ATDM Trilinos Testing and Integration

Trilinos Developers Meeting

October 25, 2018

Roscoe A. Bartlett, Joseph R. Frye

Sandia National Laboratories

SAND2018-12416 PE
Overview of ATDM Trilinos Testing and APP Integration
Trilinos Development and APP-Integration Parts

• **Trilinos / APP Git Workflows:**
  • How git repositories and branches are set up, how merges occur, what git commands are run, etc.
  • Different git workflows are used for Trilinos developers, APP developers, and APP/Trilinos co-developers
  • Gating test suites can/should be run before each “merge” in the workflow

• **Testing gates for workflows:**
  • Gating tests can be run manually or automated, daily or “every-so-often”, etc.
  • Important test suites:
    • 1) **Auto PR Trilinos builds and tests**: Owned by the Trilinos Framework team
    • 2) **ATDM Trilinos nightly builds and tests**: Jointly owned by ATDM DevOps and APP teams
    • 3) **APP nightly builds and tests**: Owned by APP teams

• **Triaging and fixing failed builds and tests:**
  • Notification of new failures
  • Triage failures
  • Address failures
  • Manage & Follow-up
Initial creation of APP fork of Trilinos

Must pass gating:
1) Auto PR Trilinos builds and tests
2) ATDM Trilinos nightly builds and tests
3) APP nightly builds and tests

This is the goal, but the APPs are updating without checking that the native Trilinos test suite actually passes on the ATDM platforms!
**Trilinos Development and APP-Integration Parts**

- **Trilinos / APP Git Workflows:**
  - Trilinos Pull Request (PR) testing & merging to ‘develop’ [Done]
  - SPARC / Trilinos subteam workflow (manual testing by Micah H.) [Done]
  - EMPIRE / Trilinos git.git workflow (EMPIRE-owned Jenkins pipeline for testing) [Done]

- **Testing gates for workflows:**
  - 1) Auto PR Trilinos builds and tests:
    - Current: MPI GCC 4.8.4, GCC 4.9.3, Intel 17
    - Future: CUDA (see Trilinos GitHub #2646)
  - 2) ATDM Trilinos nightly builds and tests:
    - ATDM Trilinos builds for EMPIRE … EMPIRE switchover in-progress/complete?
    - ATDM Trilinos builds for SPARC … ‘cee-rhel6’ gnu, intel, and clang complete
  - 3) APP nightly builds and tests
    - EMPIRE-PIC and EMPIRE-Fluid build and test suite: A few builds posting to Jenkins only
    - SPARC build and test suite: Submits to Sierra CDash site

- **Triaging and fixing failed builds and tests:**
  - Notification of new failures: Python email tool pulling data for ATDM Trilinos builds off CDash site.
  - Triage failures: Filter out non-code failures then create Trilinos GitHub Issues
    - Joe Frye creates initial GitHub Issues, Product Areas Leads follow up from there
  - Address failures:
    - New builds: a) fix, b) allow to fail, c) temporally disabling non-critical tests
    - Existing builds: a) fix, b) allow to fail, c) temporally disable, or d) reverting PR from ‘develop’
  - Manage & Follow-up: Someone must observe and ensure failures are addressed (???Who???)
2) ATDM Trilinos Nightly Builds and Tests (CDash)

- All cleaned-up builds promoted to “ATDM” CDash group and maintained (30 as of 10/22/2018).
- Every Trilinos developer can see details on build and test failures.
- Easy to query about behavior of tests over multiple days, multiple builds, etc.
- Easy for Trilinos developers to reproduce failing builds and tests on any ATDM platform.
- Pull down results using CDash API for automated workflows.
- Python tool pulling data off CDash and daily summary email.
Where to Catch Trilinos Defects on ATDM Systems?

- **Trilinos package native test suite running in ATDM platform**
  - **Best place to catch a Trilinos defect!**
  - Trilinos developers can triage and fix a defect before APP/Trilinos Integrators need to dig in to triage APP failures caused by these defects.

