BenchBuild - Empirical Research Toolkit Documentation

Release 1.3.2

Andreas Simbürger

Jan 13, 2017

Contents:

CHAPTER 1

BenchBuild: Empirical-Research Toolkit

BenchBuild provides a lightweight toolkit to conduct empirical compile-time and run-time experiments. Striving to automate all tedious and error-prone tasks, it downloads, configure and builds all supported projects fully automatic and provides tools to wrap the compiler and any resulting binary with a customized measurement.

All results can be stored as the user desires. BenchBuild tracks the execution status of all its managed projects inside an own database.

1.1 Features

- Wrap compilation commands with arbitrary measurement functions written in python.
- Wrap binary commands with arbitrary measurement functions written in python.
- Parallel benchmarking using the SLURM cluster manager.
- Compile-time support for the gentoo portage tree using the `uchroot` command.

1.2 Requirements

You need a working PostgreSQL installation (There is no special reason for PostgreSQL, but the backend is not configurable at the moment). In addition to the PostgreSQL server, you need libpqxx available for the psycopg2 package that benchbuild uses to connect.

1.3 Installation

After you have installed all necessary libraries, you can just clone this repo and install via pip.

```
$ pip install benchbuild
```

This will pull in all necessary python libraries into your local python installation. The installed program to control the study is called `benchbuild`.

1.4 Configuration

`benchbuild` can be configured in various ways: (1) command-line arguments, (2) configuration file in .json format, (3) environment variables.

You can dump the current active configuration with the command: .. code-block:: bash

$ benchbuild run -d

BB_BENCHBUILD_EBUILD="" BB_BENCHBUILD_PREFIX="/bench-build" BB_BUILD_DIR="/tmp/benchbuild" ...

You can dump this information in .json format using the command:

```
$ benchbuild run -s
```

However, be careful. It dumps _all_ configuration to .json, even those that are usually derived automatically (like UUIDs). In the future, this will be avoided automatically. For now, you should remove all ID related variables from the resulting .json file. The configuration file is searched from the current directory upwards automatically. Some key configuration variables:

`BB_BUILD_DIR`	The directory we place our temporary artifacts in.
`BB_TMP_DIR`	The directory we place our downloads in.
`BB_SRC_DIR`	The directory we pull additional artifacts from (e.g., patches)
`BB_CLEAN`	Should the build directory be cleaned after the run?
`BB_CONFIG_FILE`	Where is the config file? If you prefere an absolute location over automatic discovery.
`BB_DB_HOST`	Hostname of the database
`BB_DB_NAME`	Name of the database
`BB_DB_USER`	Username of the database
`BB_DB_PASS`	Password of the database
`BB_DB_ROLLBACK`	For testing Rollback all db actions after a run.
`BB_JOBS`	Number of threads to use for compiling / run-time testing.

You can set these in the .json config file or directly via environment variables. However, make sure that the values you pass in from the environment are valid JSON, or the configuration structure may ignore your input (or break).

1.5 SLURM Configuration

If you want to run experiments in parallel on a cluster managed by SLURM, you can use BenchBuild to generate a bash script that is compatible with SLURM's sbatch command. The following settings control SLURM's configuration:

BB_SLURM_ACCOUNT	The resource account log in to.
BB_SLURM_CPUS_PER_TASK	How many cores/threads should we request per node?
BB_SLURM_EXCLUSIVE	Should we request the node exclusively or share it with other tasks?
BB_SLURM_LOGS	Where do we put our logs (deprecated).
BB_SLURM_MAX_RUNNING	We generate array-Jobs. This parameter controls the number of array elements that are allowed to run in parallel.
BB_SLURM_MULTITHREAD	Should Hyper-Threading be enabled or not?
BB_SLURM_NICE	Adjust our priority on the cluster manually.
BB_SLURM_NICE_CLEAN	Adjust the priority of the clean jobs.
BB_SLURM_NODE_DIR	Where can we place our artifacts on the node?
BB_SLURM_PARTITION	Which partition should we run in?
BB_SLURM_SCRIPT	Base name of our resulting batch script.
BB_SLURM_TIMELIMIT	Enforce a timelimit on our batch jobs.

1.6 Gentoo Configuration

BenchBuild supports compile-time experiments on the complete portage tree of Gentoo Linux. You need to configure a few settings to make it work:

BB_GENTOO_AUTOTEST_LOC	A txt file that lists all gentoo package atoms that should be considered.
BB_GENTOO_AUTOTEST_FTP_PROXY	Proxy server for gentoo downloads.
BB_GENTOO_AUTOTEST_HTTP_PROXY	Proxy server for gentoo downloads.
BB_GENTOO_AUTOTEST_RSYNC_PROXY	Proxy server for gentoo downloads.

1.7 Convert an automatic Gentoo project to a static one

Gentoo projects are generated dynamically based on the `AutoPortage` class found in `pprof.gentoo.portage_gen`. If you want to define run-time tests for a dynamically generated project, you need to convert it to a static one, i.e., define a subclass of `AutoPortage` and add it to the configuration.

```
from pprof.projects.gentoo.portage_gen import AutoPortage

class BZip(AutoPortage):
  NAME = "app-arch"
  DOMAIN = "bzip2"

  def run_tests(self, experiment):
    """Add your custom test routines here."""
```

Now we just need to add this to the plugin registry via `benchbuild`'s configuration file @ `CFG["plugins"]["projects"]`.

1.8 Documentation

For detailed API information please refer to the full documentation:

1.9 Misc

1.9.1 benchbuild package

1.9.1.1 Subpackages

benchbuild.experiments package

Experiments module.

By default, only experiments that are listed in the configuration are loaded automatically. See configuration variables:

*_PLUGINS_AUTOLOAD *_PLUGINS_EXPERIMENTS

benchbuild.experiments.**discover**()
Import all experiments listed in PLUGINS_EXPERIMENTS.

Tests:

```
>>> from benchbuild.settings import CFG
>>> from benchbuild.experiments import discover
>>> import logging as lg
>>> import sys
>>> l = lg.getLogger('benchbuild')
>>> lg.getLogger('benchbuild').setLevel(lg.DEBUG)
>>> lg.getLogger('benchbuild').handlers = [lg.StreamHandler(stream=sys.
↪stdout)]
>>> CFG["plugins"]["experiments"] = ["benchbuild.non.existing", "benchbuild.
↪experiments.raw"]
>>> discover()
Could not find 'benchbuild.non.existing'
ImportError: No module named 'benchbuild.non'
Found experiment: benchbuild.experiments.raw
```

Subpackages

benchbuild.experiments.polly package

Submodules

benchbuild.experiments.polly.openmp module

The ‘polly-openmp’ Experiment.

This experiment applies polly’s transformations with openmp code generation enabled to all projects and measures the runtime.

This forms the baseline numbers for the other experiments.

Measurements

3 Metrics are generated during this experiment: time.user_s - The time spent in user space in seconds (aka virtual time) time.system_s - The time spent in kernel space in seconds (aka system time) time.real_s - The time spent overall in seconds (aka Wall clock)

class benchbuild.experiments.polly.openmp.**PollyOpenMP**(*projects=None*, *group=None*)
Bases: *benchbuild.experiment.RuntimeExperiment*

Timing experiment with Polly & OpenMP support.

NAME = 'polly-openmp'

actions_for_project(*p*)
Build & Run each project with Polly & OpenMP support.

benchbuild.experiments.polly.openmpvect module

The 'polly-openmp-vectorize' Experiment.

This experiment applies polly's transformations with openmp code generation enabled to all projects and measures the runtime.

This forms the baseline numbers for the other experiments.

Measurements

3 Metrics are generated during this experiment: time.user_s - The time spent in user space in seconds (aka virtual time) time.system_s - The time spent in kernel space in seconds (aka system time) time.real_s - The time spent overall in seconds (aka Wall clock)

class benchbuild.experiments.polly.openmpvect.**PollyOpenMPVectorizer**(*projects=None*, *group=None*)
Bases: *benchbuild.experiment.RuntimeExperiment*

Timing experiment with Polly & OpenMP+Vectorizer support.

NAME = 'polly-openmpvect'

run_project(*p*)

benchbuild.experiments.polly.polly module

The 'polly' Experiment

This experiment applies polly's transformations to all projects and measures the runtime.

This forms the baseline numbers for the other experiments.

Measurements

3 Metrics are generated during this experiment: time.user_s - The time spent in user space in seconds (aka virtual time) time.system_s - The time spent in kernel space in seconds (aka system time) time.real_s - The time spent overall in seconds (aka Wall clock)

class benchbuild.experiments.polly.polly.**Polly**(*projects=None*, *group=None*)
Bases: *benchbuild.experiment.RuntimeExperiment*

The polly experiment.

NAME = 'polly'

run_project(*p*)

benchbuild.experiments.polly.pollyperformance module

The 'polly' Experiment

This experiment applies polly's transformations to all projects and measures the runtime.

This forms the baseline numbers for the other experiments.

Measurements

3 Metrics are generated during this experiment: time.user_s - The time spent in user space in seconds (aka virtual time) time.system_s - The time spent in kernel space in seconds (aka system time) time.real_s - The time spent overall in seconds (aka Wall clock)

class benchbuild.experiments.polly.pollyperformance.**PollyPerformance**(*projects=None*, *group=None*)

Bases: *benchbuild.experiment.RuntimeExperiment*

The polly performance experiment.

NAME = 'pollyperformance'

run_project(*p*)

exception benchbuild.experiments.polly.pollyperformance.**ShouldNotBeNone**

Bases: RuntimeWarning

User warning, if config var is null.

benchbuild.experiments.polly.vectorize module

The 'polly-vectorize' Experiment

This experiment applies polly's transformations with stripmine vectorizer enabled to all projects and measures the runtime.

This forms the baseline numbers for the other experiments.

Measurements

3 Metrics are generated during this experiment: time.user_s - The time spent in user space in seconds (aka virtual time) time.system_s - The time spent in kernel space in seconds (aka system time) time.real_s - The time spent overall in seconds (aka Wall clock)

class benchbuild.experiments.polly.vectorize.**PollyVectorizer**(*projects=None*, *group=None*)

Bases: *benchbuild.experiment.RuntimeExperiment*

The polly experiment with vectorization enabled.

NAME = 'polly-vectorize'

run_project(*p*)

Submodules

benchbuild.experiments.compilestats module

The 'compilestats' Experiment.

This experiment is a basic experiment in the benchbuild study. It simply runs all projects after compiling it with -O3 and catches all statistics emitted by llvm

class benchbuild.experiments.compilestats.**CompilestatsExperiment**(*projects=None*, *group=None*)

Bases: *benchbuild.experiment.RuntimeExperiment*

The compilestats experiment.

NAME = 'cs'

actions_for_project(*p*)

class benchbuild.experiments.compilestats.**PollyCompilestatsExperiment**(*projects=None*, *group=None*)

Bases: *benchbuild.experiment.RuntimeExperiment*

The compilestats experiment with polly enabled.

NAME = 'p-cs'

actions_for_project(*p*)

benchbuild.experiments.compilestats.**collect_compilestats**(*project*, *experiment*, *config*, *clang*, ***kwargs*)

Collect compilestats.

benchbuild.experiments.compilestats.**get_compilestats**(*prog_out*)

Get the LLVM compilation stats from :prog_out:.

benchbuild.experiments.compilestats_ewpt module

The 'compilestats_ewpt' Experiment.

Gathers compilation statistics for compiling the project with the EWPT alias analysis enabled.

class benchbuild.experiments.compilestats_ewpt.**EWPTCompilestatsExperiment**(*projects=None*, *group=None*)

Bases: *benchbuild.experiments.compilestats.CompilestatsExperiment*

NAME = 'ewpt'

extra_cflags()

benchbuild.experiments.empty module

The 'empty' Experiment.

This experiment is for debugging purposes. It only prepares the basic directories for benchbuild. No compilation & no run can be done with it.

class benchbuild.experiments.empty.**Empty**(*projects=None*, *group=None*)

Bases: *benchbuild.experiment.Experiment*

The empty experiment.

NAME = 'empty'

actions_for_project(*p*)
Do nothing.

benchbuild.experiments.papi module

PAPI based experiments.

