Profiling tools for gcc compilers

Code Coffee - 16/11/21
Vignesh Vaikundaraman

Why do we need them?

- Increases readability of code.
- Code runs faster!
- Analyze bottlenecks (especially helpful for large codes)
- Check the code block covered in our cases.

How do we do this?

- `gprof` - profiling tool that looks at time bottlenecks
- `gcov  - code coverage tool`

`gprof` - The profiling tool for GCC compilers

Unoptimized code that doesn't work
Optimized code that doesn't work

Using `gprof`

- Flags to be added to your compilation: `-pg`

  This enables profiling in compilation

- Execute the program
- Check if a file `gmon.out` has been created (has the profiling information)
- Now converting it to readable format

```
gprof <executable> gmon.out > analysis.txt
```

We have all the required data in the file `analysis.txt`!

Interpreting the results

Two parts:

- Flat profile - sorted in time showing the cumulative time spent in reach function/subroutine.
- Call graph - showing a tree and how much time each function takes in the tree.

Options: -b : removes the instructional information from the file.

-p: print only flat profile, -q: print only call graph

gcov - Code coverage tool

Using gcov

- Flags to be added during compilation: `--coverage`
- Execute the program
- Check if `.gcna, .gcdo` files have been generated.
- `Running the gcov tool :`

```
gcov *.c/f90/cpp
```

`This should generate .c.gcov files which will have the coverage data.`

`lcov` - generating html reports

- Extension of gcov to produce easily readable html reports.
- Execution:

```
lcov --coverage --directory . --output-file coverage.info

genhtml coverage.info --output-directory out
```

References

https://www.lrz.de/services/compute/linux-cluster/tuning/gprof/index.html

https://www.thegeekstuff.com/2012/08/gprof-tutorial/

https://medium.com/@naveen.maltesh/generating-code-coverage-report-using-gnu-gcov-lcov-ee54a4de3f11