

```
<Desired path for PTF installation>/ptf/ptf_readex_1808_<mpi version>_<compiler version>
    _with_scorep/

<day of build>          build date in the form YYYY-MM-DD
<compiler version>      for example: intel2017.2.174
<mpi version>           for example: intelmpi2017.2.174
```

To configure and install PTF please now do:

```
../configure        '--prefix=<Desired_path_for_PTF_installation>/ptf/ptf_readex_1808_<mpi_
    version>_<compiler_version>_with_scorep/' \
                    --enable-developer-mode \
                    --with-starter=<slurm|superMUC> \
                    --with-ace-include=<path to ACE include> \
                    --with-ace-lib=<path to ACE lib>\
                    --with-boost-include=<path to Boost include>\
                    --with-boost-lib=<path to Boost lib>\
                    --with-cube-include=<path to Cube include> \
                    --with-cube-lib=<path to Cube lib> \
                    --with-scorep-include=<path to Score-P include> \
                    --with-scorep-lib=<path to Score-P lib> \
                    --with-cereal-include=<path to Cereal include>

make -j 24
make install
```

If you want to use the Intel compiler to compile PTF, please add the following to "../configure" :

```
    --with-compiler-suite=intel
```

Please be aware that there is no special cube module needed, if you are using the same compiler for Score-P and PTF. Therefore the options for "--with-cube-include" and "--with-cube-lib" are the same as for "--with-scorep-include" and "--with-scorep-lib", respectively.

3.4.1. Managing starter plugins

Since PTF is an online tool, the frontend starts the parallel application. To do so, it has to use the right command and parameters which are clearly depending on your batch system and your local installation. PTF comes with several starters, e.g., a generic starter named "interactive", another one for SuperMUC at LRZ and a generic one for Slurm.

PTF uses plugins for defining the start command for the application and the analysis agents. You select the plugin to be used on your machine in the configuration step.

A starter plugin defines the command to start the application. For example, on SuperMUC you use `mpiexec` while on slurm you use `srun`. Furthermore, you give the necessary parameters. For example, the slurm plugin uses a separate node for running the PTF processes while on SuperMUC the processes are assigned to the first node where your batch script is started. As said, decisions taken here are clearly dependent on your machine setup.

Please have a look at the available plugins. It might be necessary to create your own plugin using one of the available plugins as a template.

In order to create a new starter plugin on a new machine, follow these steps inside the PTF source:

1. Go to the root directory of the PTF repository.
2. Now go to the `starterplugin` folder.
3. Create a new folder with your machine specific folder name.
4. Copy your chosen `ptf-plugin.cc` template file from "interactive" folder and paste it into your created folder.
5. Now make the specific changes to start the application and the analysis agents.

Next change the configure command's `--with-starter` option with the newly created starter name (for example: `--with-starter=slurm`) during the configuration step.

In case you need help in developing your own starter, please contact the PTF support at `periscope@lists.lrz.de`.

For more details on installing PTF, refer to Section 2.3 in the PTF Installation Guide which is available for download at `http://periscope.in.tum.de/releases/latest/pdf/PTF_Installation_Guide.pdf`.

4. RRL

This section outlines how to build the READEX Runtime Library (RRL).

4.1. Requirements

The build procedure for RRL requires the following tools to be already installed:

- Score-P extension for READEX as described in Section 2.
- Intel compiler version 2017.2.174/2018.1.163 or GCC (G++ and GFortran) version 6.3.0/7.1.0. Other Intel or GCC compiler versions can also be used, but have not been explicitly tested by the READEX developers.
- CMake version 3.11 or higher.
- PAPI version 5.5.1.

Please make sure that the Score-P version is also compiled with the same compiler as the one used for RRL.

4.2. Download

Please download RRL from the following location and unpack it:

```
http://www.readex.eu/index.php/dissemination/software/RRL.tar.gz
tar -xzvf RRL.tar.gz
```

4.3. Preparing the RRL directory

Please switch to the RRL directory of the downloaded repository:

```
cd RRL
mkdir build
cd build
```

4.4. Configuring and installing the RRL

You may use the following naming scheme for "--DCMAKE_INSTALL_PREFIX":

```
<Desired path for RRL installation>/readex-rrl/rrl-readex-<mpi version>-<compiler version>

<day of build>          build date in the form YYYY-MM-DD
<compiler version>      for example: intel2017.2.174
<mpi version>           for example: intelmpi2017.2.174
```