These types of experiments (papi & papi-std) need to instrument the project with libbenchbuild support to work.

class benchbuild.experiments.papi.**PapiScopCoverage**(*projects=None*, *group=None*)
Bases: *benchbuild.experiment.RuntimeExperiment*

PAPI-based dynamic SCoP coverage measurement.

NAME = 'papi'

actions_for_project(*p*)
Create & Run a papi-instrumented version of the project.

This experiment uses the -jitable flag of libPolyJIT to generate dynamic SCoP coverage.

run()
Do the postprocessing, after all projects are done.

class benchbuild.experiments.papi.**PapiStandardScopCoverage**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.papi.PapiScopCoverage*

PAPI Scop Coverage, without JIT.

NAME = 'papi-std'

actions_for_project(*p*)
Create & Run a papi-instrumented version of the project.

This experiment uses the -jitable flag of libPolyJIT to generate dynamic SCoP coverage.

benchbuild.experiments.papi.**collect_compilestats**(*project*, *experiment*, *clang*, ***kwargs*)
Collect compilestats.

benchbuild.experiments.papi.**get_compilestats**(*prog_out*)
Get the LLVM compilation stats from :prog_out:.

benchbuild.experiments.pjtest module

A test experiment for PolyJIT.

This experiment should only be used to test various features of PolyJIT. It provides only 1 configuration (maximum number of cores) and tests 2 run-time execution profiles of PolyJIT:

1. PolyJIT enabled, with specialization
2. PolyJIT enabled, without specialization

class benchbuild.experiments.pjtest.**Test**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.polyjit.PolyJIT*

An experiment that executes all projects with PolyJIT support.

This is our default experiment for speedup measurements.

NAME = 'pj-test'

actions_for_project(*p*)

benchbuild.experiments.pjtest.**time_polyjit_and_polly**(*project*, *experiment*, *config*, *jobs*, *run_f*, *args*, ***kwargs*)

Run the given binary wrapped with time.

Parameters

- **project** – The benchbuild.project.
- **experiment** – The benchbuild.experiment.
- **config** – The benchbuild.settings.config.
- **jobs** – Number of cores we should use for this exection.
- **run_f** – The file we want to execute.
- **args** – List of arguments that should be passed to the wrapped binary.
- ****kwargs** – Dictionary with our keyword args. We support the following entries:

  project_name: The real name of our project. This might not be the same as the configured project name, if we got wrapped with ::benchbuild.project.wrap_dynamic

  has_stdin: Signals whether we should take care of stdin.

benchbuild.experiments.polyjit module

The 'polyjit' experiment.

This experiment uses likwid to measure the performance of all binaries when running with polyjit support enabled.

class benchbuild.experiments.polyjit.**Compilestats**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.polyjit.PolyJIT*

Gather compilestats, with enabled JIT.

NAME = 'pj-cs'

actions_for_project(*p*)

class benchbuild.experiments.polyjit.**PJITRaw**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.polyjit.PolyJIT*

An experiment that executes all projects with PolyJIT support.

This is our default experiment for speedup measurements.

NAME = 'pj-raw'

actions_for_project(*p*)

class benchbuild.experiments.polyjit.**PJITRegression**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.polyjit.PolyJIT*

This experiment will generate a series of regression tests.

This can be used every time a new revision is produced for PolyJIT, as it will automatically collect any new SCoPs detected, using the JIT.

The collection of the tests itself is intgrated into the JIT, so this experiment looks a lot like a RAW experiment, except we don't run anything.

NAME = 'pj-collect'

actions_for_project(*p*)

class benchbuild.experiments.polyjit.**PJITlikwid**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.polyjit.PolyJIT*

An experiment that uses likwid's instrumentation API for profiling.

This instruments all projects with likwid instrumentation API calls in key regions of the JIT.

This allows for arbitrary profiling of PolyJIT's overhead and run-time

NAME = 'pj-likwid'

actions_for_project(*p*)

class benchbuild.experiments.polyjit.**PJITpapi**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.polyjit.PolyJIT*

Experiment that uses PolyJIT's instrumentation facilities.

This uses PolyJIT to instrument all projects with libPAPI based region measurements. In the end the region measurements are aggregated and metrics like the dynamic SCoP coverage are extracted.

This uses the same set of flags as all other PolyJIT based experiments.

NAME = 'pj-papi'

actions_for_project(*p*)

run()
Do the postprocessing, after all projects are done.

class benchbuild.experiments.polyjit.**PJITperf**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.polyjit.PolyJIT*

An experiment that uses linux perf tools to generate flamegraphs.

NAME = 'pj-perf'

actions_for_project(*p*)

class benchbuild.experiments.polyjit.**PolyJIT**(*projects=None*, *group=None*)
Bases: *benchbuild.experiment.RuntimeExperiment*

The polyjit experiment.

actions_for_project(*p*)

classmethod init_project(*project*)
Execute the benchbuild experiment.

We perform this experiment in 2 steps:

1. with likwid disabled.
2. with likwid enabled.

Parameters **project** – The project we initialize.

Returns The initialized project.

class benchbuild.experiments.polyjit.**PolyJITFull**(*projects=None*, *group=None*)
Bases: *benchbuild.experiments.polyjit.PolyJIT*

An experiment that executes all projects with PolyJIT support.

This is our default experiment for speedup measurements.

NAME = 'pj'

actions_for_project(*p*)

benchbuild.experiments.polyjit.**run_raw**(*project*, *experiment*, *config*, *run_f*, *args*, ***kwargs*)

Run the given binary wrapped with nothing.

Parameters

- **project** – The benchbuild.project.
- **experiment** – The benchbuild.experiment.
- **config** – The benchbuild.settings.config.
- **run_f** – The file we want to execute.
- **args** – List of arguments that should be passed to the wrapped binary.
- ****kwargs** – Dictionary with our keyword args. We support the following entries:

  project_name: The real name of our project. This might not be the same as the configured project name, if we got wrapped with ::benchbuild.project.wrap_dynamic

  has_stdin: Signals whether we should take care of stdin.

benchbuild.experiments.polyjit.**run_with_likwid**(*project*, *experiment*, *config*, *jobs*, *run_f*, *args*, ***kwargs*)

Run the given file wrapped by likwid.

Parameters

- **project** – The benchbuild.project.
- **experiment** – The benchbuild.experiment.
- **config** – The benchbuild.settings.config.
- **jobs** – Number of cores we should use for this exection.
- **run_f** – The file we want to execute.
- **args** – List of arguments that should be passed to the wrapped binary.
- ****kwargs** – Dictionary with our keyword args. We support the following entries:

  project_name: The real name of our project. This might not be the same as the configured project name, if we got wrapped with ::benchbuild.project.wrap_dynamic

  has_stdin: Signals whether we should take care of stdin.

benchbuild.experiments.polyjit.**run_with_papi**(*project*, *experiment*, *config*, *jobs*, *run_f*, *args*, ***kwargs*)

Run the given file with PAPI support.

This just runs the project as PAPI support should be compiled in already. If not, this won't do a lot.

Parameters

- **project** – The benchbuild.project.
- **experiment** – The benchbuild.experiment.
- **config** – The benchbuild.settings.config.
- **jobs** – Number of cores we should use for this exection.
- **run_f** – The file we want to execute.
- **args** – List of arguments that should be passed to the wrapped binary.

- ****kwargs** – Dictionary with our keyword args. We support the following entries:

  project_name: The real name of our project. This might not be the same as the configured project name, if we got wrapped with ::benchbuild.project.wrap_dynamic

  has_stdin: Signals whether we should take care of stdin.

benchbuild.experiments.polyjit.**run_with_perf**(*project*, *experiment*, *config*, *jobs*, *run_f*, *args*, ***kwargs*)

Run the given binary wrapped with time.

Parameters

- **project** – The benchbuild.project.
- **experiment** – The benchbuild.experiment.
- **config** – The benchbuild.settings.config.
- **jobs** – Number of cores we should use for this exection.
- **run_f** – The file we want to execute.
- **args** – List of arguments that should be passed to the wrapped binary.
- ****kwargs** – Dictionary with our keyword args. We support the following entries:

  project_name: The real name of our project. This might not be the same as the configured project name, if we got wrapped with ::benchbuild.project.wrap_dynamic

  has_stdin: Signals whether we should take care of stdin.

benchbuild.experiments.polyjit.**run_with_time**(*project*, *experiment*, *config*, *jobs*, *run_f*, *args*, ***kwargs*)

Run the given binary wrapped with time.

Parameters

- **project** – The benchbuild.project.
- **experiment** – The benchbuild.experiment.
- **config** – The benchbuild.settings.config.
- **jobs** – Number of cores we should use for this exection.
- **run_f** – The file we want to execute.
- **args** – List of arguments that should be passed to the wrapped binary.
- ****kwargs** – Dictionary with our keyword args. We support the following entries:

  project_name: The real name of our project. This might not be the same as the configured project name, if we got wrapped with ::benchbuild.project.wrap_dynamic

  has_stdin: Signals whether we should take care of stdin.

benchbuild.experiments.polyjit.**run_without_recompile**(*project*, *experiment*, *config*, *jobs*, *run_f*, *args*, ***kwargs*)

Run the given binary wrapped with time.

Parameters

- **project** – The benchbuild.project.
- **experiment** – The benchbuild.experiment.
- **config** – The benchbuild.settings.config.

- **jobs** – Number of cores we should use for this exection.
- **run_f** – The file we want to execute.
- **args** – List of arguments that should be passed to the wrapped binary.
- ****kwargs** – Dictionary with our keyword args. We support the following entries:

  project_name: The real name of our project. This might not be the same as the configured project name, if we got wrapped with ::benchbuild.project.wrap_dynamic

  has_stdin: Signals whether we should take care of stdin.

benchbuild.experiments.raw module

The 'raw' Experiment.

This experiment is the basic experiment in the benchbuild study. It simply runs all projects after compiling it with -O3. The binaries are wrapped with the time command and results are written to the database.

This forms the baseline numbers for the other experiments.

Measurements

3 Metrics are generated during this experiment: time.user_s - The time spent in user space in seconds (aka virtual time) time.system_s - The time spent in kernel space in seconds (aka system time) time.real_s - The time spent overall in seconds (aka Wall clock)

class benchbuild.experiments.raw.**RawRuntime**(*projects=None*, *group=None*)
Bases: *benchbuild.experiment.RuntimeExperiment*

The polyjit experiment.

NAME = 'raw'

actions_for_project(*project*)
Compile & Run the experiment with -O3 enabled.

benchbuild.experiments.raw.**run_with_time**(*project*, *experiment*, *config*, *jobs*, *run_f*, *args*, ***kwargs*)
Run the given binary wrapped with time.

Parameters

- **project** – The benchbuild project that has called us.
- **experiment** – The benchbuild experiment which we operate under.
- **config** – The benchbuild configuration we are running with.
- **jobs** – The number of cores we are allowed to use. This may differ from the actual amount of available cores, obey it. We should enforce this from the outside. However, at the moment we do not do this.
- **run_f** – The file we want to execute.
- **args** – List of arguments that should be passed to the wrapped binary.
- ****kwargs** – Dictionary with our keyword args. We support the following entries:

  project_name: The real name of our project. This might not be the same as the configured project name, if we got wrapped with ::benchbuild.project.wrap_dynamic

has_stdin: Signals whether we should take care of stdin. may_wrap:

Project may signal that it they are not suitable for wrapping. Usually because they scan/parse the output, which may interfere with the output of the wrapper binary.

benchbuild.projects package

Projects module.

