HIGH PERFORMANCE COMPUTING

PROFILING

P.ADITHYA SRINIVAS CED17I007

Introduction

In computing , profiling is a form of program analysis that measures the hotspots in a code, time complexity, how a certain snippet of code,frequency duration of function call. Most commonly, profiling information serves to aid optimization of the given program.The most important “resource” in terms of high performance computing is runtime.

How profiling is done?

A common profiling strategy is to find out how much time is spent in the different functions, and lines of a code.The parts of the program that require the dominant fraction of Runtime is called a ***hotspot***.These hot spots are subsequently analyzed for possible optimization opportunities.

Two technologies are used for function and line-based profiling:

- Code instrumentation
- Sampling Instrumentation

Code instrumentation :

- It works by letting the compiler modify each function call, inserting some code that logs the call, its caller (or the complete call stack) and probably how much time it required.
- This technique incurs significant overhead, especially if the code contains many functions with short runtime.The instrumentation code will try to compensate for that, but there is always some residual uncertainty.

Sampling Instrumentation

- The program is interrupted at periodic intervals, e.g., 10 milliseconds, and the program counter (and possibly the current call stack) is recorded.
- Necessarily this process is statistical by nature, but the longer the code runs, the more accurate the results will be.
- If the compiler has equipped the object code with appropriate information, sampling can deliver execution time information down to the source line and even machine code level.
- Instrumentation is necessarily limited to functions or basic blocks (code with one entry and one exit point with no calls or jumps in between) for efficiency reasons..

Tools used

Gprof

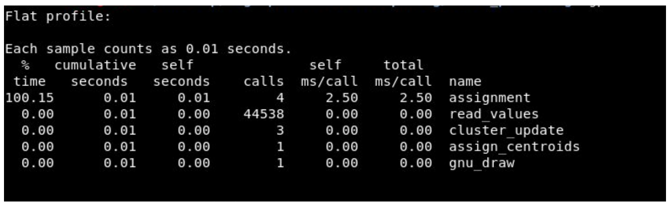
Profiling tool is gprof from the GNU binutils package.it uses both sampling and instrumentation and gives us a flat profile and call graph which gives various insights in the users code.

Commands used:

- gcc -pg sample.c
- ./a.out
- gprof -b a.out gmon.out

The above commands gives us flat profile and call graph

flat profile:



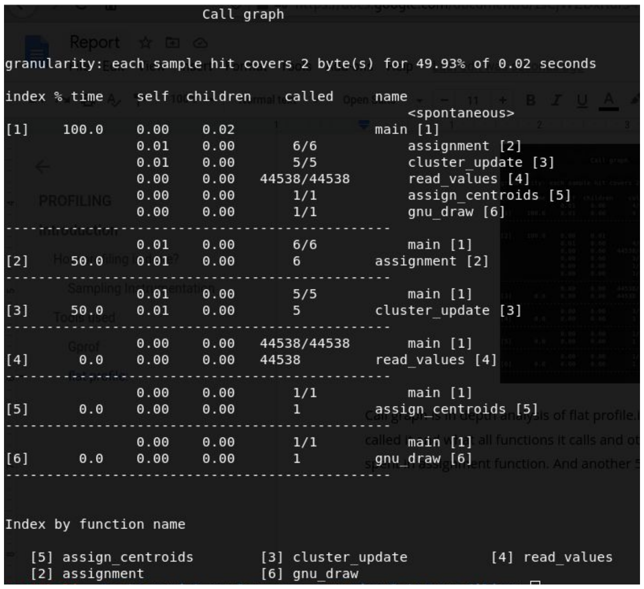
```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  ms/call  ms/call  name    
100.15      0.01     0.01        4     2.50     2.50  assignment
  0.00      0.01     0.00    44538     0.00     0.00  read_values
  0.00      0.01     0.00        3     0.00     0.00  cluster_update
  0.00      0.01     0.00        1     0.00     0.00  assign_centroids
  0.00      0.01     0.00        1     0.00     0.00  gnu_draw
```

The flat profile contains information about execution times of all the program's functions and how often they were called.

Inference :

So as we see in the output even though the function read_values has so many calls to it, it doesn't have a time impact on the program since it does take only inputs and there is nothing computationally intense operation happening.and we see most of the time is spent in assignment function.which is a clear indication of hotspot!.

Call graph:



```
                        Call graph

granularity: each sample hit covers 2 byte(s) for 49.93% of 0.02 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.02                 main [1]
                0.01    0.00       6/6           assignment [2]
                0.01    0.00       5/5           cluster_update [3]
                0.00    0.00   44538/44538       read_values [4]
                0.00    0.00       1/1           assign_centroids [5]
                0.00    0.00       1/1           gnu_draw [6]
-----------------------------------------------
                0.01    0.00       6/6           main [1]
[2]     50.0    0.01    0.00       6         assignment [2]
-----------------------------------------------
                0.01    0.00       5/5           main [1]
[3]     50.0    0.01    0.00       5         cluster_update [3]
-----------------------------------------------
                0.00    0.00   44538/44538       main [1]
[4]      0.0    0.00    0.00   44538         read_values [4]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[5]      0.0    0.00    0.00       1         assign_centroids [5]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[6]      0.0    0.00    0.00       1         gnu_draw [6]
-----------------------------------------------


Index by function name

   [5] assign_centroids        [3] cluster_update          [4] read_values
   [2] assignment              [6] gnu_draw
```

Call graph is in depth analysis of flat profile.it basically says what are all the functions which called it and what all functions it calls and other insights .As we can see 50% of time is spent in assignment function. And another 50% spent in cluster_update function .

Conclusion:

We can say that the ***assignment*** function seems to be the hotspot in the code and can be a potential bottleneck if the input is large in size.