To build the RRL, you will need x86_adapt (`https://github.com/tud-zih-energy/x86_adapt`). Please be sure to have the path to `libx86_adapt.so` in your `LD_LIBRARY_PATH`.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path to libx86_adapt.so>
```

Now, to build the RRL please do:

```
cmake ../ -DCMAKE_INSTALL_PREFIX=<Desired path for RRL installation>/readex-rrl/
      rrl_readex_1808_<mpi version>_<compiler version> -DDISABLE_CALIBRATION=ON
make -j 24
make install
```

NOTE: You may omit x86_adapt and PAPI from the build process if you choose to use RRL without the calibration feature.

5. PCPs

This section outlines how to build the different Parameter Control Plugins (PCPs) for READEX.

5.1. Requirements

The build procedure for the PCPs requires the following tools to be already installed:

- READEX Runtime Library (RRL) as described in Section 4.
- Intel compiler version 2017.2.174/2018.1.163 or GCC (G++ and GFortran) version 6.3.0/7.1.0. Other Intel or GCC compiler versions can also be used, but have not been explicitly tested by the READEX developers.

Please make sure that the RRL version is also compiled with the same compiler as the one used for the PCPs.

5.2. Download

Please download the PCP from the following location and unpack it:

```
http://www.readex.eu/index.php/dissemination/software/PCPs.tar.gz
tar -xzvf PCPs.tar.gz
```

5.3. Configuring and installing the PCP's

You may use the following naming scheme for the argument of the build script:

```
<Desired path for PCPs installation>/parameter_control_plugins/pcp_readex_1808_<mpi
    version>_<compiler version>

<day of build>          build date in the form YYYY-MM-DD
<compiler version>      for example: intel2017.2.174
```

To build the PCPs please now do:

```
cd PCPs
./build.sh <Desired path for PCPs installation>/parameter_control_plugins/pcp_readex_1808_<
    mpi version>_<compiler version>
```

6. ATP

This section outlines how to build the Application Tuning Parameter library (ATP library) for READEX.

6.1. Requirements

The build procedure for the ATP library requires the following tools to be already installed:

- READEX Runtime Library (RRL) as described in Section 4.
- Intel compiler version 2017.2.174/2018.1.163 or GCC (G++ and GFortran) version 6.3.0/7.1.0. Other Intel or GCC compiler versions can also be used, but have not been explicitly tested by the READEX developers.
- CMake version 3.11 or higher.

Please make sure that the RRL version is also compiled with the same compiler as the one used for the ATP library.

6.2. Download

Please download ATP library from the following location and unpack it:

```
http://www.readex.eu/index.php/dissemination/software/ATP.tar.gz
tar -xzvf ATP.tar.gz
```

6.3. Preparing the ATP library directory

Please do:

```
cd ATP
mkdir build
cd build
```

6.4. Configuring and installing the ATP library

You may use the following naming scheme for "-DCMAKE_INSTALL_PREFIX":

```
<Desired path for ATP library installation>/readex-atp/atp_readex_1808_<mpi version>_<
    compiler version>

<day of build>          current date in the form YYYY-MM-DD
<compiler version>      for example: intel2017.2.174
<mpi version>           for example: intelmpi2017.2.174
```

To build the ATP please now do:

```
cmake ../ -DCMAKE_INSTALL_PREFIX=<Desired path for ATP library installation>/readex-atp/
    atp_readex_1808_<mpi version>_<compiler version>
make -j 24
make install
```

7. Cluster Prediction

This section outlines how to build the Cluster Prediction library for READEX.

7.1. Requirements

The build procedure for the Cluster Prediction library requires the following tools to be already installed:

- READEX Runtime Library (RRL) as described in Section 4.
- Intel compiler version 2017.2.174/2018.1.163 or GCC (G++ and GFortran) version 6.3.0/7.1.0. Other Intel or GCC compiler versions can also be used, but have not been explicitly tested by the READEX developers.
- CMake version 3.11 or higher.
- PAPI version 5.5.1 or higher.

Please make sure that the RRL version is also compiled with the same compiler as the one used for the Cluster Prediction library.

7.2. Download

Please download Cluster Prediction library from the following location and unpack it:

```
http://www.readex.eu/index.php/dissemination/software/Cluster_Prediction.tar.gz
tar -xzvf Cluster_Prediction.tar.gz
```

7.3. Preparing the Cluster Prediction library directory

Please do:

```
cd Cluster_Prediction
mkdir build
cd build
```

7.4. Configuring and installing the Cluster Prediction library

You may use the following naming scheme for "-DCMAKE_INSTALL_PREFIX":