- **APP (EMPIRE, SPARC) native test site running on ATDM platform**
  - Less than best place to catch a Trilinos defect.
  - Requires APP/Trilinos Integrator and APP Developers to triage problems and communicate back to Trilinos developers.

- **APP developer or user when running APP code**
  - **The worst place to catch a Trilinos defect!**
  - APP Customer has to report problems back to APP Developers who have to triage the failure and then report back to Trilinos developers.

Example:

- SEACAS update [https://github.com/trilinos/Trilinos/issues/2650](https://github.com/trilinos/Trilinos/issues/2650)
  - Broke Trilinos/SEACAS CUDA test suite.
  - Did **NOT** break the EMPIRE test suite.
  - Broke usage of EMPIRE!

If update of Trilinos was gated by 100% passing SEACAS tests, then EMPIRE developers may have never seen these defects!
Injecting Failures vs. Fixing Failures
Injecting New Failures and Fixing Failures: A Race!

- **Mean-time to fail**: Average time (in days) for when a new failure shows up in ‘develop’ branch in one or more promoted ATDM Trilinos builds.
- **Mean-time to fix**: Average time (in days) to discover, triage and fix a failure on the Trilinos ‘develop’ branch in the promoted ATDM Trilinos builds.
- **The core problem**: If “mean-time to fail” is less than “mean-time to fix”, then the ATDM Trilinos builds on ‘develop’ on average will **ALWAYS be broken** (and therefore block updates of Trilinos to the APP customers)!

\[
\text{Mean-time to fix} < \text{Mean-time to fail}
\]

Promoted “ATDM” Trilinos builds have been continuously broken for 3+ months since 7/15/2018!

Freezing the ‘develop’ branch to fix failures is never going to happen!
Options for ATDM Trilinos APP Updates

- Option-1: Make ATDM Trilinos builds clean on ‘develop’ periodically

- Option-2: Create ‘atdm-release’ branches and clean up there
Option-1: Get clean ATDM Trilinos Builds on ‘develop’

- Passes gating:
  1. ATDM Trilinos nightly builds and tests
  2. APP nightly builds and tests

- Badly Breaks:
  1. ATDM Trilinos nightly builds and tests

- Passes gating:
  1. ATDM Trilinos nightly builds and tests
  2. APP nightly builds and tests

# Failures vs. Time (days)

- app-trilinos-repo/master
- trilinos-github/develop
NOTE: Note this is really just an adaptation of the [gitworkflows(7)](https://git-scm.com/docs/gitworkflows) release ‘maint’ branch!
Option-2: Trilinos ‘atdm-release’ branches: Failures
Options for ATDM Trilinos APP Updates: Summary

• **Option-1: Make ATDM Trilinos builds clean on ‘develop’ periodically**
  • **Assumes:** “Mean-time to fix” is less than the “mean-time to fail” on ‘develop’ branch.
  • **Pro:** Requires just one set of builds on the ATDM platforms.
  • **Pro:** Simpler workflow for Trilinos developers merge bug fixes to ‘develop’ branch.
  • **Pro:** Provides quicker APP updates of Trilinos.
  • **Pro:** Allows APPs like EMPIRE to co-develop Trilinos and update Trilinos ‘develop’ and get updates to the APP fairly regularly.
  • **Con:** Requires fast reaction time to detect and triage new failures and then either a) fix, b) disable, or c) revert breaking PRs so that the “mean-time to fix” is less than “mean-time to fail”.