By default, only projects that are listed in the configuration are loaded automatically. See configuration variables:

*_PLUGINS_AUTOLOAD *_PLUGINS_PROJECTS

benchbuild.projects.**discover**()

Subpackages

benchbuild.projects.apollo package

Submodules

benchbuild.projects.apollo.group module

class benchbuild.projects.apollo.group.**ApolloGroup**(*exp*)

Bases: *benchbuild.project.Project*

GROUP = 'apollo'

path_suffix = 'src'

benchbuild.projects.apollo.rodinia module

class benchbuild.projects.apollo.rodinia.**Rodinia**(*exp*)

Bases: *benchbuild.projects.apollo.group.ApolloGroup*

DOMAIN = 'scientific'

NAME = 'rodinia'

SRC_FILE = 'rodinia_3.1.tar.bz2'

VERSION = '3.1'

build()

configure()

download()

prepare()

run_tests(*experiment*)

src_dir = 'rodinia_3.1'

src_uri = 'http://www.cs.virginia.edu/~kw5na/lava/Rodinia/Packages/Current/rodinia_3.1.tar.bz2'

benchbuild.projects.apollo.scimark module

class benchbuild.projects.apollo.scimark.**SciMark**(*exp*)
Bases: *benchbuild.projects.apollo.group.ApolloGroup*

DOMAIN = 'scientific'

NAME = 'scimark'

SRC_FILE = 'scimark2_1c.zip'

VERSION = '2.1c'

build()

configure()

download()

prepare()

run_tests(*experiment*)

src_uri = 'http://math.nist.gov/scimark2/scimark2_1c.zip'

benchbuild.projects.benchbuild package

Submodules

benchbuild.projects.benchbuild.bzip2 module

class benchbuild.projects.benchbuild.bzip2.**Bzip2**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'compression'

NAME = 'bzip2'

SRC_FILE = 'bzip2-1.0.6.tar.gz'

VERSION = '1.0.6'

build()

configure()

download()

prepare()

run_tests(*experiment*)

src_dir = 'bzip2-1.0.6'

src_uri = 'http://www.bzip.org/1.0.6/bzip2-1.0.6.tar.gz'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.benchbuild.ccrypt module

class benchbuild.projects.benchbuild.ccrypt.**Ccrypt**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

ccrypt benchmark

DOMAIN = 'encryption'

NAME = 'ccrypt'

SRC_FILE = 'ccrypt-1.10.tar.gz'

VERSION = '1.10'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'ccrypt-1.10'

src_uri = 'http://ccrypt.sourceforge.net/download/ccrypt-1.10.tar.gz'

benchbuild.projects.benchbuild.crafty module

class benchbuild.projects.benchbuild.crafty.**Crafty**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

crafty benchmark

DOMAIN = 'scientific'

NAME = 'crafty'

SRC_FILE = 'crafty-25.2.zip'

VERSION = '25.2'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'crafty-25.2'

src_uri = 'http://www.craftychess.com/downloads/source/crafty-25.2.zip'

benchbuild.projects.benchbuild.crocopat module

class benchbuild.projects.benchbuild.crocopat.**Crocopat**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

crocopat benchmark

DOMAIN = 'verification'

NAME = 'crocopat'

SRC_FILE = 'crocopat-2.1.4.zip'

VERSION = '2.1.4'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'crocopat-2.1.4'

src_uri = 'http://crocopat.googlecode.com/files/crocopat-2.1.4.zip'

benchbuild.projects.benchbuild.ffmpeg module

class benchbuild.projects.benchbuild.ffmpeg.**LibAV**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

LibAV benchmark

DOMAIN = 'multimedia'

NAME = 'ffmpeg'

SRC_FILE = 'ffmpeg-3.1.3.tar.bz2'

VERSION = '3.1.3'

build()

configure()

download()

fate_dir = 'fate-samples'

fate_uri = 'rsync://fate-suite.libav.org/fate-suite/'

run_tests(*experiment*)

src_dir = 'ffmpeg-3.1.3'

src_uri = 'http://ffmpeg.org/releases/ffmpeg-3.1.3.tar.bz2'

benchbuild.projects.benchbuild.group module

class benchbuild.projects.benchbuild.group.**BenchBuildGroup**(*exp*)
Bases: *benchbuild.project.Project*

GROUP = 'benchbuild'

path_suffix = 'src'

benchbuild.projects.benchbuild.gzip module

class benchbuild.projects.benchbuild.gzip.**Gzip**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'compression'

NAME = 'gzip'

SRC_FILE = 'gzip-1.6.tar.xz'

VERSION = '1.6'

build()

configure()

download()

prepare()

run_tests(*experiment*)

src_dir = 'gzip-1.6'

src_uri = 'http://ftpmirror.gnu.org/gzip/gzip-1.6.tar.xz'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.benchbuild.js module

class benchbuild.projects.benchbuild.js.**SpiderMonkey**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

SpiderMonkey requires a legacy version of autoconf: autoconf-2.13

DOMAIN = 'compilation'

NAME = 'js'

VERSION = ''

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'gecko-dev.git'

src_uri = 'https://github.com/mozilla/gecko-dev.git'

version = ''

benchbuild.projects.benchbuild.lammps module

class benchbuild.projects.benchbuild.lammps.**Lammps**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

LAMMPS benchmark

DOMAIN = 'scientific'

NAME = 'lammps'

SRC_FILE = 'lammps.git'

build()

configure()

download()

prepare()

run_tests(*experiment*)

src_dir = 'lammps.git'

src_uri = 'https://github.com/lammps/lammps'

benchbuild.projects.benchbuild.lapack module

class benchbuild.projects.benchbuild.lapack.**Lapack**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'scientific'

NAME = 'lapack'

SRC_FILE = 'clapack.tgz'

VERSION = '3.2.1'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'CLAPACK-3.2.1'

src_uri = 'http://www.netlib.org/clapack/clapack.tgz'

class benchbuild.projects.benchbuild.lapack.**OpenBlas**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'scientific'

NAME = 'openblas'

SRC_FILE = 'OpenBLAS'

build()

configure()

download()

run_tests(*experiment*)

src_uri = 'https://github.com/xianyi/OpenBLAS'

benchbuild.projects.benchbuild.leveldb module

class benchbuild.projects.benchbuild.leveldb.**LevelDB**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'database'

NAME = 'leveldb'

SRC_FILE = 'leveldb.src'

build()

configure()

download()

run_tests(*experiment*)
Execute LevelDB's runtime configuration.

Parameters **experiment** – The experiment's run function.

src_uri = 'https://github.com/google/leveldb'

benchbuild.projects.benchbuild.linpack module

class benchbuild.projects.benchbuild.linpack.**Linpack**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

Linpack (C-Version)

DOMAIN = 'scientific'

NAME = 'linpack'

build()

configure()

download()

src_uri = 'http://www.netlib.org/benchmark/linpackc.new'

benchbuild.projects.benchbuild.lulesh module

class benchbuild.projects.benchbuild.lulesh.**Lulesh**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'scientific'

NAME = 'lulesh'

SRC_FILE = 'LULESH.cc'

build()

configure()

download()

run_tests(*experiment*)

src_uri = 'https://codesign.llnl.gov/lulesh/LULESH.cc'

benchbuild.projects.benchbuild.luleshomp module

class benchbuild.projects.benchbuild.luleshomp.**LuleshOMP**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

Lulesh-OMP

DOMAIN = 'scientific'

NAME = 'lulesh-omp'

SRC_FILE = 'LULESH_OMP.cc'

build()

configure()

download()

run_tests(*experiment*)

src_uri = 'https://codesign.llnl.gov/lulesh/LULESH_OMP.cc'

benchbuild.projects.benchbuild.mcrypt module

class benchbuild.projects.benchbuild.mcrypt.**MCrypt**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

MCrypt benchmark

DOMAIN = 'encryption'

NAME = 'mcrypt'

SRC_FILE = 'mcrypt-2.6.8.tar.gz'

VERSION = '2.6.8'

build()

configure()

download()

libmcrypt_dir = 'libmcrypt-2.5.8'

libmcrypt_file = 'libmcrypt-2.5.8.tar.gz'

libmcrypt_uri = 'http://sourceforge.net/projects/mcrypt/files/Libmcrypt/2.5.8/libmcrypt-2.5.8.tar.gz'

mhash_dir = 'mhash-0.9.9.9'

mhash_file = 'mhash-0.9.9.9.tar.gz'

mhash_uri = 'http://sourceforge.net/projects/mhash/files/mhash/0.9.9.9/mhash-0.9.9.9.tar.gz'

run_tests(*experiment*)

src_dir = 'mcrypt-2.6.8'

src_uri = 'http://sourceforge.net/projects/mcrypt/files/MCrypt/2.6.8mcrypt-2.6.8.tar.gz'

benchbuild.projects.benchbuild.minisat module

class benchbuild.projects.benchbuild.minisat.**Minisat**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

minisat benchmark

DOMAIN = 'verification'

NAME = 'minisat'

SRC_FILE = 'minisat.git'

build()

configure()

download()

run_tests(*experiment*)

src_uri = 'https://github.com/niklasso/minisat'

benchbuild.projects.benchbuild.openssl module

class benchbuild.projects.benchbuild.openssl.**LibreSSL**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

OpenSSL

DOMAIN = 'encryption'

NAME = 'libressl'

SRC_FILE = 'libressl-2.1.6.tar.gz'

VERSION = '2.1.6'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'libressl-2.1.6'

src_uri = 'http://ftp.openbsd.org/pub/OpenBSD/LibreSSL/libressl-2.1.6.tar.gz'

benchbuild.projects.benchbuild.postgres module

class benchbuild.projects.benchbuild.postgres.**Postgres**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

postgres benchmark

DOMAIN = 'database'

NAME = 'postgres'

prepare()

run_tests(*experiment*)

testfiles = ['pg_ctl', 'dropdb', 'createdb', 'pgbench']

benchbuild.projects.benchbuild.povray module

class benchbuild.projects.benchbuild.povray.**Povray**(*exp*)
Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

povray benchmark

DOMAIN = 'multimedia'

NAME = 'povray'

SRC_FILE = 'povray.git'

boost_src_dir = 'boost_1_59_0'

boost_src_file = 'boost_1_59_0.tar.bz2'

boost_src_uri = 'http://sourceforge.net/projects/boost/files/boost/1.59.0/boost_1_59_0.tar.bz2'

build()

configure()

download()

prepare()

run_tests(*experiment*)

src_uri = 'https://github.com/POV-Ray/povray'

benchbuild.projects.benchbuild.python module

class benchbuild.projects.benchbuild.python.**Python**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

python benchmarks

DOMAIN = 'compilation'

NAME = 'python'

SRC_FILE = 'Python-3.4.3.tar.xz'

VERSION = '3.4.3'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'Python-3.4.3'

src_uri = 'https://www.python.org/ftp/python/3.4.3/Python-3.4.3.tar.xz'

benchbuild.projects.benchbuild.rasdaman module

class benchbuild.projects.benchbuild.rasdaman.**Rasdaman**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'database'

NAME = 'Rasdaman'

SRC_FILE = 'rasdaman.git'

build()

configure()

download()

gdal_dir = 'gdal'

gdal_uri = 'https://github.com/OSGeo/gdal'

run_tests(*experiment*)

src_uri = 'git://rasdaman.org/rasdaman.git'

benchbuild.projects.benchbuild.ruby module

class benchbuild.projects.benchbuild.ruby.**Ruby**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'compilation'

NAME = 'ruby'

SRC_FILE = 'ruby-2.2.2.tar.gz'

VERSION = '2.2.2'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'ruby-2.2.2'

src_uri = 'http://cache.ruby-lang.org/pub/ruby/2.2.2/ruby-2.2.2.tar.gz'

benchbuild.projects.benchbuild.sdcc module

class benchbuild.projects.benchbuild.sdcc.**SDCC**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'compilation'

NAME = 'sdcc'

SRC_FILE = 'sdcc'

build()

configure()

download()

run_tests(*experiment*)

src_uri = 'svn://svn.code.sf.net/p/sdcc/code/trunk/sdcc'

benchbuild.projects.benchbuild.sevenz module

class benchbuild.projects.benchbuild.sevenz.**SevenZip**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

7Zip

DOMAIN = 'compression'

NAME = '7z'

SRC_FILE = 'p7zip_9.38.1_src_all.tar.bz2'

VERSION = '9.38.1'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'p7zip_9.38.1'

src_uri = 'http://downloads.sourceforge.net/project/p7zip/p7zip/9.38.1/p7zip_9.38.1_src_all.tar.bz2'

benchbuild.projects.benchbuild.sqlite3 module

class benchbuild.projects.benchbuild.sqlite3.**SQLite3**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'database'

NAME = 'sqlite3'

SRC_FILE = 'sqlite-amalgamation-3080900.zip'

build()

build_leveldb()

configure()

download()

fetch_leveldb()

run_tests(*experiment*)

src_dir = 'sqlite-amalgamation-3080900'

src_uri = 'http://www.sqlite.org/2015/sqlite-amalgamation-3080900.zip'

benchbuild.projects.benchbuild.tcc module

class benchbuild.projects.benchbuild.tcc.**TCC**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'compilation'

NAME = 'tcc'

SRC_FILE = 'tcc-0.9.26.tar.bz2'

VERSION = '0.9.26'

build()

configure()

download()

run_tests(*experiment*)

src_dir = 'tcc-0.9.26'

src_uri = 'http://download-mirror.savannah.gnu.org/releases/tinycc/tcc-0.9.26.tar.bz2'

benchbuild.projects.benchbuild.x264 module

class benchbuild.projects.benchbuild.x264.**X264**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

x264

DOMAIN = 'multimedia'

NAME = 'x264'

SRC_FILE = 'x264.git'

build()

configure()

download()

inputfiles = {'tbbt-small.y4m': [], 'Sintel.2010.720p.raw': ['–input-res', '1280x720']}

prepare()

run_tests(*experiment*)

src_uri = 'git://git.videolan.org/x264.git'

benchbuild.projects.benchbuild.xz module

class benchbuild.projects.benchbuild.xz.**XZ**(*exp*)

Bases: *benchbuild.projects.benchbuild.group.BenchBuildGroup*

DOMAIN = 'compression'

NAME = 'xz'

SRC_FILE = 'xz-5.2.1.tar.gz'

VERSION = '5.2.1'

build()

configure()

download()

prepare()

run_tests(*experiment*)

src_dir = 'xz-5.2.1'

src_uri = 'http://tukaani.org/xz/xz-5.2.1.tar.gz'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.gentoo package

Import all gentoo based modules.