```
<Desired path for Cluster Prediction library installation>/cluster_prediction/
      cluster_prediction_readex_1808_<mpi version>_<compiler version>

<day of build>   current date in the form YYYY-MM-DD
<compiler version>   for example: intel2017.2.174
<mpi version>    for example: intelmpi2017.2.174
```

To build the Cluster Prediction library please now do:

```
cmake ../ -DCMAKE_INSTALL_PREFIX=<Desired path for Cluster Prediction library installation
     >/cluster_prediction/cluster_prediction_readex_1808_<mpi version>_<compiler version>
make
make install
```

A. READEX tool suite installation on Taurus (TU Dresden)

A.1. The module environment

Taurus uses so called modules to allow a dynamic environment. This avoids setting different environment variables by hand. It is therefore recommended to use these modules instead of manually setting environment variables. To use READEX related modules please do:

```
module use /projects/p_readex/modules/
```

It is assumed throughout this document that the modules directory is loaded.

A.2. Allocating a node

Please allocate a node to build the READEX Tool Suite to keep the login server free. You can do so by running:

```
srun -p haswell --time=4:00:00 --pty --exclusive -n 1 -c 24 \
    --mem=60000 -A p_readex bash -l -i
```

A.3. Score-P

To build Score-P please follow these instructions.

A.3.1. Modules

If you have previously built anything else please do

```
module use /projects/p_readex/modules/
```

Please load the following modules to build Score-P using gcc/5.3.0 and Bull XMPI:

```
module load \
    svn/1.9.3 \
    pdt/3.18.1 \
    cuda/7.5.18 \
    papi/5.4.3 \
    bison/3.0.4 \
    scorep-dev/05 \
    gcc/6.3.0 \
    bullxmpi/1.2.8.4
```

If you would like to use the Intel compiler and Intel MPI instead please load:

```
module load \
    pdt/3.18.1 \
    cuda/7.5.18 \
    papi/5.4.3 \
    bison/3.0.4 \
    scorep-dev/05 \
    intel/2016.2.181 \
    intelmpi/5.1.3.181
```

A.3.2. Download

Please Download Score-P from the following location, using your Score-P access details:

```
svn co https://silc.zih.tu-dresden.de/svn/silc-root/branches/
      TRY_READEX_online_access_call_tree_extensions
```

A.3.3. Preparing the Score-P directory

Next please do:

```
cd TRY_READEX_online_access_call_tree_extensions
./bootstrap
mkdir build
cd build
```

A.3.4. Configuring and installing Score-P

You can use the p_readex directory to install your Score-P version. In order to keep an overview about the different versions, please stick to the following naming scheme for the "--prefix" argument:

```
/projects/p_readex/scorep/TRY_READEX_online_access_call_tree_extensions_r<svn revision
      number>_<mpi version>_<compiler verison>

<svn revision number>:  for example, 11271
<mpi version>:          for example, bullxmpi
<compiler version>:     for example, gcc5.3.0
```

To run configure please do:

```
../configure        '--prefix=/projects/p_readex/scorep/
      TRY_READEX_online_access_call_tree_extensions_r<scorep_revision_number>_<mpi_version>_
      <compiler_version>/' \
                                '--enable-backend-test-runs' \
                                '--with-nocross-compiler-suite=gcc' \
                                '--with-mpi=bullxmpi' \
                                '--with-libcudart=/sw/taurus/libraries/cuda/7.5.18' \
                                '--disable-silent-rules' \
                                '--with-pdt=/sw/global/tools/pdt/3.18.1/x86_64/bin' \
                                '--with-libbfd=no' \
                                '--with-papi-header=/sw/taurus/libraries/papi/5.4.3/include
                                ' \
                                '--with-papi-lib=/sw/taurus/libraries/papi/5.4.3/lib' \
                                '--with-machine-name=taurus.hrsk.tu-dresden.de' \
                                '--without-gui' \
                                '--enable-static' \
                                '--enable-shared' \
                                '--enable-debug'\
                                'CFLAGS=-g -O0' \
                                'CXXFLAGS=-g -O0'\
make -j 24
make install
```

For Intel compiler and Intel MPI please change the following flags to:

```
                                '--with-nocross-compiler-suite=intel' \
                                '--with-mpi=intel3' \
```

A.3.5. Creating the related Score-P module file

Please build a module file in order to use your Score-P version, and make it accessible for others. Please change to the Score-P module directory:

```
cd /projects/p_readex/modules/scorep
```

Please create there a file with the following name:

```
TRY_READEX_online_access_call_tree_extensions_r<scorep_revision_number>_<mpi_version>_<
    compiler_version>
```

And insert the following content

```
#%Module10######################################################################
##
## This is a modules template. Adapt it to your requirements.
## Module file for <software xyz>
##

## Source zih-modules helper script. It provides the framework for an uniform modules
    system.
## Additionally, it sets global variables that provide you information about the
## application to be loaded and some other information. See the following list:
##
## soft_arch       Provides information about the architecture.
## soft_class      Provides the software class. E.g. [applications|compiler|tools]
## soft_host       Provides the host name.
## soft_machine    Provides the machine name.
## soft_version    Provides the software version number to be loaded.
## soft_ware       Provides the software name to be loaded.
## user            Provides the user name.

source /sw/modules/global/modulescripts/zih_modules.tcl

set scorep_root "/projects/p_readex/scorep/$soft_version"

## Set modules dependencies with version information !!! e.g. intel/11.1.069

#set soft_dependencies "_"

append soft_dependencies "pdt/3.18.1_cuda/7.5.18_papi/5.4.3_<your_mpi_verison>_<your_
    compiler_version>"

## Set variable shortDescription if you want to add specific information that
## should be displayed while the module is loaded.

#set shortDescription "Use_\"bsub_-n<number-of-cpus>_...\"__to_start_the_application_xyz"


## Set variable longDescription if you want to add further information about the
## software that should be displayed at the module help command.

set longDescription "This_$soft_ware_provides_a_highly_scalable_measurement_infrastructure_
    to_trace_your_parallel_applications."

append longDescription "\nCompile_with_scorep_<usual_compiler_command>."


## Set the specific environment variables for your software package

#set lib_path /sw/<global|$soft_machine>/$soft_class/$soft_ware/$soft_version/$soft_arch

#environment settings for Score-P
setenv SCOREP_ROOT $scorep_root
setenv SCOREP_INC  $scorep_root/include
setenv SCOREP_LIB  $scorep_root/lib

prepend-path LD_LIBRARY_PATH $scorep_root/lib
prepend-path PATH $scorep_root/bin


## Source modules action information
source /sw/modules/global/modulescripts/zih_modules_action.tcl
```

where

```
<your mpi verison>     : for example, bullxmpi
<your compiler version>: for example, gcc/5.3.0
```

A.4. Using Score-P

Now you can use Score-P by doing:

```
module load scorep/TRY_READEX_online_access_call_tree_extensions_r<scorep_revision_number>
    _<mpi_version>_<compiler_version>
```

A.5. PTF

This section outlines how to build and use PTF.

A.5.1. Modules

If you have previously built anything else please do

```
module purge
```

Now load the modules to build PTF with gcc:

```
module load \
    papi/5.4.3 \
    boost/1.60.0-gnu5.3 \
    automake/1.14 \
    m4/1.4.16 \
    python/2.7 \
    autoconf/2.69 \
    libtool/2.4.2 \
    autotools/2015 \
    ace/6.3.3 \
    libunwind/1.1 \
    git \
    gcc/5.3.0  \
    flex/2.5.39 \
    bison/3.0.4 \
    cereal/1.2.1

module load scorep/TRY-READEX-online-access-call-tree-extensions-r<scorep revision number>-
    <mpi version>-<compiler version>
```

where

```
<svn revision number>:  for example: 11271
<mpi version>:          for example: bullxmpi
<compiler version>:     for example: gcc5.3.0
```

For Intel please do:

```
module load \
    papi/5.4.3 \
    boost/1.60.0-gnu5.3 \
    automake/1.14 \
    m4/1.4.16 \
    python/2.7 \
    autoconf/2.69 \
    libtool/2.4.2 \
    autotools/2015 \
    ace/6.3.3 \
    libunwind/1.1 \
    git \
    intel/2016.2.181 \
    flex/2.5.39 \
    bison/3.0.4 \
    cereal/1.2.1 \

module load scorep/TRY-READEX-online-access-call-tree-extensions-r<scorep revision number>-
    <mpi version>-<compiler version>
```

Please be sure that the Score-P version is also compiled with the Intel compiler.

A.5.2. Download

Please download the READEX branch of PTF:

```
git -c http.sslVerify=false clone https://periscope.in.tum.de/git/Periscope.git
cd Periscope
git checkout readex
```

A.5.3. Preparing the PTF directory

Please do:

```
./bootstrap
mkdir build
cd build
```

A.5.4. Configuring and installing PTF

Again you can use the p_readex project directory to install PTF. Please stick to the following naming scheme for "--prefix":

