testsuite for XAMG

This document contains a quick reference on running the Continous Integration suite for the XAMG project.

Getting the source code of testsuite

The source code of the test suite is available as a public repository at `github.com`. The URL of the repository is: *https://github.com/a-v-medvedev/testsuite.git*

NOTE: it is important to add the `--recursive` flag for a `git clone` command when you get the source code for the first time:

```
git clone --recursive https://github.com/a-v-medvedev/testsuite.git
```

If the `git clone` command has been already done without this option in place, you may init the sub-modules with a separate `git submodule update...` command later:

```
git clone https://github.com/a-v-medvedev/testsuite.git
cd XAMG
git submodule update --init --recursive
```

Running the test cycle

```
./testall_xamg_functest_competing.sh "URL" "branch" "conf"
```

The arguments meaning is:

1. `"URL"` is a git repository URL for config directory (see the config structure explanation below).
2. `"branch"` is a XAMG repository branch to test
3. `"conf"` is a XAMG build config to employ

The default values are currenly:

```
./testall_xamg_functest_competing.sh "https://github.com/a-v-medvedev/testsuite_confs.git"
   "master" "generic"
```

The result of running the script suite is a progress report which is written to stdout. The first part is stdout from the `./dnb.sh` script (`./dnb.sh` is an automated download and build script, located in `thirdparty` sub-directory). It shows the download and build progress for all pre-requisites and the XAMG library itself.

The second part is an output from `massivetests` application, which simply shows how the test tasks are submitted to the queue and the progress of their running.

The third part is a summary of all tests in a simple table form. For each test mode (that means: for each number of right-hand side vectors) and each test suite (currently we have three test suites: `blas_small`, `spmv_small`, `solve_basic_small`) we show the table of reslting states. The states are encoded as: `P` for `PASSED`, `F` for `FAILED`, `C` for `CRASH`, `A` for `ASSERT`, `T` for `TIMEOUT`, `E` for `EXCEPTION`, `N` for `NO-RESULTS`, `S` for `SKIPPED`. Each failure state is accompanied with the reference number. The separate summary file `references.txt` contain results directory name associated with the table entry in question.

The table is just an overview, the full test logs are placed in subdirectories: `sandbox_SUITE_NAME`. The `SUITE_NAME` is a name of suite (as is enumerated above: `blas_small`, `spmv_small`, or `solve_basic_small`).

Suite results subdirectory structure

Each `sandbox_SUITE_NAME` directory contains the sub-directories for each testing configuration. They are named like: `conf.CONF_NAME`. The configurations represent the different `numa_conf` codes for XAMG. The `conf.CONF_NAME` sub-directories contain `output.yaml` summary files and `result.XXX` directories for each test run. One can find the details on failed test cases in corresponding `output.yaml` file and check the `result.XXX` dir to find all the logs, stack traces and other test case output results. The `results.XXX` directories for passed tests are also saved there, so `conf.CONF_NAME` directories actually contain the full running information for the running session.

A more handy collection of the results can be found in `summary/` subdirectory. In the subdirectory one can find the copy of all summary tables, the copy of each `results.XXX` directory with failure states only, the copy of `references.txt` file which associates the `results.XXX` directories with summary table entries different from `PASSED` or `SKIPPED`.

The `summary/` subdirectories constents is also collected in `summary_XXX.tar.gz` files on the final stage.

Configuration repository structure

The configuration repository must contain three-level directory hierarchy:

```
xamg/functest/USERID_HOST
```

where `USERID_HOST` is a code for target machine and an account on it (like: `alexey_aero2`).

Inside this directory stack one must have:

```
bash# ls -1dF *
blas_small/
env.sh*
solve_basic_small/
spmv_small/
bash#
```

The `env.sh` is a machine dependent environment tuning file, the directories contain suite-dependant config files for each test suite.

The directory contents is like:

```
bash# ls -1dF *
input_axpbypcz.yaml
input_axpby.yaml
modeset.inc
params.inc
psubmit_NV1.opt.TEMPLATE
psubmit_NV2.opt.TEMPLATE
psubmit_NV4.opt.TEMPLATE
psubmit_NV8.opt.TEMPLATE
```

```
test_items.yaml
bash#
```

where `input_XXX.yaml` files represent YAML-files for each testcase, `modeset.inc` and `param.inc` tune the `massivetests` application for this test suite, `psubmit_XXX.opt.TEMPLATE` files are simple runners for `xamg_test` and are generic and typically are not changed, the `test_items.yaml` contains all output values to check for the test cases for this suite.

Adding custom testsuites to the configuration repository

One can introduce new testsuite by copying and modification of the existing test configurations. For ane new suite one should do the three basec steps: 1. Create and add the `input_XXX.yaml` files for testcases to run. 1. Modify `params.inc` to include these testcases. Change the testscope to some reasonable values (number of nodes to run tests on, number of right-hand side vectors to test, input matrices to use, etc). 1. Create a `test_items.yaml` with enties for each output value you need to check in each test case.

The `modeset.inc` and `psubmit_XXX.opt.TEMPLATE` can be typically taken without changes.