All manually entered modules can be placed in the following import section. Portage_Gen based projects will be generated automatically as soon as we can find an index generated by portage info.

Submodules

benchbuild.projects.gentoo.autoportage module

class benchbuild.projects.gentoo.autoportage.**AutoPortage**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

Generic portage experiment.

build()

run_tests(_)

benchbuild.projects.gentoo.bzip2 module

bzip2 experiment within gentoo chroot.

class benchbuild.projects.gentoo.bzip2.**BZip2**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

app-arch/bzip2

DOMAIN = 'app-arch'

NAME = 'gentoo-bzip2'

VERSION = '1.0.6'

build()

prepare()

run_tests(*experiment*)

test_archive = 'compression.tar.gz'

test_url = 'http://lairosiel.de/dist/'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.gentoo.crafty module

crafty experiment within gentoo chroot.

class benchbuild.projects.gentoo.crafty.**Crafty**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

games-board/crafty

DOMAIN = 'games-board'

NAME = 'gentoo-crafty'

build()

download()

run_tests(*experiment*)

benchbuild.projects.gentoo.eix module

eix experiment within gentoo chroot

class benchbuild.projects.gentoo.eix.**Eix**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

Represents the package eix from the portage tree.

DOMAIN = 'app-portage'

NAME = 'eix'

build()
Compiles and installes eix within gentoo chroot

run_tests(*experiment*)
Runs runtime tests for eix

benchbuild.projects.gentoo.gentoo module

The Gentoo module for running tests on builds from the portage tree.

This will install a stage3 image of gentoo together with a recent snapshot of the portage tree. For building / executing arbitrary projects successfully it is necessary to keep the installed image as close to the host system as possible. In order to speed up your experience, you can replace the stage3 image that we pull from the distfiles mirror with a new image that contains all necessary dependencies for your experiments. Make sure you update the hash alongside the gentoo image in benchbuild's source directory.

class benchbuild.projects.gentoo.gentoo.**GentooGroup**(*exp*)
Bases: *benchbuild.project.Project*

Gentoo ProjectGroup is the base class for every portage build.

CONTAINER = <benchbuild.utils.container.Gentoo object>

GROUP = 'gentoo'

SRC_FILE = None

build()

configure()

download()

write_bashrc(*path*)

write_layout(*path*)

write_makeconfig(*path*)

write_wgetrc(*path*)

benchbuild.projects.gentoo.gzip module

gzip experiment within gentoo chroot.

class benchbuild.projects.gentoo.gzip.**GZip**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

app-arch/gzip

DOMAIN = 'app-arch'

NAME = 'gentoo-gzip'

build()

prepare()

run_tests(*experiment*)

test_archive = 'compression.tar.gz'

test_url = 'http://lairosiel.de/dist/'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.gentoo.info module

Get package infos, e.g., specific ebuilds for given languages, from gentoo chroot.

class benchbuild.projects.gentoo.info.**Info**(*exp*)
Bases: *benchbuild.projects.gentoo.autoportage.AutoPortage*

Info experiment to retrieve package information from portage.

DOMAIN = 'debug'

NAME = 'gentoo-info'

build()

benchbuild.projects.gentoo.info.**get_string_for_language**(*language_name*)
Maps language names to the corresponding string for qgrep.

benchbuild.projects.gentoo.lammps module

LAMMPS (sci-physics/lammps) project within gentoo chroot.

class benchbuild.projects.gentoo.lammps.**Lammps**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

sci-physics/lammps

DOMAIN = 'sci-physics'

NAME = 'gentoo-lammps'

build()

prepare()

run_tests(*experiment*)

test_archive = 'lammps.tar.gz'

test_url = 'http://lairosiel.de/dist/'

benchbuild.projects.gentoo.portage_gen module

Generic experiment to test portage packages within gentoo chroot.

benchbuild.projects.gentoo.portage_gen.**PortageFactory**(*name*, *NAME*, *DOMAIN*, *BaseClass=<class 'benchbuild.projects.gentoo.autoportage.AutoPortage'>*)

Create a new dynamic portage project.

Auto-Generated projects can only be used for compilie-time experiments, because there simply is no run-time test defined for it. Therefore, we implement the run symbol as a noop (with minor logging).

This way we avoid the default implementation for run() that all projects inherit.

Parameters

- **name** – Name of the dynamic class.
- **NAME** – NAME property of the dynamic class.
- **DOMAIN** – DOMAIN property of the dynamic class.
- **BaseClass** – Base class to use for the dynamic class.

Returns A new class with NAME,DOMAIN properties set, unable to perform run-time tests.

Examples

```
>>> from benchbuild.projects.gentoo.portage_gen import PortageFactory
>>> from benchbuild.experiments.empty import Empty
>>> c = PortageFactory("test", "NAME", "DOMAIN")
>>> c
<class '__main__.test'>
>>> i = c(Empty())
>>> i.NAME
'NAME'
>>> i.DOMAIN
'DOMAIN'
```

benchbuild.projects.gentoo.postgresql module

postgresql experiment within gentoo chroot.

class benchbuild.projects.gentoo.postgresql.**Postgresql**(*exp*)

Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

dev-db/postgresql

DOMAIN = 'dev-db/postgresql'

NAME = 'gentoo-postgresql'

build()

outside(*chroot_path*)

Return the path with the outside prefix.

Parameters **chroot_path** – the path inside the chroot.

Returns Absolute path outside this chroot.

run_tests(*experiment*)

benchbuild.projects.gentoo.sevenz module

p7zip experiment within gentoo chroot.

class benchbuild.projects.gentoo.sevenz.**SevenZip**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

app-arch/p7zip

DOMAIN = 'app-arch'

NAME = 'gentoo-p7zip'

build()

run_tests(*experiment*)

benchbuild.projects.gentoo.x264 module

media-video/x264-encoder within gentoo chroot.

class benchbuild.projects.gentoo.x264.**X264**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

media-video/x264-encoder

DOMAIN = 'media-libs'

NAME = 'gentoo-x264'

build()

inputfiles = {'tbbt-small.y4m': [], 'Sintel.2010.720p.raw': ['–input-res', '1280x720']}

prepare()

run_tests(*experiment*)

test_url = 'http://lairosiel.de/dist/'

benchbuild.projects.gentoo.xz module

xz experiment within gentoo chroot.

class benchbuild.projects.gentoo.xz.**XZ**(*exp*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

app-arch/xz

DOMAIN = 'app-arch'

NAME = 'gentoo-xz'

build()

prepare()

run_tests (*experiment*)

test_archive = 'compression.tar.gz'

test_url = 'http://lairosiel.de/dist/'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.lnt package

Submodules

benchbuild.projects.lnt.lnt module

LNT based measurements.

class benchbuild.projects.lnt.lnt.**LNTGroup**(*exp*)
Bases: *benchbuild.project.Project*

LNT ProjectGroup for running the lnt test suite.

DOMAIN = 'lnt'

GROUP = 'lnt'

VERSION = '9.0.1.13'

build()

configure()

download()

src_dir = 'lnt'

src_uri = 'http://llvm.org/git/lnt'

test_suite_dir = 'test-suite'

test_suite_uri = 'http://llvm.org/git/test-suite'

class benchbuild.projects.lnt.lnt.**MultiSourceApplications**(*exp*)
Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

NAME = 'MultiSourceApplications'

run_tests(*experiment*)

class benchbuild.projects.lnt.lnt.**MultiSourceBenchmarks**(*exp*)
Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

NAME = 'MultiSourceBenchmarks'

run_tests(*experiment*)

class benchbuild.projects.lnt.lnt.**Povray**(*exp*)
Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

NAME = 'Povray'

download()

povray_src_dir = 'Povray'

povray_url = 'https://github.com/POV-Ray/povray'

run_tests(*experiment*)

class benchbuild.projects.lnt.lnt.**SPEC2006**(*exp*)
Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

NAME = 'SPEC2006'

download()

run_tests(*experiment*)

class benchbuild.projects.lnt.lnt.**SingleSourceBenchmarks**(*exp*)
Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

NAME = 'SingleSourceBenchmarks'

run_tests(*experiment*)

benchbuild.projects.polybench package

Submodules

benchbuild.projects.polybench.polybench module

class benchbuild.projects.polybench.polybench.**Adi**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'adi'

class benchbuild.projects.polybench.polybench.**Atax**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'atax'

class benchbuild.projects.polybench.polybench.**BicG**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'bicg'

class benchbuild.projects.polybench.polybench.**Cholesky**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'cholesky'

class benchbuild.projects.polybench.polybench.**Correlation**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'correlation'

class benchbuild.projects.polybench.polybench.**Covariance**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'covariance'

class benchbuild.projects.polybench.polybench.**Deriche**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'deriche'

class benchbuild.projects.polybench.polybench.**Doitgen**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'doitgen'

class benchbuild.projects.polybench.polybench.**Durbin**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'durbin'

class benchbuild.projects.polybench.polybench.**FDTD2D**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'fdtd-2d'

class benchbuild.projects.polybench.polybench.**FloydWarshall**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'floyd-warshall'

class benchbuild.projects.polybench.polybench.**Gemm**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'gemm'

class benchbuild.projects.polybench.polybench.**Gemver**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'gemver'

class benchbuild.projects.polybench.polybench.**Gesummv**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'gesummv'

class benchbuild.projects.polybench.polybench.**Gramschmidt**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'gramschmidt'

class benchbuild.projects.polybench.polybench.**Heat3D**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'heat-3d'

class benchbuild.projects.polybench.polybench.**Jacobi1D**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'jacobi-1d'

class benchbuild.projects.polybench.polybench.**Jacobi2Dimper**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'jacobi-2d'

class benchbuild.projects.polybench.polybench.**Lu**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'lu'

class benchbuild.projects.polybench.polybench.**LuDCMP**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'ludcmp'

class benchbuild.projects.polybench.polybench.**Mvt**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'mvt'

class benchbuild.projects.polybench.polybench.**Nussinov**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'nussinov'

class benchbuild.projects.polybench.polybench.**PolyBenchGroup**(*exp*)
Bases: *benchbuild.project.Project*

DOMAIN = 'polybench'

GROUP = 'polybench'

SRC_FILE = 'polybench-c-4.2.tar.gz'

VERSION = '4.2'

build()

configure()

download()

path_dict = {'correlation': 'datamining', 'covariance': 'datamining', '2mm': 'linear-algebra/kernels', '3mm': 'linear-a

src_dir = 'polybench-c-4.2'

src_uri = 'http://downloads.sourceforge.net/project/polybench/polybench-c-4.2.tar.gz'

class benchbuild.projects.polybench.polybench.**Seidel2D**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'seidel-2d'

class benchbuild.projects.polybench.polybench.**Symm**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'symm'

class benchbuild.projects.polybench.polybench.**Syr2k**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'syr2k'

class benchbuild.projects.polybench.polybench.**Syrk**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'syrk'

class benchbuild.projects.polybench.polybench.**ThreeMM**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = '3mm'

class benchbuild.projects.polybench.polybench.**Trisolv**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'trisolv'

class benchbuild.projects.polybench.polybench.**Trmm**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'trmm'

class benchbuild.projects.polybench.polybench.**TwoMM**(*exp*)
Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = '2mm'

benchbuild.reports package

Register reports for an experiment

class benchbuild.reports.**Report**(*exp_ids*, *out_path*)
Bases: object

SUPPORTED_EXPERIMENTS = None

class benchbuild.reports.**ReportRegistry**(*name*, *bases*, *dict*)
Bases: type

reports = {'raw': [<class 'benchbuild.reports.raw.RawReport'>]}

benchbuild.reports.**discover**()
Import all experiments listed in *_PLUGINS_REPORTS.