```
/projects/p_readex/ptf/ptf-<day of build>-readex-<compiler version>-slurm_starter-<mpi
    version>-with-scorep/

<day of build>          actual date in the form YYYY-MM-DD
<compiler version>      for example: gcc5.3
<mpi version>           for example: bullxmpi
```

To configure and install PTF please now do:

```
../configure         '--prefix=/projects/p_readex/ptf/ptf-<day_of_build>-readex-<compiler_
    version>-slurm_starter-<mpi_version>-with-scorep/' \
                     --enable-developer-mode \
                     --with-starter=slurm \
                     --enable-ace \
                     --enable-boost \
                     --with-boost-lib=$BOOST_LIB\
                     --enable-cube \
                     --with-cube-include=$SCOREP_INC \
                     --with-cube-lib=$SCOREP_LIB \
                     --enable-scorep \
                     --with-scorep-include=$SCOREP_INC \
                     --with-scorep-lib=$SCOREP_LIB \
                     --enable-cereal \
                     --with-cereal-include=$CEREAL_INC

make -j 24
make install
```

If you want to use the Intel compiler to compile PTF, please add the following to "../configure" :

```
    --with-compiler-suite=intel
```

Please be aware that there is no special cube module needed, if you are using the same compiler for Score-P and PTF. Therefore the options for "--with-cube-include" and "--with-cube-lib" are the same as for "--with-scorep-include" and "--with-scorep-lib".

A.5.5. Creating the related PTF module file

Please build a module file in order to use your PTF version, and make it accessible for others. Please change to the PTF module directory:

```
cd /projects/p_readex/modules/ptf
```

Please create there a file with the following name:

```
ptf-<day of build>-readex-<compiler version>-slurm_starter-<mpi version>-with-scorep
```

And insert the following content

```
#%Module10#####################################################################
##
## This is a modules template. Adapt it to your requirements.
## Module file for <software xyz>
##

## Source zih-modules helper script. It provides the framework for an uniform modules
     system.
## Additionally, it sets global variables that provide you information about the
## application to be loaded and some other information. See the following list:
##
## soft_arch        Provides information about the architecture.
## soft_class       Provides the software class. E.g. [applications|compiler|tools]
## soft_host        Provides the host name.
## soft_machine     Provides the machine name.
## soft_version     Provides the software version number to be loaded.
## soft_ware        Provides the software name to be loaded.
## user             Provides the user name.

source /sw/modules/global/modulescripts/zih_modules.tcl

set ptf_root "/projects/p_readex/ptf/$soft_version"

## Set modules dependencies with version information !!! e.g. intel/11.1.069

#set soft_dependencies "_"

append soft_dependencies "<mpi_version>_boost/1.60.0-gnu5.3_<compiler_version>_ace/6.3.3_
     libunwind/1.1"


## Set variable shortDescription if you want to add specific information that
## should be displayed while the module is loaded.

#set shortDescription "Use_\"bsub_-n<number-of-cpus>_...\"__to_start_the_application_xyz"


## Set variable longDescription if you want to add further information about the
## software that should be displayed at the module help command.

#set longDescription "This_$soft_ware_provides_a_highly_scalable_measurement_infrastructure
     _to_trace_your_parallel_applications."

#append longDescription "\nCompile_with_scorep_<usual_compiler_command>."


## Set the specific environment variables for your software package

#set lib_path /sw/<global|$soft_machine>/$soft_class/$soft_ware/$soft_version/$soft_arch

#environment settings for Score-P
setenv PTF_ROOT $ptf_root
setenv PTF_INC  $ptf_root/include
setenv PTF_LIB  $ptf_root/lib

prepend-path LD_LIBRARY_PATH $ptf_root/lib
prepend-path PATH $ptf_root/bin


## Source modules action information
source /sw/modules/global/modulescripts/zih_modules_action.tcl
```

where

```
<compiler version>      for example: gcc/5.3.0
<mpi version>           for example: bullxmpi
```

A.5.6. Using PTF

Now you can use PTF by doing:

```
module load ptf/ptf-<day of build>-readex-<compiler version>-slurm_starter-<mpi version>-
     with-scorep
```

A.6. RRL

This Section outlines how to build and use RRL.

A.6.1. Modules

If you have previously built anything else please do

```
module purge
```

Now load the modules to build the RRL:

