Cronometraje y Perfilado - Perfilado de Subrutinas*

José Enrique Alvarez Estrada

Translated By:

José Enrique Alvarez Estrada

Based on *Timing and Profiling - Subroutine Profiling*† by

Charles Severance

Kevin Dowd

A veces va a desear más detalle que el que proporciona el cronometraje global de la aplicación, pero tal vez no tenga tiempo de modificar el código para insertar varios cientos de llamadas a *etime* en su código. Los perfiles también son muy útiles cuando le han entregado un extraño programa de 20,000 líneas de código, y le han pedido que averigüe cómo funciona y luego mejore su rendimiento.

La mayoría de los compiladores proporcionan la facilidad de insertar llamadas cronométricas automáticas en su código, tanto a la entrada como a la salida de cada rutina, a tiempo de compilación. Cuando se ejecute su programa, se registrarán los tiempos de entrada y salida, y luego se volcarán en un archivo. Una herramienta separada resume los patrones de ejecución y produce un reporte que muestra el porcentaje del tiempo gastado en cada una de las rutinas que usted escribió, así como en las rutinas de biblioteca.

El perfil debe proporcionarle a usted un sentido de la forma en que se está ejecutando. Esto es, puede ver que el 10% del tiempo se gasta en la subrutina A, el 5% en la subrutina B, etc. Naturalmente, si junta todas las subrutinas deben sumar el 100% del tiempo de ejecución global. A partir de estos porcentajes puede construir usted una imagen - un *perfil* — de la distribución de la ejecución cuando el programa corre. Aunque no sean representativos de alguna herramienta de perfilado en particular, los histogramas en Figure 1 (Perfil Agudo - Dominado por la Rutina 1) y Figure 2 (Perfil plano - ninguna rutina predomina) representan estos porcentajes, ordenados de izquierda a derecha, donde cada columna vertical representa una rutina diferente. Ayudan a ilustrar las diferentes formas de perfil.

*Version 1.2: Oct 19, 2011 12:07 pm +0000
†http://cnx.org/content/m33713/1.3/
‡http://creativecommons.org/licenses/by/3.0/