Tests:

```
>>> from benchbuild.settings import CFG
>>> from benchbuild.reports import discover
>>> import logging as lg
>>> import sys
>>> l = lg.getLogger('benchbuild')
>>> lg.getLogger('benchbuild').setLevel(lg.DEBUG)
>>> lg.getLogger('benchbuild').handlers = [lg.StreamHandler(stream=sys.
↪stdout)]
>>> CFG["plugins"]["reports"] = ["benchbuild.non.existing", "benchbuild.
↪reports.raw"]
>>> discover()
Could not find 'benchbuild.non.existing'
Found report: benchbuild.reports.raw
```

Submodules

benchbuild.reports.raw module

class benchbuild.reports.raw.**RawReport**(*exp_ids*, *outfile*)
Bases: *benchbuild.reports.Report*

SUPPORTED_EXPERIMENTS = ['raw']

generate()

get_exp_ids()

report()

session = <sqlalchemy.orm.session.Session object>

benchbuild.utils package

Submodules

benchbuild.utils.actions module

This defines classes that can be used to implement a series of Actions.

class benchbuild.utils.actions.**Any**(*actions*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Just run all actions, no questions asked.'

NAME = 'ANY'

class benchbuild.utils.actions.**Build**(*project*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Build the project'

NAME = 'BUILD'

class benchbuild.utils.actions.**Clean**(*project_or_experiment*, *action_fn=None*, *check_empty=False*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Cleans the build directory'

NAME = 'CLEAN'

class benchbuild.utils.actions.**CleanExtra**(*project_or_experiment*, *action_fn=None*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Cleans the extra directories.'

NAME = 'CLEAN EXTRA'

class benchbuild.utils.actions.**Configure**(*project*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Configure project source files'

NAME = 'CONFIGURE'

class benchbuild.utils.actions.**Download**(*project*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Download project source files'

NAME = 'DOWNLOAD'

class benchbuild.utils.actions.**Echo**(*message*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Print a message.'

NAME = 'ECHO'

class benchbuild.utils.actions.**Experiment**(*experiment*, *actions*)
Bases: *benchbuild.utils.actions.Any*

DESCRIPTION = 'Run a experiment, wrapped in a db transaction'

NAME = 'EXPERIMENT'

begin_transaction()

end_transaction(*experiment*, *session*)

class benchbuild.utils.actions.**MakeBuildDir**(*project_or_experiment*, *action_fn=None*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Create the build directory'

NAME = 'MKDIR'

class benchbuild.utils.actions.**Prepare**(*project*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Prepare project build folder'

NAME = 'PREPARE'

class benchbuild.utils.actions.**RequireAll**(*actions*)
Bases: *benchbuild.utils.actions.Step*

class benchbuild.utils.actions.**Run**(*project*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Execute the run action'

NAME = 'RUN'

class benchbuild.utils.actions.**Step**(*project_or_experiment*, *action_fn=None*)
Bases: object

DESCRIPTION = None

NAME = None

onerror()

class benchbuild.utils.actions.**StepClass**
Bases: abc.ABCMeta

class benchbuild.utils.actions.**StepResult**
Bases: enum.Enum

An enumeration.

ERROR = 2

OK = (1,)

benchbuild.utils.actions.**log_before_after**(*name*, *desc*)

benchbuild.utils.actions.**to_step_result**(*f*)

benchbuild.utils.bootstrap module

Helper functions for bootstrapping external dependencies.

benchbuild.utils.bootstrap.**check_uchroot_config**()

benchbuild.utils.bootstrap.**find_package**(*binary*)

benchbuild.utils.bootstrap.**install_package**(*pkg_name*)

benchbuild.utils.bootstrap.**install_uchroot**()

benchbuild.utils.bootstrap.**linux_distribution_major**()

benchbuild.utils.bootstrap.**provide_package**(*pkg_name*)

benchbuild.utils.bootstrap.**provide_packages**(*pkg_names*)

benchbuild.utils.compiler module

Helper functions for dealing with compiler replacement.

This provides a few key functions to deal with varying/measuring the compilers used inside the benchbuild study. From a high-level view, there are 2 interesting functions:

- lt_clang(cflags, ldflags, func)
- lt_clang_cxx(cflags, ldflags, func)

These generate a wrapped clang/clang++ in the current working directory and hide the given cflags/ldflags from the calling build system. Both will give you a working plumbum command and call a python script that redirects to the real clang/clang++ given the additional cflags&ldflags.

The wrapper-script generated for both functions can be found inside:

- wrap_cc()

The remaining methods:

- llvm()
- llvm_libs()
- clang()
- clang_cxx()

Are just convencience methods that can be used when interacting with the configured llvm/clang source directories.

benchbuild.utils.compiler.**clang**()
Get a usable clang plumbum command.

This searches for a usable clang in the llvm binary path (See llvm()) and returns a plumbum command to call it.

Returns plumbum Command that executes clang++

benchbuild.utils.compiler.**clang_cxx**()
Get a usable clang++ plumbum command.

This searches for a usable clang++ in the llvm binary path (See llvm()) and returns a plumbum command to call it.

Returns plumbum Command that executes clang++

benchbuild.utils.compiler.**llvm**()
Get the path where all llvm binaries can be found.

Environment variable: BB_LLVM_DIR

Returns LLVM binary path.

benchbuild.utils.compiler.**llvm_libs**()
Get the path where all llvm libraries can be found.

Environment variable: BB_LLVM_DIR

Returns LLVM library path.

benchbuild.utils.compiler.**lt_clang**(*cflags*, *ldflags*, *func=None*)
Return a clang that hides CFLAGS and LDFLAGS.

This will generate a wrapper script in the current directory and return a complete plumbum command to it.

Parameters

- **cflags** – The CFLAGS we want to hide.
- **ldflags** – The LDFLAGS we want to hide.
- **func** (*optional*) – A function that will be pickled alongside the compiler. It will be called before the actual compilation took place. This way you can intercept the compilation process with arbitrary python code.

Returns (benchbuild.utils.cmd): Path to the new clang command.

benchbuild.utils.compiler.**lt_clang_cxx**(*cflags*, *ldflags*, *func=None*)
Return a clang++ that hides CFLAGS and LDFLAGS.

This will generate a wrapper script in the current directory and return a complete plumbum command to it.

Parameters

- **cflags** – The CFLAGS we want to hide.
- **ldflags** – The LDFLAGS we want to hide.
- **func** (*optional*) – A function that will be pickled alongside the compiler. It will be called before the actual compilation took place. This way you can intercept the compilation process with arbitrary python code.

Returns (benchbuild.utils.cmd): Path to the new clang command.

benchbuild.utils.compiler.**wrap_cc_in_uchroot**(*cflags*, *ldflags*, *func=None*, *cc_name='clang'*)
Generate a clang wrapper that may be called from within a uchroot.

This basically does the same as lt_clang/lt_clang_cxx. However, we do not create a valid plumbum command. The generated script will only work inside a uchroot environment that has is root at the current working directory, when calling this function.

Parameters

- **cflags** – The CFLAGS we want to hide
- **ldflags** – The LDFLAGS we want to hide
- **func** (*optional*) – A function that will be pickled alongside the compiler. It will be called before the actual compilation took place. This way you can intercept the compilation process with arbitrary python code.
- **uchroot_path** – Prefix path of the compiler inside the uchroot.
- **cc_name** – Name of the generated script.

benchbuild.utils.compiler.**wrap_cxx_in_uchroot**(*cflags*, *ldflags*, *func=None*)
Delegate to wrap_cc_in_uchroot).

benchbuild.utils.container module

Container utilites.

class benchbuild.utils.container.**Container**
Bases: object

filename

local
Finds the current location of a container. Also unpacks the project if necessary.

Returns The path, where the container lies in the end.

Return type target

remote

class benchbuild.utils.container.**Gentoo**
Bases: *benchbuild.utils.container.Container*

latest_src_uri (**args*, ***kwargs*)

name = 'gentoo'

remote
Get a remote URL of the requested container.

class benchbuild.utils.container.**Ubuntu**
Bases: *benchbuild.utils.container.Container*

name = 'ubuntu'

remote
Get a remote URL of the requested container.

benchbuild.utils.container.**cached** (*func*)

benchbuild.utils.container.**is_valid_container** (*container*, *path*)
Checks if a container exists and is unpacked.

Parameters **path** – The location where the container is expected.

Returns True if the container is valid, False if the container needs to unpacked or if the path does not exist yet.

benchbuild.utils.container.**unpack_container** (*container*, *path*)
Method that checks if a directory for the container exists, checks if erlent support is needed and then unpacks the container accordingly.

Parameters **path** – The location where the container is, that needs to be unpacked.

benchbuild.utils.db module

Database support module for the benchbuild study.

benchbuild.utils.db.**create_run** (*cmd*, *prj*, *exp*, *grp*)
Create a new 'run' in the database.

This creates a new transaction in the database and creates a new run in this transaction. Afterwards we return both the transaction as well as the run itself. The user is responsible for committing it when the time comes.

Parameters

- **cmd** – The command that has been executed.
- **prj** – The project this run belongs to.
- **exp** – The experiment this run belongs to.
- **grp** – The run_group (uuid) we blong to.

Returns The inserted tuple representing the run and the session opened with the new run. Don't forget to commit it at some point.

benchbuild.utils.db.**create_run_group**(*prj*)
Create a new 'run_group' in the database.

This creates a new transaction in the database and creates a new run_group within this transaction. Afterwards we return both the transaction as well as the run_group itself. The user is responsible for committing it when the time comes.

Parameters **- The project for which we open the run_group.** (*prj*) –

Returns A tuple (group, session) containing both the newly created run_group and the transaction object.

benchbuild.utils.db.**persist_compilestats**(*run*, *session*, *stats*)
Persist the run results in the database.

Parameters

- **run** – The run we attach the compilestats to.
- **session** – The db transaction we belong to.
- **stats** – The stats we want to store in the database.

benchbuild.utils.db.**persist_config**(*run*, *session*, *cfg*)
Persist the configuration in as key-value pairs.

Parameters

- **run** – The run we attach the config to.
- **session** – The db transaction we belong to.
- **cfg** – The configuration we want to persist.

benchbuild.utils.db.**persist_experiment**(*experiment*)
Persist this experiment in the benchbuild database.

Parameters **experiment** – The experiment we want to persist.

benchbuild.utils.db.**persist_likwid**(*run*, *session*, *measurements*)
Persist all likwid results.

Parameters

- **run** – The run we attach our measurements to.
- **session** – The db transaction we belong to.
- **measurements** – The likwid measurements we want to store.

benchbuild.utils.db.**persist_perf**(*run*, *session*, *svg_path*)
Persist the flamegraph in the database.

The flamegraph exists as a SVG image on disk until we persist it in the database.

Parameters

- **run** – The run we attach these perf measurements to.
- **session** – The db transaction we belong to.
- **svg_path** – The path to the SVG file we want to store.

benchbuild.utils.db.**persist_project**(*project*)
Persist this project in the benchbuild database.

Parameters **project** – The project we want to persist.

benchbuild.utils.db.**persist_time**(*run*, *session*, *timings*)
Persist the run results in the database.

Parameters

- **run** – The run we attach this timing results to.
- **session** – The db transaction we belong to.
- **timings** – The timing measurements we want to store.

benchbuild.utils.downloader module

Downloading helper functions for benchbuild.