```
module load modenv/both
module load scorep/TRY_READEX_online_access_call_tree_extensions_r<scorep revision number>_
    <mpi version>_<compiler version>
module load cmake/<3.1 or later>
module load papi/5.5.1
module load protobuf/3.5.0-intel2017.2.174
module load TensorFlow/1.6.0-intel-2018a-Python-3.6.4
```

where

```
<svn revision number>:  for example: 11271
<mpi version>:          for example: bullxmpi
<compiler version>:     for example: gcc5.3.0
```

A.6.2. Download

Please download the READEX RRL git:

```
git clone --recursive git@gitlab.hrz.tu-chemnitz.de:READEX/RRL.git
```

You'll need to be registered at the TU Chemnitz gitlab. If you are not, please log in into `https://gitlab.hrz.tu-chemnitz.de/` and place your public ssh key there. Please have a look at `https://docs.gitlab.com/ce/ssh/README.html` for details about ssh keys.

A.6.3. Preparing the RRL directory

Please do:

```
cd readex-rrl/TUD_RRL/
mkdir build
cd build
```

A.6.4. Configuring and installing the RRL

Again you can use the p_readex project directory to install RRL. Please stick to the following naming scheme for "–DCMAKE_INSTALL_PREFIX":

```
/projects/p_readex/readex-rrl/rrl-<day of build>-<compiler version>-<mpi version>

<day of build>          actual date in the form YYYY-MM-DD
<compiler version>      for example: gcc5.3.0
<mpi version>           for example: bullxmpi
```

To build the RRL you need x86_adapt (`https://github.com/tud-zih-energy/x86_adapt`). Please be sure to have the libx86_adapt.so in your "LD_LIBRARY_PATH". To do so on Taurus please do:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

Now, to build the RRL please now do:

```
export CPATH=/sw/taurus/eb/TensorFlow/1.6.0-intel-2018a-Python-3.6.4/include:$CPATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib:/sw/taurus/eb/TensorFlow/1.6.0-intel
      -2018a-Python-3.6.4/lib/python3.6/site-packages/tensorflow/../_solib_k8/
      _U_S_Sthird_Uparty_Smkl_Cintel_Ubinary_Ublob___Uexternal_Smkl_Slib/

cmake ../ -DCMAKE_INSTALL_PREFIX=/projects/p_readex/readex-rrl/rrl-<day of build>-<compiler
       version>-<mpi version> -DEXTERN_TENSORFLOW=ON -DEXTERN_PROTOBUF=ON
make -j24
make install
```

A.6.5. Creating the related RRL module file

Please build a module file in order to use your RRL version, and make it accessible for others. Please change to the RRL module directory:

```
cd /projects/p_readex/modules/readex-rrl
```

Please create there a file with the following name:

```
rrl-<day of build>-<compiler version>-<mpi version>
```

And insert the following content

```
#%Module10#####################################################################
##
## This is a modules template. Adapt it to your requirements.
## Module file for <software xyz>
##

## Source zih-modules helper script. It provides the framework for an uniform modules
     system.
## Additionally, it sets global variables that provide you information about the
## application to be loaded and some other information. See the following list:
##
## soft_arch       Provides information about the architecture.
## soft_class      Provides the software class. E.g. [applications|compiler|tools]
## soft_host       Provides the host name.
## soft_machine    Provides the machine name.
## soft_version    Provides the software version number to be loaded.
## soft_ware       Provides the software name to be loaded.
## user            Provides the user name.

source /sw/modules/global/modulescripts/zih_modules.tcl

set rrl_root "/projects/p_readex/readex-rrl/$soft_version"

## Set modules dependencies with version information !!! e.g. intel/11.1.069

#set soft_dependencies

append soft_dependencies "<your_mpi_verison>_<your_compiler_version>"

## Set variable shortDescription if you want to add specific information that
## should be displayed while the module is loaded.

#set shortDescription "Use_\"bsub_-n<number-of-cpus>_...\"__to_start_the_application_xyz"


## Set variable longDescription if you want to add further information about the
## software that should be displayed at the module help command.

set longDescription "TODO"

append longDescription "\nTODO"


## Set the specific environment variables for your software package

#set lib_path /sw/<global|$soft_machine>/$soft_class/$soft_ware/$soft_version/$soft_arch

#environment settings for Score-P
setenv RRL_ROOT $rrl_root
setenv RRL_INC  $rrl_root/include
setenv RRL_LIB  $rrl_root/lib

prepend-path LD_LIBRARY_PATH $rrl_root/lib
prepend-path PATH $rrl_root/bin
prepend-path CPAHT $rrl_root/bin/include
```

