2019-11-22: Profiling Part 2

Steve H opened the meeting by announcing that the focused topic for today, as described in the meeting announcement:

On Oct 24 we discussed the idea of integrating profiling into JEDI unit testing for the purpose of identifying problems such as memory leaks, load imbalances, and other potential signs of bugs that may affect performance. Then we discussed several options for profiling tools, including the GPTL library, the psutil python-based tools now being used for JEDI-Rapids, and the TAU profiler. We also briefly discussed the MAP profiler that is included with the DDT debugger in the Arm Forge package (it was agreed that this can be an extremely useful tool for profiling but since it is proprietary and largely graphical, it's probably not a good fit for the automated CI testing).

This week we will discuss another option, namely the intel vtune/inspector/advisor tools. After reviewing our options, the discussion with then turn to how we might actually implement them in the CI testing.

Then the meeting proceeded with Ryan, who presented the following slides:

2019-11-20 Intel ...ofiling Tools.pdf

Please consult the slides for details - what follows is only a brief summary. Ryan began by describing three distinct tools that intel provides for profiling and optimization, namely **Advisor** (focusing on vectorization and shared-memory threading), **Inspector** (focusing on memory and thread checking) and **Vtune** Amplifier (general profiling). The remainder of the presentation focused on Vtune Amplifier which intel intends to rename as Vtune Profiler with the next release of Parallel Studio.

Though these three tools are shipped with Intel's Parallel Studio software package (which also includes the compilers and intel mpi), they can also be downloaded from intel's web site free of charge (though we need to verify that they remain free of charge to use). Furthermore, they can be used to profile code that is compiled with gnu and clang compilers as well as intel compilers.

No special compiler flags or code instrumentation is necessary to use these tools. Intel only recommends that you compile with debug information (-g), which is already done in jedi Debug and default (debug with release info) builds.

Ryan proceeded by showing example diagnostics as viewed from the GUI, which is user-friendly. And, he showed an example pull request in oops where insight he obtained from vtune was used to decrease the run time of a particular test by 45%.

Then he demonstrated the command-line usage, which is particularly relevant to today's topic of automated testing. He compared it to valgrind in the sense that the user/developer only needs to prepend a command (plus options) before the call to the application in order to invoke the profiler. Profiling results can also be saved and reloaded to monitor changes in performance.

After Ryan concluded his presentation, open discussion followed. Steve H asked how we would implement this for CI - would we write a test that runs the profiler on other tests? Ryan responded with another possible option of just optionally prepending the profile call to the tests as is now done with valgrind in saber.

Yannick and Mark M then described two use cases: 1) for optimization of code, and 2) to detect changes in performance that might indicate code bugs. The CI testing would focus more on the latter. Mark M mentioned that there might be some advantage to using the same tool (e.g. vtune) for both use cases - for example, if CI identifies a bottleneck, it might be beneficial to explore it more thoroughly via the GUI. Mark O suggested we could look for both sudden performance changes and trends over time. Ryan added that vtune has the ability to show differences from the last time it ran (similar to reporting by CodeCov).

Steve H asked if both the GUI and command-line tools were free. Ryan confirmed that they were both free to download. *Note - we have conflicting information from an intel representative that Vtune is not free indefinitely but Ryan has been using it for some time without any problem. So, we just need to give it a try and see.* Mark M asked if Vtune can be installed via a package manager like homebrew or apt. It appears the answer may be no but they can be downloaded from the web site. Steve asked if it can run on a laptop. Ryan said yes, but Mac can currently only run the GUI due to security issues with command-line executables.

Ryan also added that the command-line output can be represented as json or xml and benchmarks can be saved for monitoring performance changes.

We then had a brief summary of the other tools we discussed at our Oct 24 meeting (see those notes for details), including the python psutils package and the GPTL library.

There was then some discussion on whether we should monitor absolute run time or relative run time when flagging performance changes. The former would be platform-dependent so it might be good for CI testing on AWS nodes but you may wish to have the ability to disable the profiling tests on different systems that may have different timings.

Yannick suggested that a simple way to start with this is to start monitoring test times (as reported by OOPS) and see how much they vary.

Steve H commented that vtune might be easier in some ways than GTPL because it does not require any new compiler options or code instrumentation. Mark O and Yannick then mentioned that they measure somewhat different things - vtune takes a sampling approach whereas GPTL logs the statistics of individual function calls. So, both may be complimentary.

Travis suggested we use the caching abilities of codebuild to track performance trends. Xin advocated for valgrind testing. Travis agreed but added that exceptions need to be set up for external codes such as MOM that may throw many errors and warnings but that we have no control over to fix.

Ryan advocated for CDash as another option to track overall test times that integrates well with ctest/cmake. It can also track changes across different machines and compilers. We would probably need to query the API for automated flagging of performance changes and reporting this back to GitHub reviewers. Ryan suggested that this could be linked to Travis CI. Maryam added that we could do it through GitHub webhooks. Yannick advocated for simplicity - one more indicator of green (pass) or red (fail).

Mark M asked how we might proceed in terms of the containers - should we install vtune or GPTL? Yannick responded that before we do that, the first step would just be to gather statistics on the test times and see how much they vary. Then we can decide from there. Ryan mentioned that adding Vtune to the containers would increase their size by 0.5-1 GB. Mark O suggested that users could install Vtune in their home directories so it would be available from the containers but not shipped with them. But, this doesn't solve the issue for CI testing. Mark O added that psutil would not add much overhead to the containers.

Yannick then suggested that the next step would be to assign someone to work on this. Jim R said he has already been working on integrating GTPL into jedi with cmake help from Mark O. He's working on a FindGTPL cmake module and is close to a draft PR. Mark M mentioned that a cmake module isn't necessary to link the library in to jedi but Mark O added that it would be beneficial to set up top-level CMake flags that could then be used to set compiler options. There was some discussion on whether GTPL should be linked in through the rpath or through LD_LIBRARY_PATH - either should work.

Yannick summarized by identifying three tasks and individuals to work on them:

- Collecting statistics on test run times in oops (Maryam)
- Implementing valgrind tests (Xin)
- Monitoring overall performance trends with CDash (Ryan, Maryam)

There was then some discussion of keeping the container size small to limit test time. It was asked if caching the container and test data can help. Maryam confirmed it is possible but doesn't help much with the timing. Storing the containers and data on EBS volumes is another option but Mark O added that these take a while to spin up as well. Yannick agreed there is no magic solution - any way you slice it, when an AWS instance is launched to do the testing, the data has to get there. The only way to avoid this would be to have a dedicated AWS node up all the time, which we can consider.

Yannick closed the meeting with an appeal for all JEDI users and developers to provide feedback on what other topics they would like to see addressed at our biweekly topic meetings. Feel free to email Mark M, Yannick, or any member of the JEDI core team with suggestions or enter your suggestions in the ZenHub board directly.