The helpers defined in this module provide access to some common Downloading methods for the source code of benchbuild projects. All downloads will be cached in BB_TMP_DIR and locked down with a hash that is generated after the first download. If the hash matches the file/folder found in BB_TMP_DIR, nothing will be downloaded at all.

Supported methods: Copy, CopyNoFail, Wget, Git, Svn, Rsync

benchbuild.utils.downloader.**Copy**(*From*, *To*)
Small copy wrapper.

Parameters

- **From** (*str*) – Path to the SOURCE.
- **To** (*str*) – Path to the TARGET.

benchbuild.utils.downloader.**CopyNoFail**(*src*, *root=None*)
Just copy fName into the current working directory, if it exists.

No action is executed, if fName does not exist. No Hash is checked.

Parameters

- **src** – The filename we want to copy to '.'.
- **root** – The optional source dir we should pull fName from. Defaults to benchbuild.settings.CFG["tmpdir"].

Returns True, if we copied something.

benchbuild.utils.downloader.**Git**(*src_url*, *tgt_name*, *tgt_root=None*)
Get a shallow clone of the given repo

Parameters

- **src_url** (*str*) – Git URL of the SOURCE repo.
- **tgt_name** (*str*) – Name of the repo folder on disk.
- **tgt_root** (*str*) – TARGET folder for the git repo. Defaults to `CFG["tmpdir"]`

benchbuild.utils.downloader.**Rsync**(*url*, *tgt_name*, *tgt_root=None*)
RSync a folder.

Parameters

- **url** (*str*) – The url of the SOURCE location.
- **fname** (*str*) – The name of the TARGET.
- **to** (*str*) – Path of the target location. Defaults to `CFG["tmpdir"]`.

benchbuild.utils.downloader.**Svn**(*url*, *fname*, *to=None*)
Checkout the SVN repo.

Parameters

- **url** (*str*) – The SVN SOURCE repo.
- **fname** (*str*) – The name of the repo on disk.
- **to** (*str*) – The name of the TARGET folder on disk. Defaults to `CFG["tmpdir"]`

benchbuild.utils.downloader.**Wget**(*src_url*, *tgt_name*, *tgt_root=None*)
Download url, if required.

Parameters

- **src_url** (*str*) – Our SOURCE url.
- **tgt_name** (*str*) – The filename we want to have on disk.
- **tgt_root** (*str*) – The TARGET directory for the download. Defaults to `CFG["tmpdir"]`.

benchbuild.utils.downloader.**get_hash_of_dirs**(*directory*)
Recursively hash the contents of the given directory.

Parameters **directory** (*str*) – The root directory we want to hash.

Returns A hash of all the contents in the directory.

benchbuild.utils.downloader.**source_required**(*src_file*, *src_root*)
Check, if a download is required.

Parameters

- **src_file** – The filename to check for.
- **src_root** – The path we find the file in.

Returns True, if we need to download something, False otherwise.

benchbuild.utils.downloader.**update_hash**(*src*, *root*)
Update the hash for the given file.

Parameters

- **src** – The file name.
- **root** – The path of the given file.

benchbuild.utils.log module

benchbuild.utils.log.**configure**()
Load logging configuration from our own defaults.

benchbuild.utils.log.**set_defaults**()
Configure the loggers default settings.

benchbuild.utils.path module

Path utilities for benchbuild.

benchbuild.utils.path.**determine_path**()
Borrowed from wxglade.py

benchbuild.utils.path.**list_to_path**(*pathlist*)
Convert a list of path elements to a path string.

benchbuild.utils.path.**mkdir_uchroot**(*dirpath*, *root='.'*)
Create a file inside a uchroot env.

You will want to use this when you need to create a file with apropriate rights inside a uchroot container with subuid/subgid handling enabled.

Parameters

- **dirpath** – The dirpath that should be created. Absolute inside the uchroot container.
- **root** – The root PATH of the container filesystem as seen outside of the container.

benchbuild.utils.path.**mkfile_uchroot**(*filepath*, *root='.'*)
Create a file inside a uchroot env.

You will want to use this when you need to create a file with apropriate rights inside a uchroot container with subuid/subgid handling enabled.

Parameters

- **filepath** – The filepath that should be created. Absolute inside the uchroot container.
- **root** – The root PATH of the container filesystem as seen outside of the container.

benchbuild.utils.path.**path_to_list**(*pathstr*)
Conver a path string to a list of path elements.

benchbuild.utils.path.**template_str**(*template*)
Read a template file from the resources and return it as str.

benchbuild.utils.run module

Experiment helpers.

exception benchbuild.utils.run.**GuardedRunException**(*what*, *db_run*, *session*)
Bases: Exception

BB Run exception.

Contains an exception that ocurred during execution of a benchbuild experiment.

benchbuild.utils.run.**begin**(*command*, *pname*, *ename*, *group*)
Begin a run in the database log.

Parameters

- **command** – The command that will be executed.
- **pname** – The project name we belong to.
- **ename** – The experiment name we belong to.
- **group** – The run group we belong to.

Returns (run, session), where run is the generated run instance and session the associated transaction for later use.

benchbuild.utils.run.**begin_run_group**(*project*)
Begin a run_group in the database.

A run_group groups a set of runs for a given project. This models a series of runs that form a complete binary runtime test.

Parameters **project** – The project we begin a new run_group for.

Returns `(group, session)` where group is the created group in the database and session is the database session this group lives in.

benchbuild.utils.run.**end**(*db_run*, *session*, *stdout*, *stderr*)
End a run in the database log (Successfully).

This will persist the log information in the database and commit the transaction.

Parameters

- **db_run** – The `run` schema object we belong to
- **session** – The db transaction we belong to.
- **stdout** – The stdout we captured of the run.
- **stderr** – The stderr we capture of the run.

benchbuild.utils.run.**end_run_group**(*group*, *session*)
End the run_group successfully.

Parameters

- **group** – The run_group we want to complete.
- **session** – The database transaction we will finish.

benchbuild.utils.run.**fail**(*db_run*, *session*, *retcode*, *stdout*, *stderr*)
End a run in the database log (Unsuccessfully).

This will persist the log information in the database and commit the transaction.

Parameters

- **db_run** – The `run` schema object we belong to
- **session** – The db transaction we belong to.
- **retcode** – The return code we captured of the run.
- **stdout** – The stdout we captured of the run.
- **stderr** – The stderr we capture of the run.

benchbuild.utils.run.**fail_run_group**(*group*, *session*)
End the run_group unsuccessfully.

Parameters

- **group** – The run_group we want to complete.
- **session** – The database transaction we will finish.

benchbuild.utils.run.**fetch_time_output**(*marker*, *format_s*, *ins*)
Fetch the output /usr/bin/time from a.

Parameters

- **marker** – The marker that limits the time output
- **format_s** – The format string used to parse the timings

- **ins** – A list of lines we look for the output.

Returns A list of timing tuples

benchbuild.utils.run.**guarded_exec**(*cmd*, *project*, *experiment*, ***kwargs*)
Guard the execution of the given command.

Parameters

- **cmd** – the command we guard.
- **pname** – the database we run under.
- **ename** – the database session this run belongs to.
- **run_group** – the run group this execution will belong to.

Raises RunException – If the cmd encounters an error we wrap the exception in a RunException and re-raise. This ends the run unsuccessfully.

benchbuild.utils.run.**handle_stdin**(*cmd*, *kwargs*)
Handle stdin for wrapped runtime executors.

This little helper checks the kwargs for a *has_stdin* key containing a boolean value. If necessary it will pipe in the stdin of this process into the plumbum command.

Parameters

- **cmd** (*benchbuild.utils.cmd*) – Command to wrap a stdin handler around.
- **kwargs** – Dictionary containing the kwargs. We check for they key *has_stdin*

Returns A new plumbum command that deals with stdin redirection, if needed.

benchbuild.utils.run.**in_builddir**(*sub='.'*)
Decorate a project phase with a local working directory change.

Parameters **sub** – An optional subdirectory to change into.

benchbuild.utils.run.**run**(*command*, *retcode=0*)
Execute a plumbum command, depending on the user's settings.

Parameters **command** – The plumbumb command to execute.

benchbuild.utils.run.**store_config**(*func*)
Decorator for storing the configuration in the project's builddir.

benchbuild.utils.run.**uchroot**(**args*, ***kwargs*)
Return a customizable uchroot command.

Parameters **args** – List of additional arguments for uchroot (typical: mounts)

Returns chroot_cmd

benchbuild.utils.run.**uchroot_env**(*mounts*)
Compute the environment of the change root for the user.

Parameters **mounts** – The mountpoints of the current user.

Returns paths ld_libs

benchbuild.utils.run.**uchroot_mounts**(*prefix*, *mounts*)
Compute the mountpoints of the current user.

Parameters

- **prefix** – Define where to job was running if it ran on a cluster.

- **mounts** – All mounts the user currently uses in his file system.

Returns mntpoints

benchbuild.utils.run.**uchroot_no_args**()
Return the uchroot command without any customizations.

benchbuild.utils.run.**uchroot_no_llvm**(**args*, ***kwargs*)
Return a customizable uchroot command.

The command will be executed inside a uchroot environment.

Parameters **args** – List of additional arguments for uchroot (typical: mounts)

Returns chroot_cmd

benchbuild.utils.run.**uchroot_with_mounts**(**args*, ***kwargs*)
Return a uchroot command with all mounts enabled.

benchbuild.utils.run.**unionfs**(*base_dir='./base'*, *image_dir='./image'*, *image_prefix=None*, *mountpoint='./union'*)
Decorator for the UnionFS feature.

This configures a unionfs for projects. The given base_dir and/or image_dir are layered as follows:

image_dir=RW:base_dir=RO

All writes go to the image_dir, while base_dir delivers the (read-only) versions of the rest of the filesystem.

The unified version will be provided in the project's builddir. Unmouting is done as soon as the function completes.

Parameters

- **base_dir** – The unpacked container of a project delievered by a method out of the container utils.
- **image_dir** – Virtual image of the actual file system represented by the build_dir of a project.
- **image_prefix** – Useful prefix if the projects run on a cluster, to identify where the job came from and where it runs.
- **mountpoint** – Location where the filesystems merge, currently per default as './union'.

benchbuild.utils.run.**unionfs_set_up**(*ro_base*, *rw_image*, *mountpoint*)
Setup a unionfs via unionfs-fuse.

Parameters

- **ro_base** – base_directory of the project
- **rw_image** – virtual image of actual file system
- **mountpoint** – location where ro_base and rw_image merge

benchbuild.utils.run.**unionfs_tear_down**(*mountpoint*, *tries=3*)
Tear down a unionfs mountpoint.

benchbuild.utils.run.**with_env_recursive**(*cmd*, ***envvars*)
Recursively updates the environment of cmd and all its subcommands.

Parameters

- **- A plumbum command-like object**(*cmd*) –
- **- The environment variables to update**(***envvars*) –

Returns The updated command.

benchbuild.utils.schema module

Database schema for benchbuild.

benchbuild.utils.schema.**CONNECTION_MANAGER = <benchbuild.utils.schema.SessionManager object>**
Import this session manager to create new database sessions as needes.

class benchbuild.utils.schema.**CompileStat**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store compilestats as given by LLVM's '-stats' commoand.

component

id

name

run_id

value

class benchbuild.utils.schema.**Config**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store customized information about a run.

You can store arbitrary configuration information about a run here. Use it for extended filtering against the run table.

name

run_id

value

class benchbuild.utils.schema.**Event**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store PAPI profiling based events.

duration

id

name

run_id

start

tid

type

class benchbuild.utils.schema.**Experiment**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store metadata about experiments.

begin

description

end

id

name

class benchbuild.utils.schema.**GlobalConfig**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store customized information about a run.

You can store arbitrary configuration information about a run here. Use it for extended filtering against the run table.

experiment_group

name

value

class benchbuild.utils.schema.**Likwid**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store measurement results of likwid based experiments.

core

metric

region

run_id

value

class benchbuild.utils.schema.**Metadata**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store metadata information for every run.

