GPROF

Introdução à Avaliação de Desempenho e Dependabilidade de Sistemas

Professor Paulo Maciel – prmm@cin.ufpe.br

Agenda

- O que é o Gprof?
- Por que usar um *profiler?*
- Como utilizar o Gprof?
  - Compilando um programa para *profiling*
  - Executando um programa para gerar *profile data*
  - Executando o Gprof
- Interpretando a saída do Gprof
  - Flat Profile
  - Call Graph

O que é o Gprof?

GProf é uma ferramenta para análise dinâmica da execução de programas (*profiler)*.

Ele produz um “perfil” da execução de programas C, Pascal ou Fortran77.

Essa ferramenta pode ser usada em conjunto com o GCC.

O Gprof é um projeto GNU, sob a licença GNU GPL.

O que é o Gprof?

Gprof reporta informação de tempo de execução de cada uma das funções do programa.

Gprof também mostra uma árvore de chamada de funções do programa, ou seja, podemos verificar o tempo de execução de uma dada função e de suas sub-rotinas.

Por que usar um *profiler?*

- *Profiling* é importante porque nos permite verificar ONDE o programa está gastando tempo.

Essa informação mostra quais partes do programa estão executando mais lentamente do que você esperava.

Essas partes do programa (funções) são candidatas a serem reescritas visando uma melhora no desempenho do seu programa.

Por que usar um *profiler?*

- *Profiling* é importante também para sabermos qual função chamou quais outras funções durante a execução.

Essa informação mostra quais funções estão sendo chamadas mais ou menos frequentemente que você esperava.

Esse tipo de informação é útil para identificar *bugs* que ainda não haviam sido percebidos na execução do programa.

Por que usar um *profiler?*

Como o Gprof utiliza uma informação que é gerada em tempo de execução do progama, então ele pode ser usado em programas que são muito grandes ou muito complexos para serem analisados simplesmente lendo o código fonte.

Como utilizar o Gprof?

São 5 passos:

1. Tenha um programa que funcione (O Gprof é somente um *profiler,* ele não é um *debugger*)
2. *Compilar* e *Linkar* o programa com a opção de *profiling* habilitada (-pg)
3. Executar o programa normalmente
4. Executar o Gprof
5. Analisar a informação gerada pelo Gprof

Como utilizar o Gprof?

Se não estiver instalado o compilador e o gprof, utilize os comandos:

apt-get install build-essential

apt-get install binutils

Como utilizar o Gprof?

Compilando um programa para *profiling*

Utilize a opção -pg além das opções que você já utiliza na hora de *compilar* e *linkar.*

Exemplo:

```
gcc -o meuprograma meuprograma.c -g -pg
```

Como utilizar o Gprof?

Executando um programa para gerar *profile data*

Execute o programa normalmente, como se nada tivesse acontecido.

Entretanto talvez a execução do programa seja um pouco mais lenta, pois existe um gasto de tempo ao coletar informação do tempo de funções, além da escrita do *profile data.*

Como utilizar o Gprof?

Executando um programa para gerar *profile data*

Note que o *profile data* gerado será extremamente dependente da entrada que você dará ao seu programa.

Exemplo:

Se o primeiro comando que você der ao seu programa for “exit(0)”, então seu *profile data* só terá o tempo de inicialização e *cleanup* do programa.

Como utilizar o Gprof?

Executando o Gprof

Após executar o seu programa, ele agora gerou um arquivo chamado “gmon.out” no diretório onde está o executável.

Note que seu programa deve ter permissão para escrever no diretório onde o mesmo está rodando, caso contrário será lançado um erro.

Como utilizar o Gprof?

Executando o Gprof

Agora que temos nosso *profile data* “gmon.out”, podemos executar o Gprof.