```
## Source modules action information
source /sw/modules/global/modulescripts/zih_modules_action.tcl
```

where

```
<scorep revision number>      : revision number of the scorep version you used
<mpi version>                 : mpi version of the scorep version you used
<compiler version>            : compiler of the scorep version you used
```

A.6.6. Using RRL

Now you can use RRL by doing:

```
module load readex-rrl/rrl-<day of build>-<compiler version>-<mpi version>
export SCOREP_SUBSTRATE_PLUGINS='rrl'
export SCOREP_RRL_VERBOSE="DEBUG"

export SCOREP_TUNING_PLUGINS='OpenMPTP,...'
<or>
export SCOREP_RRL_PLUGINS='OpenMPTP,...'

#optional, if a tuning model is present
#export SCOREP_RRL_TMM_PATH="/path/to/the/tuning/model.json"
```

A.6.7. Building doc

Please go to your build folder and do

```
module load doxygen/1.8.11
make doc
```

Please take also a look at the README.

A.7. PCP's

This Section outlines how to build the different Parameter Control Plugins.

A.7.1. Modules

If you have previously built anything else please do

```
module purge
```

Now load the modules to build the RRL:

```
module load readex-rrl/rrl-<day of build>-<compiler version>-<mpi version>
```

Where is:

```
<day of build>:          for example: 2016-10-11
<mpi version>:           for example: bullxmpi
<compiler version>:      for example: gcc5.3.0
```

A.7.2. Download

Please download the PCP git:

```
git clone git@gitlab.hrz.tu-chemnitz.de:READEX/PCPs.git
```

A.7.3. Configuring and installing the PCP's

Again you can use the p_readex project directory to install the PCP's. Please stick to the following naming scheme for the second argument of the build script:

```
 /projects/p_readex/parameter_control_plugins/pcp_<day of build>-<compiler version>

 <day of build>          actual date in the form YYYY-MM-DD
 <compiler version>      for example: gcc5.3.0
```

To build the PCP's please now do:

```
cd PCPs
./build.sh /projects/p_readex/parameter_control_plugins/pcp_<day of build>-<compiler
     version>
```

A.7.4. Creating the related PCP's module file

Please build a module file in order to use your PCP version, and make it accessible for others. Please change to the PCP module directory:

```
cd /projects/p_readex/modules/pcp/
```

Please create there a file with the following name:

```
pcp_<day of build>-<compiler version>
```

And insert the following content

```
#%Module10#####################################################################
##
## This is a modules template. Adapt it to your requirements.
## Module file for <software xyz>
##

## Source zih-modules helper script. It provides the framework for an uniform modules
     system.
## Additionally, it sets global variables that provide you information about the
## application to be loaded and some other information. See the following list:
##
## soft_arch       Provides information about the architecture.
## soft_class      Provides the software class. E.g. [applications|compiler|tools]
## soft_host       Provides the host name.
## soft_machine    Provides the machine name.
## soft_version    Provides the software version number to be loaded.
## soft_ware       Provides the software name to be loaded.
## user            Provides the user name.

source /sw/modules/global/modulescripts/zih_modules.tcl

set pcp_root "/projects/p_readex/parameter_control_plugins/pcp_<day_of_build>-<compiler_
     version>"

## Set modules dependencies with version information !!! e.g. intel/11.1.069

#set soft_dependencies ""

append soft_dependencies "<your_mpi_verison>_<your_compiler_version>"

## Set variable shortDescription if you want to add specific information that
## should be displayed while the module is loaded.

#set shortDescription "Use_\"bsub_-n<number-of-cpus>_...\"__to_start_the_application_xyz"


## Set variable longDescription if you want to add further information about the
## software that should be displayed at the module help command.

set longDescription "TODO"

append longDescription "\nTODO"


## Set the specific environment variables for your software package
```

```
#set lib_path /sw/<global|$soft_machine>/$soft_class/$soft_ware/$soft_version/$soft_arch

#environment settings for Score-P
setenv PCP_ROOT $pcp_root
setenv PCP_LIB  $pcp_root/lib

prepend-path LD_LIBRARY_PATH $pcp_root/lib

## Source modules action information
source /sw/modules/global/modulescripts/zih_modules_action.tcl
```

where

```
<day of build>:        for example: 2016-10-11
<mpi version>:         for example: bullxmpi
<compiler version>:    for example: gcc5.3.0
```

A.7.5. Using PCP's

Now you can use the PCP's by adding them to the SCOREP_TUNING_PLUGINS or SCOREP_RRL_PLUGINS:

```
module load pcp/pcp_<day of build>-<compiler version>

export SCOREP_TUNING_PLUGINS='OpenMPTP,cpu_freq_plugin,epb_plugin,uncore_freq_plugin,...'
<or>
export SCOREP_RRL_PLUGINS='OpenMPTP,cpu_freq_plugin,epb_plugin,uncore_freq_plugin,...'
```

Please take also a look to the README files.

A.8. ATP

This section outlines how to build and use the ATP library.

A.8.1. Modules

If you have previously built anything else please do

```
module purge
```

Now load the modules to build the ATP library:

```
module load readex-rrl/rrl-<day of build>-<compiler version>-<mpi version>
module load cmake/<3.1 or later>
```

where

```
<svn revision number>:  for example: 11271
<mpi version>:          for example: bullxmpi
<compiler version>:     for example: gcc5.3.0
```

A.8.2. Download

Please download the ATP git:

```
git clone git@gitlab.hrz.tu-chemnitz.de:READEX/ATP.git
```

You'll need to be registered at the TUC gitlab. If you are not, please log in into `https://gitlab.hrz.tu-chemnitz.de/` and place your public ssh key there. Please have a look at `https://docs.gitlab.com/ce/ssh/README.html` for details about ssh keys.

A.8.3. Preparing the ATP directory

Please do:

```
cd ATP/
mkdir build
cd build
```

A.8.4. Configuring and installing the ATP

Again you can use the p_readex project directory to install ATP. Please stick to the following naming scheme for "–DCMAKE_INSTALL_PREFIX":

```
 /projects/p_readex/readex-atp/atp-<day of build>-<compiler version>-<mpi version>

 <day of build>          actual date in the form YYYY-MM-DD
 <compiler version>      for example: gcc5.3.0
 <mpi version>           for example: bullxmpi
```

To build the ATP please now do:

```
cmake ../ -DCMAKE_INSTALL_PREFIX=/projects/p_readex/readex-atp/atp-<day of build>-<compiler
      version>-<mpi version>
make -j24
make install
```

A.8.5. Creating the related ATP module file

Please build a module file in order to use your ATP version, and make it accessible for others. Please change to the ATP module directory:

```
cd /projects/p_readex/modules/readex-atp
```

Pleas create there a file with the following name:

```
atp-<day of build>-<compiler version>-<mpi version>
```

And insert the following content

```
#%Module10######################################################################
##
## This is a modules template. Adapt it to your requirements.
## Module file for <software xyz>
##

## Source zih-modules helper script. It provides the framework for an uniform modules
    system.
## Additionally, it sets global variables that provide you information about the
## application to be loaded and some other information. See the following list:
##
## soft_arch       Provides information about the architecture.
## soft_class      Provides the software class. E.g. [applications|compiler|tools]
## soft_host       Provides the host name.
## soft_machine    Provides the machine name.
## soft_version    Provides the software version number to be loaded.
## soft_ware       Provides the software name to be loaded.
## user            Provides the user name.

source /sw/modules/global/modulescripts/zih_modules.tcl

set atp_root "/projects/p_readex/readex-atp/$soft_version"

## Set modules dependencies with version information !!! e.g. intel/11.1.069

#set soft_dependencies

append soft_dependencies "readex-rrl/rrl-2017-08-22-gcc6.3.0-bullxmpi"

## Set variable shortDescription if you want to add specific information that
## should be displayed while the module is loaded.
```

```
#set shortDescription "Use \"bsub -n<number-of-cpus> ...\" to start the application xyz"


## Set variable longDescription if you want to add further information about the
## software that should be displayed at the module help command.

set longDescription "TODO"

append longDescription "\nTODO"


## Set the specific environment variables for your software package

#set lib_path /sw/<global|$soft_machine>/$soft_class/$soft_ware/$soft_version/$soft_arch

#environment settings for ATP
setenv ATP_PATH $atp_root/bin

prepend-path LD_LIBRARY_PATH $atp_root/lib
prepend-path PATH $atp_root/bin
prepend-path CPATH $atp_root/include


## Source modules action information
source /sw/modules/global/modulescripts/zih_modules_action.tcl
```

where

```
  <scorep revision number>      : revision number of the scorep version you used
  <mpi version>                 : mpi version of the scorep version you used
  <compiler version>            : compiler of the scorep version you used
```

A.8.6. Using ATP

Now you can use ATP by doing:

```
module load readex-atp/atp-<day of build>-<compiler version>-<mpi version>
```

Please take also a look at the README file.