HP-SEE

Profiling with GNU GProf

www.hp-see.eu

Aleksandar Jovic
Institute of Physics Belgrade, Serbia
Scientific Computing Laboratory
ajovic@ipb.ac.rs

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

The HP-SEE initiative is co-funded by the European Commission under the FP7 Research Infrastructures contract no. 261499

Profiling-Introduction

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- Profiling allows you to learn where your program spent its time and which functions called which other functions while it was executing
- This information can show you which pieces of your program are slower than you expected, and might be candidates for rewriting to make your program execute faster
- It can also tell you which functions are being called more or less often than you expected
- This may help you spot bugs that had otherwise been unnoticed
- History:
  - `prof` (1979.)
  - `gprof` (1982.) GNU `gprof` was written by Jay Fenlason

Gprof-Introduction

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- The gprof utility produces an execution profile of C, Pascal, or Fortran77 programs.
- Detail time statistics for each subroutine
- Create relative graph for all subroutines
- Analysis the program bottleneck
- It's a very powerful program
- The simplest one

Gprof-Introduction

- Profiling steps:
  - Compiling a program for profiling
  - Executing the program (You must execute your program to generate a profile data file)
  - You must run `gprof` to analyze the profile data
- 2 forms of output are available from the analysis:
  - *flat profile*
  - *call graph*

Compiling a program for profiling

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- I assume that you know how to write, compile, and execute programs.
- Recompile the original source code with flag `–pg`
- This option `–pg` affects both compiling and linking
- `$ gcc –pg sourcecode.c -o` *`executablefile`*
- `[ajovic@ui moj_C]$ gcc –pg savrsen.c –o savrsen`
- If you compile only some of the modules of the program with `-pg', you can still profile the program, but you won't get complete information about the modules that were compiled without `-pg'. The only information you get for the functions in those modules is the total time spent in them; there is no record of how many times they were called, or from where.

Executing the program

- Your program will write the profile data into a file called `` `gmon.out' `` just before exiting. If there is already a file called `` `gmon.out' ``, its contents are overwritten
- Run the program:
  - `[ajovic@ui moj_C]$ ./savrsen`
  - `[ajovic@ui moj_C]$ ls`
  - You should now see a file in the same directory called `gmon.out`. This file is used by `gprof` to build your profile report
- Run `gprof`:
  - `$ gprof` *List_of_options ExecuteFile* `gmon.out` > OutputFile
  - *List_of_options* can be omitted
  - *ExecuteFile* can be omitted when the file name is `a.out`
- Run `gprof` using the following syntax
  - `gprof savrsen gmon.out > output.txt`

List of options

- List of options:
  - `-b` omit the table or data illustration on *output file*
  - `-e(E) subroutine_name` exclude the subroutine `subroutine_name` from the table (and exclude its elapsed time). The `-e` option tells gprof to not print information about the *subroutine_name* (and its children) in the call graph
  - `-f(F) subroutine_name`: only display the subroutine SRName on the table (and its elapsed time). The `-f` option causes gprof to limit the call graph to the function *function_name* and its children
  - `-Z` only display all subroutines table which are unused on the program
  - `-v` causes gprof to print the current version number, and then exit

Flat profile

- *Flat profile* - The *flat profile* shows the total amount of time your program spent executing each function
  - `[ajovic@ui moj_C]$ gcc -pg eratosten.c -o eratosten`
  - `[ajovic@ui moj_C]$ gprof -b eratosten gmon.out > erat.txt`
  - `[ajovic@ui moj_C]$ vim erat.txt`

```
Each sample counts as 0.01 seconds.
%     cumulative    self                 self     total
time    seconds     seconds   calls   ms/call    ms/call     name
56.1    0.80        0.80      1        796.79     796.79     make
30.54    1.23       0.43   100000001    0.00        0.00    isprime
9.94    1.37        0.14                            3.35     main
```

  - `% time:` the percent of self seconds from total program elapsed time
  - `cumulative seconds` : the seconds cumulate from self seconds
  - `self seconds` : total elapsed time called by its parents, not including its children's elapsed time. Equal to (self s/call)*(calls)
  - `calls` : total number for each subroutine called by its parents

Flat profile

- `self s/call` **:** elapsed time for each time called by its parents, not including its children's elapsed time

- `total s/call` **:** total elapsed time called by its parents, including its children's elapsed time

- `name` : subroutine name

Call graph

- *Call graph* - The call graph shows how much time was spent in each function and its children
  - `$ ifort -pg primer_gprof.f -o f_primer`
  - `$ gprof –b f_primer gmon.out > output.txt`
  - `$ vim output.txt`
- Primary line :
  - `Index   % time    self  children    called     name`
  - `Index` – Each function has an index number, which appears at the beginning of its primary line
  - `% time` – This is the percentage of the total time that was spent in this function, including time spent in subroutines called from this function
  - `self` - This is the total amount of time spent in this function
  - `children` - This is the total amount of time spent in the subroutine calls made by this function
  - `called` - This is the number of times the function was called
  - `name` – This is the name of the current function

Example1

```
granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
            0.00     0.00        1/2           fizika_ [3]
            0.00     0.00        1/2           matematika_ [4]
[1]          0.0     0.00     0.00       2          hemija_ [1]
-----------------------------------------------
                0.00     0.00        1/1           main [223]
[2]      0.0     0.00     0.00        1         MAIN__ [2]
                0.00     0.00        1/1           matematika_ [4]
                0.00     0.00        1/1           fizika_ [3]
-----------------------------------------------
                0.00     0.00        1/1           MAIN__ [2]
[3]      0.0     0.00     0.00        1         fizika_ [3]
                0.00     0.00        1/2           hemija_ [1]
-----------------------------------------------
                0.00     0.00        1/1           MAIN__ [2]
[4]      0.0     0.00     0.00        1         matematika_ [4]
                0.00     0.00        1/2           hemija_ [1]
-----------------------------------------------
```

Example2

```
[ajovic@ui moj_C]$ gcc -pg eratosten.c -o eratosten
[ajovic@ui moj_C]$ gprof -b eratosten gmon.out > erat.txt
[ajovic@ui moj_C]$ vim erat.txt
```

```
granularity: each sample hit covers 2 byte(s) for 0.73% of 1.37 seconds

index % time    self  children    called                      name
                                                          <spontaneous>
[1]    100.0    0.14    1.23                              main [1]
                0.80    0.00          1/1                     make [2]
                0.43    0.00  100000001/100000001               isprime [3]
-----------------------------------------------------------------------
                0.80    0.00    1/1                           main [1]
[2]     58.1    0.80    0.00    1                         make [2]
-----------------------------------------------------------------------
                0.43    0.00    100000001/100000001             main [1]
[3]     31.6    0.43    0.00    100000001                 isprime [3]
-----------------------------------------------------------------------
```

References

- http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html