• **Option-2: Create ‘atdm-release’ branches and clean up there**
  • **Assumes:** “Mean-time to fix” is less than the “mean-time to failure” (not true right)
  • **Pro:** More leisurely reaction time to fix defects since no race with “mean-time to fail”.
  • **Pro:** Guaranteed periodic Trilinos updates with 100% clean ATDM Trilinos builds.
  • **Con:** Requires double the number of builds; one on ‘develop’, one on ‘atdm-release’
  • **Con:** More complex workflow for Trilinos developers to commit fixes to ‘atdm-release’ and then merge back to “develop” in two Trilinos PRs per bug-fix branch!
  • **Con:** More complex workflow for APP Trilinos co-developers involving branches, cherry-picks (e.g. EMPIRE git-git-like workflow and SPARC cherry-picking workflow).
General Software Engineering Principles for Defects
General SE Principles for Defects

- **Cost of a defect goes up (significantly) the longer it takes to detect and correct a defect.**

- **Lean/Agile SE Practices for dealing with defects:**
  - Strong automated testing (have tests help new detect defects)
  - Continuous testing (reduce the time to detect new defects caught by tests)
  - Continuous integration (reduce time to detect conflict defects)
  - **STOP THE LINE** when a new defect gets into the main development branch
    - Fixing defects in previously working software is higher priority than developing new features!
Reducing Time to Detect, Triage, and Address Trilinos Failures
Reduce Time to Detect, Triage, and Fix Defects

- **Reduce time to detect and triage new Trilinos defects**
  - Run nightly ATDM Trilinos builds against ‘develop’ and run APP native tests against Trilinos ‘develop’ [*Much progress but more to do*]
  - Filter out non-code failures and create new ATDM Trilinos GitHub Issues with dedicated person(s) to do top-level triage. [*In Progress, Joe Frye*]
  - Python tool to keep track of new failures not already covered by Trilinos GitHub issues [*In Progress*]
  - Add categories according to severity (e.g. “critical”, “blocker”, “nonblocker”) [*ToDo*]

- **Reduce time to fix defects**
  - Make it easy for Trilinos developers to reproduce failures for ATDM Trilinos builds [*Done*]
  - Send regular reminders to Trilinos Product Area Leads and assigned Trilinos developers about un-resolved ATDM Trilinos GitHub issues. [*ToDo*]
Detecting New Failures/Missing Results: CDash Email

Build and Test results for Promoted ATDM Trilinos Builds on 2018-10-24

- Missing expected builds: bme=1
- Failing tests without issue tracker: twoi=1
- Failing tests with issue tracker: twi=9

Failures in **red** require action!

- Missing test results!
- Failing test without issue tracker!
Reproducing ATDM Trilinos Builds: Trilinos Wiki

ATDM Trilinos Builds
Roscoe A. Bartlett edited this page on Jul 23 · 1 revision

The ATDM Project maintains a number of builds of the Trilinos `develop` branch on a number of different platforms used by the ATDM project. Information about these builds can be found at the internal SNL Confluence page:

- ATDM Builds of Trilinos

The ATDM builds of Trilinos are shown on the open CDash site pages:

- ATDM Trilinos builds summary on CDash
- ATDM Trilinos builds failing tests on CDash

Information on reproducing ATDM Trilinos builds shown on CDash can be found at in the open page:

- Local ATDM builds of Trilinos
Quick-start

After cloning the Trilinos git repo on one of the supported ATDM machines, a local configure of Trilinos enabling a few packages is performed as:

```
$ cd <some_build_dir> /
$ source $TRILINOS_DIR/cmake/std/atdm/load-env.sh <job-name>

$ cmake \
    -GNinja \
    -DTrilinos_CONFIGURE_OPTIONS_FILE:STRING=cmake/std/atdm/ATDMDevEnv.cmake \
    -DTrilinos_ENABLE_TESTS=ON -DTrilinos_ENABLE_<Package1>=ON \ 
    $TRILINOS_DIR
$ make NP=16 # Uses ninja -j16
$ ctest -j16 # Might need to be run with srun or some other command, see below
```

The command:

```
$ source $TRILINOS_DIR/cmake/std/atdm/load-env.sh <job-name>
```

determines what machine you are on (using `hostname`) and then loads the correct environment automatically for that machine and for the build options passed through in `<job-name>` (or errors out if the current machine is not one of the supported machines).

