Hands-On Exercises for Performance Tuning Workshop

October 29, 2019

These exercises go along with the Performance Tuning workshop. The code is based on the HPCCG miniapp from Mantevo. These exercises assume the use of the Intel 18.0.3 compiler and MVAPICH2 2.3, but you are welcome to use other compilers.

1. Open a Pitzer desktop session through OnDemand (1 node for 3 hours) and open a terminal.

2. Download the source code:

```
git clone git@code.osu.edu:khuvis.1/
    performance2019_handson.git
```

3. Begin by compiling and running the code:

```
module load intel/18.0.3 mvapich2/2.3
make
mpiexec -np 2 ./test_HPCCG 150 150 150
```

4. Your run in the previous step should have exited early with a segmentation fault. So, use the ARM DDT debugger to determine where the code failed and fix the error. Make sure to set the optimization level to `-O0` and add debug symbols with `-g` by changing the value of `CPP_OPT_FLAGS` in the Makefile, then recompile.

```
make clean; make
module load arm-ddt
ddt -np 2 ./test_HPCCG 150 150 150
```

5. Now that you have a working code, test different compiler options for optimization levels to see how it affects performance.

```
time mpiexec -np 2 ./test_HPCCG 150 150 150
```

Compiler flag	Runtime (seconds)
`-O0`	
`-O1`	
`-O2`	
`-O3`	

6. Next, let's get an overview of the bottlenecks in the code with the ARM performance report:

```
module load arm-pr
perf-report -np 2 ./test_HPCCG 150 150 150
```

   Open the HTML file in your browser to view the report. What are the bottlenecks in the code?

7. Next, add the `-qopt-report=5` compiler flag and recompile to view an optimization report.

8. Use ARM MAP to see which functions/lines of the code are most expensive. Make sure that you are compiling with `-xHost` to ensure optimal vectorization.

```
module load arm-map
map -np 2 ./test_HPCCG 150 150 150
```

   Look for any opportunities to improve vectorization among the most expensive functions.

   Hints:

   - You should see a recommendation in the optimization report to add the `-qopt-zmm-usage=high` command-line option for your function. Make sure to add it to the Makefile.
   - Try replacing an assignment to an array element with a temporary variable to enable vectorization.

9. Next, Look for any opportunities to improve the memory access patterns among the most expensive function you identified with ARM MAP.

   Hint: Look for nested loops that are not ordered correctly.

10. Use ITAC to get a timeline of the run of the code.

```
module load intelmpi
mpiexec -trace -np 40 ./test_HPCCG 150 150 150
traceanalyzer <stf_file>
```

    Look at the Event Timeline (under Charts). Do you see any communication patterns that could be replaced by a single MPI command?