Exemplo:

```
gprof meuprograma > run4.withO2.stats
```

Interpretando a saída do Gprof

- Ao analisar a saída do Gprof, no nosso exemplo é o arquivo “run4.withO2.stats”, percebemos que ele é dividido em duas seções:

- Flat Profile

- Call Graph

Flat Profile

- **% time** - The percentage of the total running time of the program used by this function.
- **Cumulative seconds** - A running sum of the number of seconds accounted for by this function and those listed above it.
- **Self seconds** - The number of seconds accounted for by this function alone. This is the major sort for this listing.
- **Calls** - The number of times this function was invoked, if this function is profiled, else blank.
- **Self ms/call** - The average number of milliseconds spent in this function per call, if this function is profiled, else blank.
- **Total ms/call** - The average number of milliseconds spent in this function and its descendent per call, if this function is profiled, else blank.
- **Name** - The name of the function. This is the minor sort for this listing. The index shows the location of the function in the gprof listing. If the index is in parenthesis it shows where it would appear in the gprof listing if it were to be printed.

Interpretando a saída do Gprof

- Flat Profile

Mostra quanto tempo o programa gastou em cada função, e também quantas vezes a função foi chamada.

A informação é bem clara e podemos rapidamente identificar qual função está “queimando” mais ciclos no programa.

Call Graph

- **Index** - A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function is in the table.
- **% time** - This is the percentage of the `total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc., these numbers will NOT add up to 100%.
- **Self** - This is the total amount of time spent in this function. For function's parents, this is the amount of time that was propagated directly from the function into this parent. While, for function's children, this is the amount of time that was propagated directly from the child into the function.
- **Children** - This is the total amount of time propagated into this function by its children. For the function's parents, this is the amount of time that was propagated from the function's children into this parent. While, for the function's children, this is the amount of time that was propagated from the child's children to the function.
- **Called** - This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls. For the function's parents, this is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'. While, for the function's children, this is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.
- **Name** - The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number. For function's parents, this is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number. For function's children, this is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

Interpretando a saída do Gprof

- Call Graph

Mostra, para cada função, quais funções à chamaram, e também quais outras funções ela chamou, e quantas vezes.

Mostra também uma estimativa de quanto tempo foi gasto nas sub-rotinas de cada função.

Gprof - opções

- Existem várias opções de execução do Gprof

Seguem algumas opções interessantes:

Linux / Unix Command: *gprof*

Command Library

NAME

gprof - display call graph profile data

SYNOPSIS

gprof [-[abcDhilLsTvwxyz]] [-[ACeEfFJnNOpPqQZ][*name*]]
[-I *dirs*] [-d[*num*]] [-k *from/to*]
[-m *min-count*] [-t *table-length*]
[--[no-]annotated-source[=*name*]]
[--[no-]exec-counts[=*name*]]
[--[no-]flat-profile[=*name*]] [--[no-]graph[=*name*]]
[--[no-]time=*name*] [--all-lines] [--brief]
[--debug[=*level*]] [--function-ordering]
[--file-ordering] [--directory-path=*dirs*]
[--display-unused-functions] [--file-format=*name*]
[--file-info] [--help] [--line] [--min-count=*n*]
[--no-static] [--print-path] [--separate-files]
[--static-call-graph] [--sum] [--table-length=*len*]
[--traditional] [--version] [--width=*n*]
[--ignore-non-functions] [--demangle[=*STYLE*]]
[--no-demangle] [*image-file*] [*profile-file* ...]

Gprof - opções

- A opção --function-ordering

Faz com que o Gprof mostre uma sugestão de “ordenação” de chamada de funções do programa baseado no *profile data*.

Essa sugestão tem a intenção de melhorar paginação, tlb e comportamento cache (para programas em sistemas que suportem ordenação arbitrária de funções em um executável)

TLB é uma cache que o hardware de gerenciamento de memória utiliza para melhorar a velocidade da tradução de endereços virtuais.

Gprof - opções

- Se existem funções estáticas no programa e você deseja omiti-las do profile, utilize a opção -a

- Se você não deseja mais que o gprof imprima no arquivo de saída as definições de cada coluna, você pode suprimir essas informações extras com a flag -b

Gerando um DOT Graph

- Existe uma ferramenta que recebe como entrada a informação gerada pelo Gprof (e também outros profilers) e converte essa informação em um DOT Graph.

- A ferramenta se chama “GProf2Dot”.

- https://code.google.com/p/jrfonseca/wiki/Gprof2Dot