The `<job-name>` argument is a single string of the form `XXX<-keyword0>-<keyword1>-...`. The standard order and format of this string is:
Reproducing ATDM Trilinos Builds: Systems Info

Specific instructions for each system

- ride/white
- shiller/hansen
- chama/serrano
- mutrino
- SEMS rhel6 environment
- CEE rhel6 environment
- waterman

ride/white

Once logged on to white (on the SON) or ride (on the SRN), one can directly configure and build on the login node (being careful not to overload the node). But to run the tests, one must run on the compute nodes using the `bsub` command to run them using a CUDA build. For example, to configure, build and run the tests for the `cuda-debug` build for say Muelu on white, (after cloning Trilinos on the `develop` branch) one would do:

```
$ cd <some_build_dir>/

$ source $TRILINOS_DIR/cmake/std/atdm/load-env.sh cuda-debug

$ cmake \
    -G Ninja \
    -DTrilinos_CONFIGURE_OPTIONS_FILE:STRING=cmake/std/atdm/ATDMDevEnv.cmake \ 
    -DTrilinos_ENABLE_TESTS=ON -DTrilinos_ENABLE_Muelu=ON \ 
    $TRILINOS_DIR

$ make NP=16

$ bsub -x -Is -q rhel7F -n 16 ctest -j16
```
Reproducing ATDM Trilinos Builds: GitHub Issue

Steps to Reproduce

One should be able to reproduce this failure on waterman as described in:

  More specifically, the commands given for waterman are provided at:
  The exact commands to reproduce this issue should be:

```
$ cd <some_build_dir>/

$ source $TRILINOS_DIR/cmake/std/atdm/load-env.sh cuda-9.2-release-debug

$ cmake \
  -GNinja \
  -DTrilinos_CONFIGURE_OPTIONS_FILE:STRING=cmake/std/atdm/ATDMDevEnv.cmake \
  -DTrilinos_ENABLE_TESTS=ON -DTrilinos_ENABLE_Intrepid2=ON \ 
  $TRILINOS_DIR

$ make NP=20

$ bsub -x -Is -n 20 ctest -j20
```
Addressing Trilinos Failures
How to Address Trilinos Failures?

• **Keeping already cleaned-up promoted builds clean**
  a) Fix the failures => **Best option!**
  b) Mark failing tests as “expected may fail” and not trigger global failure in Python tool:
     • Only for non-blocking issues
     • Allows us to watch test run but not block updates of Trilinos to APPs
     • Best for cases where someone is working (or soon is going to work) to fix non-blocking failures.
  c) (Temporarily) disable failing tests:
     • Only for non-blocking issues
     • Best for cases where no-one is going to work on fixing the failures anytime soon.
  d) Revert the commit(s) (or PR merge) causing the failure:
     • => Perhaps best option for critical or blocking failures that can’t be fixed soon!

• **Initial failures from setting up new platforms**
  a) Fix the failures
  b) (Temporarily) disable failing tests (non-blocking issues only)
  c) Mark failing tests as “expected may fail” and not trigger global failure in Python tool (non-blocking issues only)
     • **NOTE:** Reverting commits is NOT an option for cleaning up failures that occur when setting up new builds on new platforms or envs on existing platforms.
Following up on ATDM Trilinos GitHub Issues
Problems with Trilinos GitHub Issue Management

- **Problems not working issues:**
  - Ignoring new ATDM Trilinos GitHub Issues (e.g. no replies for weeks)
  - Not following up on Issues after some initial investigation

- **Problems closing issues:**
  - Closing Issues before getting confirmation on CDash (sometimes issue is *not* addressed)
  - Not closing issues after issue has been addressed (clutter up list of active issues)

