How to use the READEX Test Suite

September 25, 2018

Contents

1 Introduction **2**

2 The readex-apps repository **2**

3 Example analysis for AMG2013 **4**

1 Introduction

This document explains how to use the READEX test suite of applications instrumented by the continuously integrated READEX tool suite version on the Taurus supercomputer. For better understanding of the READEX tool suite we refer reader to the documents

- how_to_build_readex_toolsuite.pdf,
- how_to_use_readex_toolsuite.pdf.

2 The readex-apps repository

Currently, the repository consists of several benchmark and production applications listed in Table 1. Apart from the source files, each application's directory contains bash scripts to compile the application in several configurations. Namely, this includes the compilation of the uninstrumented (plain) version used as a reference and compilation for every tool in the READEX tool suite (scorep-autofilter, readex-dyn-detect, PTF, and RRL). The compilation scripts have been prepared both for manual and automatic instrumentation by Score-P (All the applications are manually intrumented, which is inserted in to the code on demand during compilation.). While the former approach serves for the evaluation of possible savings, the latter can be used to evaluate the automated READEX tool suite in comparison to the manual effort. In addition to the compilation scripts the repository contains scripts to launch the individual READEX tools and a brief `READEX_README.txt` help file.

application	**path**	**maintained by**
AMG2013	benchmark_apps/amg2013	IT4I
Blasbench	benchmark_apps/blasbench	IT4I
Kripke	benchmark_apps/kripke	IT4I
Lulesh	benchmark_apps/lulesh	ICHEC,TUM
NPB3.3	benchmark_apps/NPB3.3-MZ-MPI	TUD
MiniMD	benchmark_apps/miniMD	ICHEC
Elmer	benchmark_apps/elmerfem	IT4I
BEM4I	production_apps/BEM4I	IT4I
ESPRESO	production_apps/ESPRESO	IT4I
INDEED	production_apps/INDEED	GNS
OpenFOAM	production_apps/OPENFOAM	IT4I

Table 1: List of applications in the readex-apps repository.

To set up the environment source script files from the `readex_env` directory. On Taurus, `readex_env` is a symbolic link to a directory in `readex-apps/env` and allows for easy switching among different compilation environments (Intel, GNU). This approach ensures that the application uses the current state of the continuously integrated READEX tool suite and in addition provides a rather easy way for porting the test suite for another cluster equipped with the tool suite – the user has to adapt the sourced files, update the symbolic link to a preferred location, and use the provided compilation and run scripts without significant changes (the run scripts expect to be launched by the SLURM scheduler).

To summarize, the repository contains the files listed in Tables 2, 3.

file	**description**
set_env_cxx.source	general environment for the compiler
set_env_plain.source	environment for the uninstrumented version
set_env_saf.source	environment for scorep-autofilter
set_env_rdd.source	environment for readex-dyn-detect
set_env_ptf_hdeem.source	environment for PTF with HDEEM
set_env_ptf_rapl.source	environment for PTF with RAPL
set_env_rrl.source	environment for RRL

Table 2: Summary of environment files available in `readex-apps/env`.

file	**description**
READEX_README.txt	a help file
set_env_XXX.source	application specific environment
compile_for_plain.sh	compilation of the uninstrumented version
compile_for_saf.sh	compilation for scorep-autofilter
compile_for_rdd.sh	compilation for readex-dyn-detect, automatic instr.
compile_for_rdd_manual.sh	compilation for readex-dyn-detect, manual instr.
compile_for_ptf.sh	compilation for PTF and RRL, automatic instr.
compile_for_ptf_manual.sh	compilation for PTF and RRL, manual instr.
run_plain.sh	runs the uninstrumented version
run_saf.sh	runs scorep-autofilter
run_rdd.sh	runs readex-dyn-detect
extend_readex_config.sh	extends readex_config.xml produced by readex-dyn-detect
run_ptf.sh	runs PTF
run_rrl.sh	runs the application with RRL

Table 3: Summary of files specific for each application.

3 Example analysis for AMG2013

The AMG2013 benchmark is located at

`readex-apps/benchmark_apps/amg2013`

To apply READEX tool suite on AMG2013 follow the steps listed below.

1. Update the symbolic link `readex_env` to point to the environment of your choice. The current options on Taurus are
   - `ln -sfnv ../../env/bullxmpi1.2.8.4_gcc6.3.0/ readex_env`
   - `ln -sfnv ../../env/intelmpi2017.2.174_intel2017.2.174/ readex_env`
2. Compile the uninstrumented version and perform a testing run by
   - `./compile_for_plain.sh` (the executable created is `./test/amg2013_plain`),
   - `sbatch ./run_plain.sh`
3. Compile the code for `scorep-autofilter` and run the filtering by
   - `./compile_for_saf.sh` (the executable created is `./test/amg2013_saf`),
   - `sbatch ./run_saf.sh` (output in `scorep.filt`).

   Note that this step is only relevant for automatic instrumentation. In case of manual instrumentation only the manually inserted regions are taken into account and no filtering is necessary.
4. Compile the code for `readex-dyn-detect` and run the dynamism detection tool by
   - `./compile_for_rdd.sh` for automatic instrumentation by Score-P taking into account the filter file produced above or `./compile_for_rdd_manual.sh` for a manually instrumented version (the executable created is `./test/amg2013_rdd`),
   - `sbatch ./run_rdd.sh` (output in `readex_config.xml`).
5. The output `readex_config.xml` contains the description of significant regions for PTF. It is up to the user to specify the tuning parameters, the tuning strategy and further details described in `how_to_use_readex_toolsuite.pdf`. A script for automatic extension of the configuration file is provided for each app in the test suite and can be called by
   - `./extend_readex_config.sh` (output in `readex_config_ptf.xml`).
6. Compile the code for `PTF` and run the analysis by
   - `./compile_for_ptf.sh` for automatic instrumentation by Score-P taking into account the filter file produced above or `./compile_for_ptf_manual.sh` for a manually instrumented version (the executable created is `./test/amg2013_ptf`),

amg2013_rrl	1	4	8	2	12	109429	22456.414
amg2013_rrl	2	4	8	2	12	107916	22354.112
amg2013_rrl	3	4	8	2	12	107463	22471.416

Table 4: Energy measurements stored in amg2013_rrl_plain_hdeem.out.

amg2013_rrl	1	4	8	2	12	126788	20952.191
amg2013_rrl	2	4	8	2	12	126162	20926.660
amg2013_rrl	3	4	8	2	12	126297	20907.157

Table 5: Energy measurements stored in amg2013_rrl_rrl_hdeem.out.

- `sbatch ./run_ptf.sh` (output in `tuning_model.json`).

7. Since the compilation for RRL is the same as for PTF, the binary `./test/amg2013_ptf` can be reused to test dynamic switching according to `tuning_model.json` by RRL. To run the tuned application and compare the energy consumption to the uninstrumented version (`./test/amg2013_plain`) use

   - `sbatch ./run_rrl.sh` (output of uninstrumented and RRL tuned versions measured by HDEEM in `amg2013_rrl_plain_hdeem.out` and `amg2013_rrl_rrl_hdeem.out`, respectively).

The format of the files `amg2013_rrl_XXX_hdeem.out` follow Tables 4, 5. The respective columns contain the name of the test, test number, number of nodes employed, total number of MPI processes, number of MPI processes per node, number of OpenMP threads per MPI process, runtime of the app in milliseconds, and energy consumption in Joules. It can be seen from the two tables that while the runtime increases for the tuned version (second-last column), the energy consumed decreases (last column).