If you happen to have some free-form data that belongs to the database, this is the place for it.

name

run_id

value

class benchbuild.utils.schema.**Metric**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store default metrics, simple name value store.

name

run_id

value

class benchbuild.utils.schema.**PerfEvent**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store PAPI profiling based events.

duration

id

name

run_id

start

tid

type

class benchbuild.utils.schema.**Project**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store project metadata.

description

domain

group_name

name

src_url

version

class benchbuild.utils.schema.**RegressionTest**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store regression tests for all projects.

This relation is filled from the PolyJIT side, not from benchbuild-study. We collect all JIT SCoPs found by PolyJIT in this relation for regression-test generation.

module

name

project_name

run_id

class benchbuild.utils.schema.**Run**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store a run for each executed test binary.

begin

command

end

experiment_group

experiment_name

id

project_name

run_group

status

class benchbuild.utils.schema.**RunGroup**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store information about a run group.

begin

end

experiment

id

project

status

class benchbuild.utils.schema.**RunLog**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store log information for every run.

Properties like, start time, finish time, exit code, stderr, stdout are stored here.

begin

config

end

run_id

status

stderr

stdout

class benchbuild.utils.schema.**SessionManager**
Bases: object

get()

benchbuild.utils.slurm module

SLURM support for the benchbuild study.

This module can be used to generate bash scripts that can be executed by the SLURM controller either as batch or interactive script.

benchbuild.utils.slurm.**dump_slurm_script**(*script_name*, *benchbuild*, *experiment*, *projects*)
Dump a bash script that can be given to SLURM.

Parameters

- **script_name** (*str*) – name of the bash script.
- **commands** (*list (benchbuild.utils.cmd)*) – List of plumbum commands to write to the bash script.
- ****kwargs** – Dictionary with all environment variable bindings we should map in the bash script.

benchbuild.utils.slurm.**prepare_directories**(*dirs*)
Make sure that the required directories exist.

Parameters - the directories we want. (*dirs*) –

benchbuild.utils.slurm.**prepare_slurm_script**(*experiment*, *projects*)
Prepare a slurm script that executes the experiment for a given project.

Parameters

- **experiment** – The experiment we want to execute

- **projects** – All projects we generate an array job for.

benchbuild.utils.user_interface module

User interface helpers for benchbuild.

benchbuild.utils.user_interface.**ask**(*question*, *default_answer=False*, *default_answer_str='no'*)

benchbuild.utils.user_interface.**query_yes_no**(*question*, *default='yes'*)

Ask a yes/no question via raw_input() and return their answer.

Parameters

- **question** (*str*) – Question hat is presented to the user.
- **default** (*str*) – The presumed answer, if the user just hits <Enter>. It must be "yes" (the default), "no" or None (meaning an answer is required of the user).

Returns (boolean): True, if 'yes', False otherwise.

benchbuild.utils.versions module

Gather version information for BB.

benchbuild.utils.versions.**get_git_hash**(*from_url*)

Get the git commit hash of HEAD from :from_url.

Parameters **from_url** – The file system url of our git repository.

Returns git commit hash of HEAD, or empty string.

benchbuild.utils.versions.**get_version_from_cache_dir**(*src_file*)

Creates a version for a project out of the hash.

The hash is taken from the directory of the source file.

Parameters **src_file** – The source file of the project using this function.

Returns Either returns the first 8 digits of the hash as string, the entire hash as a string if the hash consists out of less than 7 digits or None if the path is incorrect.

benchbuild.utils.wrapping module

benchbuild.utils.wrapping.**strip_path_prefix**(*ipath*, *prefix*)

Strip prefix from path.

Parameters

- **ipath** – input path
- **prefix** – the prefix to remove, if it is found in :ipath:

Examples

```
>>> strip_path_prefix("/foo/bar", "/bar")
'/foo/bar'
>>> strip_path_prefix("/foo/bar", "/")
'foo/bar'
>>> strip_path_prefix("/foo/bar", "/foo")
'/bar'
>>> strip_path_prefix("/foo/bar", "None")
'/foo/bar'
```

benchbuild.utils.wrapping.**wrap**(*name*, *runner*, *sprefix=None*, ***template_vars*)

Wrap the binary :name: with the function :runner:.

This module generates a python tool that replaces :name: The function in runner only accepts the replaced binaries name as argument. We use the cloudpickle package to perform the serialization, make sure :runner: can be serialized with it and you're fine.

Parameters

- **name** – Binary we want to wrap
- **runner** – Function that should run instead of :name:

Returns A plumbum command, ready to launch.

benchbuild.utils.wrapping.**wrap_cc**(*filepath*, *cflags*, *ldflags*, *compiler*, *extension*, *compiler_ext_name=None*, ***template_vars*)

Substitute a compiler with a script that hides CFLAGS & LDFLAGS.

This will generate a wrapper script in the current directory and return a complete plumbum command to it.

Parameters

- **filepath** (*str*) – Path to the wrapper script.
- **cflags** (*list (str)*) – The CFLAGS we want to hide.
- **ldflags** (*list (str)*) – The LDFLAGS we want to hide.
- **compiler** (*benchbuild.utils.cmd*) – Real compiler command we should call in the script.
- **extension** – A function that will be pickled alongside the compiler. It will be called before the actual compilation took place. This way you can intercept the compilation process with arbitrary python code.
- **compiler_ext_name** – The name that we should give to the generated dill blob for :func:

Returns (benchbuild.utils.cmd): Command of the new compiler we can call.

benchbuild.utils.wrapping.**wrap_dynamic**(*self*, *name*, *runner*, *sprefix=None*, ***template_vars*)

Wrap the binary :name with the function :runner.

This module generates a python tool :name: that can replace a yet unspecified binary. It behaves similar to the :wrap: function. However, the first argument is the actual binary name.

Parameters

- **name** – name of the python module
- **runner** – Function that should run the real binary
- **base_class** – The base_class of our project.

- **base_module** – The module of base_class.

Returns: plumbum command, readty to launch.

benchbuild.utils.wrapping.**wrap_dynamic_in_uchroot**(*self*, *name*, *runner*, *sprefix=None*)

benchbuild.utils.wrapping.**wrap_in_uchroot**(*name*, *runner*, *sprefix=None*)

1.9.1.2 Submodules

1.9.1.3 benchbuild.bootstrap module

class benchbuild.bootstrap.**BenchBuildBootstrap**(*executable*)
Bases: plumbum.cli.application.Application

Bootstrap benchbuild external dependencies, if possible.

main(**args*)

store_config
Sets an attribute

1.9.1.4 benchbuild.container module

class benchbuild.container.**BashStrategy**
Bases: *benchbuild.container.ContainerStrategy*

The user interface for setting up a bash inside the container.

run(*context*)

class benchbuild.container.**Container**(*exe*)
Bases: plumbum.cli.application.Application

Manage uchroot containers.

VERSION = '1.2.1'

builddir(*tmpdir*)
Set the current builddir of the container.

input_file(*container*)
Find the input path of a uchroot container.

main(**args*)

mounts(*user_mount*)
Save the current mount of the container into the settings.

output_file(*container*)
Find and writes the output path of a chroot container.

shell(*custom_shell*)
The command to run inside the container.

verbosity
Sets an attribute

class benchbuild.container.**ContainerBootstrap**(*executable*)
Bases: plumbum.cli.application.Application

Check for the needed files.

install_cmake_and_exit()
Tell the user to install cmake and aborts the current process.

main(**args*)

class benchbuild.container.**ContainerCreate**(*executable*)
Bases: plumbum.cli.application.Application

Create a new container with a predefined strategy.

We offer a variety of creation policies for a new container. By default a basic 'spawn a bash' policy is used. This just leaves you inside a bash that is started in the extracted container. After customization you can exit the bash and pack up the result.

main(**args*)

strategy(*strategy*)

class benchbuild.container.**ContainerList**(*executable*)
Bases: plumbum.cli.application.Application

Prints a list of the known containers.

main(**args*)

class benchbuild.container.**ContainerRun**(*executable*)
Bases: plumbum.cli.application.Application

Execute commannds inside a prebuilt container.

main(**args*)

class benchbuild.container.**ContainerStrategy**
Bases: object

Interfaces for the different containers chosen by the experiment.

run(*context*)

class benchbuild.container.**MockObj**(***kwargs*)
Bases: object

class benchbuild.container.**SetupPolyJITGentooStrategy**
Bases: *benchbuild.container.ContainerStrategy*

Interface of using gentoo as a container for an experiment.

configure()
Configure the gentoo container for a PolyJIT experiment.

run(*context*)
Setup a gentoo container suitable for PolyJIT.

write_bashrc(*path*)
Write inside a bash and update the shell if necessary.

write_layout(*path*)
Create a layout from the given path.

write_makeconfig(*path*)
Create the stringed to be written in the settings.

write_wgetrc(*path*)
Wget the project from a specified link.

benchbuild.container.**clean_directories**(*builddir*, *in_dir=True*, *out_dir=True*)
Remove the in and out of the container if confirmed by the user.

benchbuild.container.**find_hash**(*container_db*, *key*)
Find the first container in the database with the given key.

benchbuild.container.**main**(**args*)

benchbuild.container.**pack_container**(*in_container*, *out_file*)

benchbuild.container.**run_in_container**(*command*, *container_dir*, *mounts*)
Run a given command inside a container.

Mounts a directory as a container at the given mountpoint and tries to run the given command inside the new container.

benchbuild.container.**set_input_container**(*container*, *cfg*)
Save the input for the container in the configurations.

benchbuild.container.**setup_bash_in_container**(*builddir*, *container*, *outfile*, *mounts*, *shell*)
Setup a bash environment inside a container.

Creates a new chroot, which the user can use as a bash to run the wanted projects inside the mounted container, that also gets returned afterwards.

benchbuild.container.**setup_container**(*builddir*, *container*)
Prepare the container and returns the path where it can be found.

benchbuild.container.**setup_directories**(*builddir*)
Create the in and out directories of the container.

1.9.1.5 benchbuild.driver module

class benchbuild.driver.**PollyProfiling**(*executable*)
Bases: plumbum.cli.application.Application

Frontend for running/building the benchbuild study framework.

VERSION = '1.2.1'

main(**args*)

verbosity
Sets an attribute

benchbuild.driver.**main**(**args*)
Main function.

1.9.1.6 benchbuild.experiment module

BenchBuild's skeleton for experiments.

An benchbuild.experiment defines a series of phases that constitute a benchbuild compatible experiment. This is the default implementation of an experiment.

Clients can derive from class class::benchbuild.experiment.Experiment and override the methods relvant to their experiment.

An experiment can have a variable number of phases / steps / substeps.

Phases / Steps / Substeps

All phases/steps/substeps support being used as a context manager. All 3 of them catch ProcessExecutionErrors that may be thrown from plumbum, without aborting the whole experiment. However, an error is tracked.

Actions

An experiment performs the following actions in order:

1. clean - Clean any previous runs that collide with our directory
2. **prepare - Prepare the experiment, this is a good place to copy relevant** files over for testing.
3. **run (run_tests) - Run the experiment. The 'meat' lies here. Override** This to perform all your experiment needs.

class benchbuild.experiment.**Experiment**(*projects=None*, *group=None*)
Bases: object

A series of commands executed on a project that form an experiment.

The default implementation should provide a sane environment for all derivates.

One important task executed by the basic implementation is setting up the default set of projects that belong to this project. As every project gets registered in the ProjectFactory, the experiment gets a list of experiment names that work as a filter.

NAME = None

actions()

actions_for_project(*project*)
Get the actions a project wants to run.

Parameters **project** (*benchbuild.Project*) – the project we want to run.

populate_projects(*projects_to_filter=None*, *group=None*)
Populate the list of projects that belong to this experiment.

Parameters

- **projects_to_filter** (*list*) – List of projects we want to assign to this experiment. We intersect the list of projects with the list of supported projects to get the list of projects that belong to this experiment.
- **group** (*str*) – In addition to the project filter, we provide a way to filter whole groups.

class benchbuild.experiment.**ExperimentRegistry**(*name*, *bases*, *dict*)
Bases: type

Registry for benchbuild experiments.

experiments = {'cs': <class 'benchbuild.experiments.compilestats.CompilestatsExperiment'>, 'p-cs': <class 'benchbuil

class benchbuild.experiment.**RuntimeExperiment**(*projects=None*, *group=None*)
Bases: *benchbuild.experiment.Experiment*

Additional runtime only features for experiments.

get_papi_calibration(*project*, *calibrate_call*)
Get calibration values for PAPI based measurements.

