Using TAU to Identify Parallel Performance Issues

This exercise will give you experience using performance tools, in this case TAU, to identify parallel performance issues in production-level scientific applications. Although the details differ between different tools, the **process** of performance profiling is similar. The intent of this exercise is to help you become familiar with that process.

If you wish, you are encouraged to use your own scientific application of interest for this exercise. Otherwise, you will work with the UNRES MD code developed by the Scheraga group at Cornell. The UNRES MD code utilizes a carefully-derived protein force field to study and predict protein folding pathways by means of molecular dynamics simulations. Previously, UNRES used many replicas of the protein of interest, each run in serial, to study the folding pathway. This approach worked well for very small proteins, but it was unable to access the long timescales necessary for the folding of larger proteins. To access longer timescales, the Scheraga group introduced fine-grained parallelism into UNRES, allowing each protein replica to be run in parallel. After introducing this fine-grained parallelism the code was profiled and a couple of major parallel performance issues were detected and resolved, resulting in greatly improved performance. In this exercise you will profile the original version of the fine-grained UNRES code and identify the most important issues limiting parallel performance in the original code.

Preparation: The code and other materials that you need for this exercise are located at:

http://staff.psc.edu/blood/petascale_school_2010

There are two different tarballs for kraken and ranger. Login to the machine of your choice and use wget to grab the appropriate tarball. Unpack the tarball and read the brief README file in the UNRES directory. There are some scripts there to help you set up your environment, compile, and run jobs instrumented with TAU.

Steps:

1. Compile the UNRES code and do a scaling study from 1 to 64 cores to get the baseline performance for this code before instrumentation with TAU. The relevant timing in UNRES is the time for the MD loop to complete. You can find this time in seconds by running:

   ```
   grep "MD steps" t0395.MD.out_GB000
   ```

   Record your results so you can compare them with the instrumented version.

2. Recompile UNRES with TAU instrumentation. Use a Makefile that includes papi, mpi, and pdt (do not include “phase”). Look in the directory given in the example build script

for the necessary TAU Makefiles. Run the code again (only one run is necessary, but make sure it is over at least several cores) to measure the overhead of full instrumentation. Compare the timing of this run to the original. You will use this run to create a selective instrumentation file to eliminate most of the overhead.

3. Go to the directory where you ran the instrumented UNRES. You should see a number of profile.* files corresponding to the number of cores over which you ran. Make sure you have an X server running on your machine (and that you connected via ssh –X), and execute the command `paraprof.` Using paraprof, create a selective instrumentation file as demonstrated in the lecture. Use this selective instrumentation file to rebuild UNRES. Also include –optKeepFiles and –optPreProcess in your TAU_OPTIONS. Use the same TAU Makefile as before and redo the scaling study. This time, however, set the TAU_CALLPATH variable to 1 and TAU_CALLPATH_DEPTH to 50 to collect callpath information.

4. Compare the timings of your selectively instrumented runs to your original runs. The amount of overhead should now be significantly reduced. If there is still substantial difference in the timings (greater than 10%) then we will need to assess whether the timings of major functions reported by TAU seem reasonable. You may need to do several runs with and without TAU to determine whether differences are due to TAU instrumentation or random variability between runs.

5. Once you have instrumented runs with low overhead you can use paraprof to explore the profiling data that you have collected. Remember you can use "paraprof --pack" to merge all of your profile* files into a single .ppk file. You can copy this back to your workstation to use the TAU GUI analysis tools locally. Use paraprof to look at the callpath information for PE 0 and PE 1. Try to find the hotspots in the code and the path to the main loops. Since UNRES uses a master/worker approach, the callpath for PE 0 is different from the other PEs. You'll notice that runtime is dominated by a couple of routines whose time does not change with increasing processors. These routines are part of the startup time, and as mentioned before, we are only interested in the time corresponding to the MD loop.

6. To isolate the timings that you are interested in, you can use the callpath information and your knowledge of the code to define execution phases. For the purposes of this exercise, this has been done for you. Paste the contents of select_MDphase.tau (in your UNRES directory) at the end of your selective instrumentation file. Rebuild UNRES with the new selective instrumentation file, but this time choose a TAU_MAKEFILE that includes papi, mpi, pdt, and **phase**. Rerun your scaling study with the new executable. This time

do *not* set the TAU_CALLPATH variables in your batch script (phase and callpath profiling interfere with one another in TAU).

7. As demonstrated in the lecture, look at the profiles for the functions in the MD phase. If possible, you will probably want to create *.ppk files (via "paraprof --pack") and copy these back to your own machine to look at them with your local version of TAU. For each run, use the filter in the function legend to filter out all functions that are not labeled with the name of the MD phase. For each run, add the mean time to the comparison window. Look at how the functions perform as you increase the number of processors. Also, look at the performance of individual functions across all the processors at high processor counts. Determine which functions are limiting the parallel efficiency of the code and what the problems are.

8. Time permitting, create a selective instrumentation file to include only the top 15 or so major routines from the MD phase. To find these routines, look at a profile that had the TAU_CALLPATH environment set. You'll want to look at a profile that was done at a high processor count so you see which routines are the time-consuming routines at scale. Don't forget to include routines that **call** the time-consuming routines, especially those that call time-consuming MPI routines. Now rebuild and rerun UNRES at the higher processor counts with tracing enabled rather than profiling. You will probably want to edit the t0395.MD.inp file so that nsteps is equal to 10-50 steps or so (after you edit this file, make sure that the ampersand on the far right of the line you edited still lines up in the same column as the other ampersands). Use Jumpshot to look at the trace and get a feel for the computation and communication patterns in the code. You may want to check out the "cheat sheet" here:
   http://www.psc.edu/general/software/packages/tau/TAU-quickref.pdf

   You should also be able to clearly identify from the trace the scaling problems that you saw by looking at the profiles.