2019-10-10 Software Testing and Verification

Yannick opened the meeting by announcing that several PRs were merged in earlier this week that touched a lot of areas of the code base, and asked if anyone has had any issues. No one responded so it appears that the code base is in good shape.

Yannick gave the agenda which contained two main topics: Automated testing and Floating point exception trapping. We started with automated testing

Automated Testing

Where do we store/manage our test data?

We are hitting the bandwidth limits of git-lfs a lot now that we are automatically testing many of the JCSDA repositories. It was suggested that we might do better storing the large files (ie, files checked into git-lfs) on Amazon S3. We could add a test at the beginning that checks that the proper files are downloaded, and if not updates accordingly to minimize the download traffic.

Maryam has started looking into using S3 for test files, and has so far noted (compared to git-lfs):

- The tests run much faster with S3
- S3 is a more sustainable solution
- CodeBuild (Amazon auto-test service) is better connected to S3

A discussion ensued with these highlights:

Advantages of using S3 for our large file storage

- git-lfs under the hood also uses S3, but the primary difference is that using S3 directly provides us with a significant money savings
    - S3 still charges for storage and bandwidth, but there are no egress charges (for moving data around) for AWS compute instances in the same AWS region. So, we can arrange it so there are no egress charges for CI testing in AWS CodeBuild. There would be egress charges for downloading data to locations other than AWS.
    - Estimates of savings is about 4x compared to git-lfs for storage itself (we still need to estimate the difference for egress based on usage patterns)
- Advantages noted by Maryam above

Disadvantages of using S3

- We will need to open up access to more ports on NOAA HPC systems (Hera, WCOS), which will be difficult to sell to the IT/Security folks
    - One of the selling points of git-lfs was that it required open access on a much smaller set of ports

Notes

- git-lfs can point to different repos such as our JCSDA S3 buckets, but we will still get charged for downloads
- Can we simply store a copy of the test data on Hera/WCOS and avoid data handling charges?
- Can we host data on a UCAR site, perhaps on a Google Drive?
    - Since UCAR is an educational organization, we get "unlimited" storage on Google Drive
        - We know that a few TeraBytes of storage is, so far, acceptable
    - Some partners may not have access to our UCAR/JCSDA Google Drive

If you have any further thoughts on this, contact Maryam or Mark. If needed, we can set up a discussion on the GitHub JEDI Team.

Tiered Testing Approach

We have a large set of unit tests now (small, fast running, targeted to specific modules) along with larger more expensive tests (targeted to flows such as 3DVar, 4DVar, etc.). With the automated testing on every PR and commit to PRs, it is getting too expensive to run all tests in this mode. We are proposing to split the testing, initially, into three tiers so that the more expensive tests can be run only on a nightly or weekly basis. We would add environment variable controls to each repo (SABER is an example) that the automated test flow can inspect to know which tiers to run. Here is the proposed tier numbering:

- Tier 1: run only unit tests, fast running, for the PR and commit testing
- Tier 2: run unit tests, plus medium expensive testing
- Tier 3: run all tests

Tier 2, 3 allow for splitting the expensive tests into medium and high categories. The proposed test that checks and downloads data can be utilized for managing the tiered test data.

Here are highlights from the discussion:

Notes

- Should we store data for different tiers in different directories to facilitate downloading? Only data needed for the desired test tier should be downloaded, to reduce time and egress charges from LFS or S3.
- How do we make sure everyone follows the tier scheme?
- There is a potential for test escapes (undetected defects) in that deferring an expensive test to the weekend may allow a defect (that the expensive test catches) to be merged into develop before the weekend since the Tier 1 tests all passed
- For Tier 1, we could build in release mode (compile optimization enabled) in order to get this testing to run as fast as possible
    - Debug build slows the tests way down

Valgrind

Valgrind is a tool to check for invalid memory usage such as a memory leak. It potentially can be installed in the automated testing by using a script to check the output files of Valgrind and issue a pass/fail return code. Valgrind is slow however (Benjamin quoted that it slowed tests running in 1-2 minutes down to several hours), so it would need to be a Tier 3 test. Valgrind tests could be implemented by means of a shell script enabled by an environment variable. For an example, see saber PR #33 which is now under review. Alternatively, or in addition, they could be implemented through the CDash web application.

CDash

CDash is a GUI viewer that displays the results of build and test actions. It can post the results of building with different platforms/compilers along with the results of testing. You can see compiler warnings/errors and test pass/fail status all in a neat display. Ryan gave a short demo of CDash to the group. CDash requires a background server running PHP. It appears possible to add in more than build/test status in that it should be possible to add Valgrind to the dislpay using customization features of CDash.

Floating Point Exception (FPE) Trapping

Jim R has a PR under review that adds the ability to trap FPE's. It is controlled by an environment variable that toggles (enables/disables) the trapping. "Trapping" means that when an FPE is detected, the code writes out the type of exception (divide by zero, overflow, etc.), writes out a backtrace and then aborts. By default the trapping is disabled, so that the behavior matches that of the current system.

For Jim's PR, when FPE trapping is enabled, it is enabled globally in every component of the system. This is not desirable since there are code modules out of our control (models, ropp, etc) which could experience FPEs and we would be stuck with the aborting action. What we would like to have is the ability to enable/disable FPE trapping on a component basis so that for core modules such as OOPS, UFO, IODA we could enable trapping (and fix the FPEs) while disabling the trapping for other components.

It was decided to go ahead and merge Jim's PR since we can still use this PR as is to enable trapping on tests that don't use the external components (UFO tests for example) and get started repairing FPE defects. We can add in the component-level trapping in a subsequent PR.

Other topics

Yannick asked if there were any other topics.

Steve H asked about introducing/automating performance tools (assess execution time and memory usage). There was general interest in this topic and here are highlights from the discussion:

- Mark O mentioned that a python performance tool (psutil?) is available in the workflow
    - Easy to use
    - Lightweight, but good for collecting long term statistics
- Might make sense to do performance testing only on the larger runs (higher Tier level)
- Tau (profiler) is available on Amazon
    - Works with MPI
    - Nice interface (graph with routines and process element are the axes with timing plotted as a surface)
    - Dumps results into a file which, among other things, allows us to compare results between different runs
- DDT profiler (MAP?) is available on Cheyenne
- Intel has a profiler too
- Jim R has a profiling scheme
- We could add a test to IODA that reads in a large radiance file and analyze the run with a profiler
- From a practical sense, we cannot put multiple profilers in a container because this will excessively inflate the container size

Action Items

- Investigate the access restrictions on Hera/WCOS, and the possibility of storing test data on Hera/WCOS
- For those that want to continue the discussion about test data storage/management, please contact Mark M or Maryam
- Investigate and provide demos of performance assessment tools: Tau, Intel, Jim's profiler, etc.
    - Need summaries of each profiler so that we can make a decision about what to provide in the containers
    - Mark M will open a ZenHub issue on the special topics board for discussing the results of these demos

Tom added in closing that he is very happy with the high level of collaboration in the group. This is a critical behavior leading to our success. Tom added that it's great to be adding in the extensive test/verification capabilities to the system, which will lead to operational grade software.