Day 2 Discussion: Collegeville 2021
Breakout Room 1
Technical Approaches to Improved Software Teams

Overview: The content your discussion group creates in this document will be synthesized in a blog posting for https://bssw.io

Instructions:

1. Pick one person in your discussion group to create a new copy of this Google Doc
2. Make a copy of this template in a new Google Doc (the person from step 1)
3. Share the edit link to the document in step 2 with others (copy and paste into Zoom chat)
4. Co-edit the document: Can have one lead writer with others modifying, or another approach
5. Send the document to Mike Heroux at the end of the session by email (mheroux@csbsju.edu)

Add group member names for anyone who wants attribution in the blog post:

1. Barry Smith, Argonne National Laboratory. Developer of open source PETSc package.
2. Charles Ferenbaugh, Los Alamos National Laboratory
3. Sarah Osborn (osborn9), Lawrence Livermore National Laboratory
4. Ulrike Meier Yang, Lawrence Livermore National Laboratory, ulrikeyang
5. Sameer Shende, University of Oregon, developer of TAU and E4S.

Discuss as a group the most promising technical approaches you see as opportunities for scientific software teams. Summarize discussion in outline form.

- Portable Performance tools
    - Having tools to analyze the performance of the software will greatly help, especially with GPU architectures. Tools such as TAU [http://tau.uoregon.edu] that operate consistently on un-modified binaries across multiple GPU architectures can help assess the performance of numerical libraries, tools, and applications.
    - Desired features
        - Map from low level profile information to the abstractions and high level code of each package. Requires an API that is used by each “library/framework” to “mark” the code.
        - Visualization of performance data

- Container technology
    - Container technology holds immense promise for reproducibility of scientific results. The ability to package software dependency in the form of custom containers, building from base containers can help improve programmer productivity. The E4S project [https://e4s.io] provides containers that operate on multiple architectures and support GPUs.

        - Potential issues: What if users want to use a particular version? Want the ability to control specific features? You can make custom containers with Spack with the ability to specify specific versions of packages (e.g., trilinos@13.0.1) and create a container image. Examples of this are described in the workflow with the Nalu-wind ECP application and a container built with it as demonstrated in the videos at the E4S Industry Council workshop: https://www.exascaleproject.org/event/e4sforindustry/.
        -
- Distributed VCS/pull request/ regression testing workflow
    - Imposes discipline on source code changes and testing of changes, ability to add comments and have a discussion is very helpful.
    - Best if integrated with testing of changes - see next item
    - Increases # of people who know a part of the code, reduces “bus factor”
    - Additional features desired.
        - GitLab does not handle reviewers properly automatically for example. Someone must add reviewers manually and must know the appropriate reviewers to add.
        - Help with the iterative process of MR. Make it easier to revisit a MR after changes. With bitbucket you can add tasks for the submitter and get a message when all tasks are completed instead of getting a message for each individual update
            - An intermediate granularity for getting notified of every comment / change to when it's time for a reviewer to come back and review a MR.
            - Would also like to avoid having to reread old/stale comments.
- Regression testing/ CI
    - Old way: batch testing once/night, if something failed had to weed out which commit caused it to fail
    - Now: CI runs tests more regularly, easier to narrow down cause
    - Issue: if a test fails on a small, obscure machine, the person who needs to debug might not have access to that machine. CI should provide “debug access” to all test systems”.
    - Testing regularly makes releases simpler
    - Need to make sure people test on GPU machines - easy to break
    - Test coverage - enforcing for new code - but hard to automate because of corner cases
    - Testing for coding standards before MR
    - Testing against third-party libraries that are also changing
- Mechanisms for adopting new technologies, approaches, paradigms
    - example: GPUs - need to do major restructure of code - Kokkos, OpenMP offload, OpenACC - require a lot of discussion, decision
    - discuss using email, slack, issues, random comments in MRs, ...
    - another example: mixed precision

        - Can’t just have a working group that is solely focused on this, because they need to be aware of other changes to code, etc.
        - Need everyone to be involved with everything -- Not feasible!
        - Way it often goes: put one smart person on it, *hope* they make good decisions
    - Need a paradigm shift to avoid having to write the volumes of code that are necessary to support evolving architectures (GPUs – Nvidia, AMD, …..).
    - Don’t have funding to put a full time person on a new (large) effort; Also, others are busy with other projects, with code maintenance, etc.

About 20 prior to the end of the session, around 1:40 pm CDT, try to reach consensus on 3 - 5 high-level technical approaches your team identified

- We listed 5 approaches and agree that all of them are important - didn’t have time to drill down further