Parameters

- **project** (Project) – Unused (deprecated).
- **calibrate_call** (*benchbuild.utils.cmd*) – The calibration command we will use.

persist_calibration(*project*, *cmd*, *calibration*)
Persist the result of a calibration call.

Parameters

- **project** (*benchbuild.Project*) – The calibration values will be assigned to this project.
- **cmd** (*benchbuild.utils.cmd*) – The command we used to generate the calibration values.
- **calibration** (*int*) – The calibration time in nanoseconds.

benchbuild.experiment.**get_group_projects**(*group*, *experiment*)
Get a list of project names for the given group.

Filter the projects assigned to this experiment by group.

Parameters

- **group** (*str*) – The group.
- **experiment** (*benchbuild.Experiment*) – The experiment we draw our projects to filter from.

Returns (list): A list of project names for the group that are supported by this experiment.

benchbuild.experiment.**newline**(*ostream*)
Break the current line in the output stream.

Don't reuse the current line, if :o: is not attached to a tty.

Parameters **o** (*stream*) – The stream we insert a newline.

Returns (stream): The stream

benchbuild.experiment.**phase**(*name*, *pname='FIXME: Unset'*, *cleaner=None*)
Introduce a new phase.

This just introduces a new (cosmetic) phase distinction between different experiment phases. This can be used as a contextmanager to distinguish different experiment phases.

Parameters

- **name** (*str*) – Name of the phase.
- **pname** (*str*) – Project Name this phase will be started for.
- **cleaner** (*callable*) – Cleaner callable that will be invoked as soon as the phase failed and we cannot recover.

benchbuild.experiment.**static_var**(*varname*, *value*)
Decorate something with a static variable.

Example

```
@staticvar(bar, 0)
def foo():
    foo.bar = 1
    return foo.bar
```

Parameters

- **varname** (*str*) – The name of the static variable.
- **value** – The initial value of the static variable.

Returns A decorator that adds a new attribute to the given object.

benchbuild.experiment.**step**(*name*)
Introduce a new step.

This just introduces a new (cosmetic) step distinction between different experiment steps. This can be used as a contextmanager to distinguish different experiment steps.

Parameters **name** (*str*) – The name of the step

benchbuild.experiment.**substep**(*name*)
Introduce a new substep.

This just introduces a new (cosmetic) substep distinction between different experiment steps. This can be used as a contextmanager to distinguish different experiment steps.

Parameters **name** (*str*) – The name of the substep

1.9.1.7 benchbuild.likwid module

Likwid helper functions.

Extract information from likwid's CSV output.

benchbuild.likwid.**fetch_cols**(*fstream*, *split_char=','*)
Fetch columns from likwid's output stream.

Parameters

- **fstream** – The filestream with likwid's output.
- **split_car** (*str*) – The character we split on, default ','

Returns (list(str)): A list containing the elements of fstream, after splitting at split_char.

benchbuild.likwid.**get_likwid_perfctr**(*infile*)
Get a complete list of all measurements.

Parameters **infile** – The filestream containing all likwid output.

Returns A list of all measurements extracted from likwid's file stream.

benchbuild.likwid.**get_measurements**(*region*, *core_info*, *data*, *extra_offset=0*)
Get the complete measurement info from likwid's region info.

Parameters

- **region** – The region we took a measurement in.
- **core_info** – The core information.

- **data** – The raw data.
- **extra_offset** (*int*) – default = 0

Returns (list((region, metric, core, value))): A list of measurement tuples, a tuple contains the information about the region, the metric, the core and the actual value.

benchbuild.likwid.**read_struct**(*fstream*)
Read a likwid struct from the text stream.

Parameters **fstream** – Likwid's filestream.

Returns (dict(str: str)): A dict containing all likwid's struct info as key/value pairs.

benchbuild.likwid.**read_structs**(*fstream*)
Read all structs from likwid's file stream.

Parameters **fstream** – Likwid's output file stream.

Returns A generator that can be used to iterate over all structs in the fstream.

benchbuild.likwid.**read_table**(*fstream*)
Read a likwid table info from the text stream.

Parameters **fstream** – Likwid's filestream.

Returns (dict(str: str)): A dict containing likwid's table info as key/value pairs.

benchbuild.likwid.**read_tables**(*fstream*)
Read all tables from likwid's file stream.

Parameters **fstream** – Likwid's output file stream.

Returns A generator that can be used to iterate over all tables in the fstream.

1.9.1.8 benchbuild.log module

Analyze the BB database.

class benchbuild.log.**BenchBuildLog**(*executable*)
Bases: plumbum.cli.application.Application

Frontend command to the benchbuild database.

experiment(*experiments*)
Set the experiments to fetch the log for.

experiment_ids(*experiment_ids*)
Set the experiment ids to fetch the log for.

log_type(*types*)
Set the output types to print.

main()
Run the log command.

project(*projects*)
Set the projects to fetch the log for.

project_ids(*project_ids*)
Set the project ids to fetch the log for.

benchbuild.log.**print_logs**(*query*, *types=None*)
Print status logs.

benchbuild.log.**print_runs**(*query*)
Print all rows in this result query.

1.9.1.9 benchbuild.project module

Project handling for the benchbuild study.

class benchbuild.project.**Project**(*exp*, *group=None*)
Bases: object

benchbuild's Project class.

A project defines how run-time testing and cleaning is done for this IR project

CONTAINER = <benchbuild.utils.container.Gentoo object>

DOMAIN = None

GROUP = None

NAME = None

SRC_FILE = None

VERSION = None

build()
Build the project.

clean()
Clean the project build directory.

compiler_extension
Return the compiler extension registered to this project.

configure()
Configure the project.

download()
Download the input source for this project.

prepare()
Prepare the build diretory.

run(*experiment*)
Run the tests of this project.

This method initializes the default environment and takes care of cleaning up the mess we made, after a successfull run.

Parameters **experiment** – The experiment we run this project under

run_tests(*experiment*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters **experiment** – The experiment we run this project under

run_uuid
Get the UUID that groups all tests for one project run.

Parameters **create_new** – Create a fresh UUID (Default: False)

runtime_extension
Return the runtime extension registered for this project.

setup_derived_filenames()
Construct all derived file names.

class benchbuild.project.**ProjectDecorator**(*name*, *bases*, *attrs*)
Bases: *benchbuild.project.ProjectRegistry*

Decorate the interface of a project with the in_builddir decorator.

This is just a small safety net for benchbuild users, because we make sure to run every project method in the project's build directory.

decorated_methods = ['build', 'configure', 'download', 'prepare', 'run_tests']

class benchbuild.project.**ProjectRegistry**(*name*, *bases*, *attrs*)
Bases: type

Registry for benchbuild projects.

projects = {'bzip2': <class 'benchbuild.projects.benchbuild.bzip2.Bzip2'>, 'ccrypt': <class 'benchbuild.projects.benchb

1.9.1.10 benchbuild.report module

class benchbuild.report.**BenchBuildReport**(*executable*)
Bases: plumbum.cli.application.Application

Generate Reports from the benchbuild db.

experiment_ids(*ids*)

experiments(*experiments*)

main(**args*)

outfile(*outfile*)

1.9.1.11 benchbuild.run module

benchbuild's run command.

This subcommand executes experiments on a set of user-controlled projects. See the output of benchbuild run –help for more information.

class benchbuild.run.**BenchBuildRun**(*executable*)
Bases: plumbum.cli.application.Application

Frontend for running experiments in the benchbuild study framework.

experiment_tag(*description*)

experiments(*experiments*)

group(*group*)

list_experiments()

list_projects()

main()
Main entry point of benchbuild run.

pretend
Sets an attribute

projects (*projects*)

show_config
Sets an attribute

store_config
Sets an attribute

benchbuild.run.**print_projects**(*exp*)
Print a list of projects registered for that experiment.

Parameters **exp** – The experiment to print all projects for.

1.9.1.12 benchbuild.settings module

Settings module for benchbuild.

All settings are stored in a simple dictionary. Each setting should be modifiable via environment variable.

class benchbuild.settings.**Configuration**(*parent_key*, *node=None*, *parent=None*, *init=True*)
Bases: object

Dictionary-like data structure to contain all configuration variables.

This serves as a configuration dictionary throughout benchbuild. You can use it to access all configuration options that are available. Whenever the structure is updated with a new subtree, all variables defined in the new subtree are updated from the environment.

Environment variables are generated from the tree paths automatically. CFG["build_dir"] becomes BB_BUILD_DIR CFG["llvm"]["dir"] becomes BB_LLVM_DIR

The configuration can be stored/loaded as JSON.

Examples

```
>>> from benchbuild import settings as s
>>> c = s.Configuration('bb')
>>> c['test'] = 42
>>> c['test']
BB_TEST=42
>>> str(c['test'])
'42'
>>> type(c['test'])
<class 'benchbuild.settings.Configuration'>
```

filter_exports()

has_default()
Check, if the node contains a 'default' value.

has_value()
Check, if the node contains a 'value'.

init_from_env()
Initialize this node from environment.

If we're a leaf node, i.e., a node containing a dictionary that consist of a 'default' key, compute our env variable and initialize our value from the environment. Otherwise, init our children.

is_leaf()
Check, if the node is a 'leaf' node.

load(*_from*)
Load the configuration dictionary from file.

store(*config_file*)
Store the configuration dictionary to a file.

update(*cfg_dict*)
Update the configuration dictionary with new content.

This just delegates the update down to the internal data structure. No validation is done on the format, be sure you know what you do.

Parameters **cfg_dict** – A configuration dictionary.

value()
Return the node value, if we're a leaf node.

Examples

```
>>> from benchbuild import settings as s
>>> c = s.Configuration("test")
>>> c['x'] = { "y" : { "value" : None }, "z" : { "value" : 2 }}
>>> c['x']['y'].value() == None
True
>>> c['x']['z'].value()
2
>>> c['x'].value()
TEST_X_Y=null
TEST_X_Z=2
```

exception benchbuild.settings.**InvalidConfigKey**
Bases: RuntimeWarning

Warn, if you access a non-existing key benchbuild's configuration.

class benchbuild.settings.**UUIDEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)
Bases: json.encoder.JSONEncoder

Encoder module for UUID objects.

default(*obj*)
Encode UUID objects as string.

benchbuild.settings.**available_cpu_count**()
Get the number of available CPUs.

Number of available virtual or physical CPUs on this system, i.e. user/real as output by time(1) when called with an optimally scaling userspace-only program.

Returns Number of avaialable CPUs.

benchbuild.settings.**escape_json**(*raw_str*)
Shell-Escape a json input string.

Parameters **raw_str** – The unescaped string.

benchbuild.settings.**find_config**(*default='.benchbuild.json'*, *root='.'*)
Find the path to the default config file.

We look at :root: for the :default: config file. If we can't find it there we start looking at the parent directory recursively until we find a file named :default: and return the absolute path to it. If we can't find anything, we return None.

Parameters

- **default** – The name of the config file we look for.
- **root** – The directory to start looking for.

Returns Path to the default config file, None if we can't find anything.

benchbuild.settings.**to_env_dict**(*config*)
Convert configuration object to a flat dictionary.

benchbuild.settings.**update_env**()

1.9.1.13 benchbuild.slurm module

Dump SLURM script that executes the selected experiment with all projects.

This basically provides the same as benchbuild run, except that it just dumps a slurm batch script that executes everything as an array job on a configurable SLURM cluster.

class benchbuild.slurm.**Slurm**(*executable*)
Bases: plumbum.cli.application.Application

Generate a SLURM script.

experiment(*cfg_experiment*)
Specify experiments to run

experiment_tag(*description*)
A description for this experiment run

group(*groups*)
Run a group of projects under the given experiments

main()
Main entry point of benchbuild run.

projects(*projects*)
Specify projects to run

1.9.1.14 benchbuild.test module

class benchbuild.test.**BenchBuildTest**(*executable*)
Bases: plumbum.cli.application.Application

Create regression tests for polyjit from the measurements database.

get_check_line(*name*, *module*)

main()

opt_flags()

prefix(*prefix*)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X