**Proposed Solution => Send out weekly email listing**
- Send out emails to Trilinos Product Area Leads for open ATDM Issues in their area
- Send out emails to assignees of open ATDM Trilinos issues?
### ATDM Trilinos Linear Solvers Product Area GitHub Issues on 2018-10-24

<table>
<thead>
<tr>
<th>#Issue</th>
<th>ATDM Priority</th>
<th>Created</th>
<th>Last Updated</th>
<th># Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>#3499</td>
<td>Blocker</td>
<td>2018-09-25</td>
<td>2018-10-03</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3686</td>
<td>Blocker</td>
<td>2018-10-16</td>
<td>2018-10-16</td>
<td>0</td>
</tr>
</tbody>
</table>

- Anasazi tests failing in ATDM build on mutrino
- Likely broken by PR #3481 merged to ‘develop’ on 9/21/2018 …
- Teko_ModALPreconditioner_MPI_1 Failing in ATDM cee-rhel6-clang-opt-serial build
- ???

- Python tool that pulls data off GitHub (query labels ‘ATDM’, ‘TPA: Linear Solvers’, …)
- **To Who?** Send only to Trilinos Product Area Leads? Send email also to Issue assignees?
- **Frequency?** Once a week? Twice a week?
Wrapping Up
Observations and Open Questions

• SNL ATDM is test-bed for components approach for exascale HPC software!
• Two approaches being compared:
  • Full buy-in to using and co-developing components: EMPIRE
  • Avoid deep dependencies or co-development of components: SPARC
• The role of Trilinos in ATDM will either be viewed as:
  • A) Successful:
    • => Encouraging the usage of components approaches
    • => Leading to more future funding for Trilinos?
  • B) Unsuccessful:
    • => Discouraging the usage of components approaches
    • => Leading to less future funding for Trilinos?
• Questions:
  • At the end of ATDM, which approach will be viewed more successful?
  • What will that say about Trilinos?
  • What will this say about the future of components in CSE/HPC exascale?
  • If we can’t succeed with components with Trilinos in ATDM at SNL, how we expect this to work in the larger ECP project across labs, universities, etc.?

Let’s stabilize Trilinos for ATDM APPs and remove that as an excuse!
Trilinos Development and APP-Integration Parts

• **Trilinos / APP Git Workflows:**
  - Trilinos Pull Request (PR) testing & merging to ‘develop’ [Done]
  - SPARC / Trilinos subteam workflow (manual testing by Micah H.) [Done]
  - EMPIRE / Trilinos git.git workflow (EMPIRE-owned Jenkins pipeline for testing) [Done]

• **Testing gates for workflows:**
  - 1) Auto PR Trilinos builds and tests:
    - Current: MPI GCC 4.8.4, GCC 4.9.3, Intel 17
    - Future: CUDA (see Trilinos GitHub #2646)
  - 2) ATDM Trilinos nightly builds and tests:
    - ATDM Trilinos builds for EMPIRE … EMPIRE switchover in-progress/complete?
    - ATDM Trilinos builds for SPARC … ‘cee-rhel6’ gnu, intel, and clang complete
  - 3) APP nightly builds and tests
    - EMPIRE-PIC and EMPIRE-Fluid build and test suite: A few builds posting to Jenkins only
    - SPARC build and test suite: Submits to Sierra CDash site

• **Triaging and fixing failed builds and tests:**
  - Notification of new failures: Python email tool pulling data for ATDM Trilinos builds off CDash site.
  - Triage failures: Filter out non-code failures then create Trilinos GitHub Issues
    - Joe Frye creates initial GitHub Issues, Product Areas Leads follow up from there
  - Address failures:
    - New builds: a) fix, b) allow to fail, c) temporally disabling non-critical tests
    - Existing builds: a) fix, b) allow to fail, c) temporally disable, or d) reverting PR from ‘develop’
  - Manage & Follow-up: Someone must observe and ensure failures are addressed (???Who???)
THE END