Perfil Agudo - Dominado por la Rutina 1

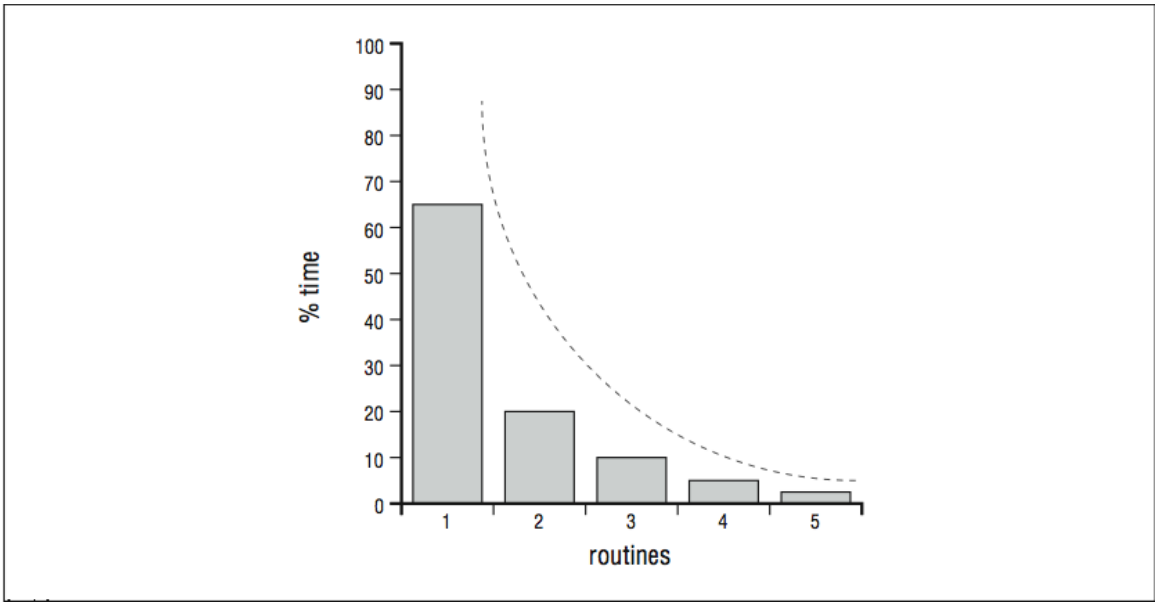

Figure 1

Un *perfil agudo* nos dice que la mayoría del tiempo se está gastando en uno o dos procedimientos, y que si quiere usted mejorar el rendimiento del programa, debe enfocar sus esfuerzos en afinar dichos procedimientos. Una optimización menor en una línea de código que se ejecuta intensamente puede a veces tener un enorme efecto en el tiempo de ejecución global, dando la oportunidad correcta. Un *perfil plano*,[1] por otra parte, le indica que el tiempo de ejecución se reparte entre muchas rutinas, y el esfuerzo gastado en optimizar cualquiera de ellas será poco benéfico para acelerar el programa. Por supuesto, también hay programas cuyo perfil de ejecución cae en algún punto intermedio.

[1] A menudo e abusa un poco del término "perfil plano". Lo estamos usando para describir un perfil que muestra una distribución homogénea de tiempo a lo largo del programa. También verá que se emplea para marcar una diferencia de un perfil de un perfil del grafo de llamadas, como se describe más abajo.

Perfil plano - ninguna rutina predomina

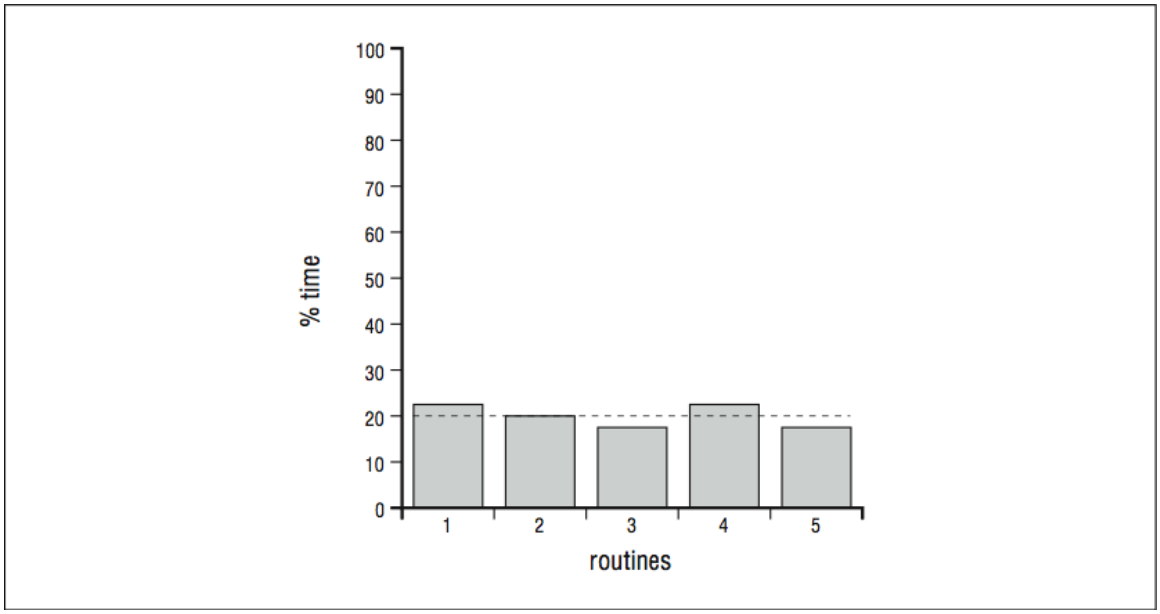

Figure 2

No podemos predecir con absoluta certeza qué será lo que encuentre usted cuando perfile sus programas, pero hay algunas tendencias generales. Por ejemplo, los códigos científicos e ingenieriles construidos alrededor de soluciones matriciales a menudo exhiben perfiles muy agudos. El tiempo de ejecución está dominado por el trabajo realizado en un puñado de rutinas. Para afinar el código, necesita enfocar sus esfuerzos en tales rutinas, hasta hacerlas más eficientes. Puede que ello involucre reestructurar ciclos para exponer el paralelismo, proporcionar indicios al compilador, o reacomodar las referencias a memoria. En cualquier caso, el reto es tangible; puede ver los problemas que debe solucionar.

Por supuesto, existen límites a la mejora en tiempo de ejecución que obtendrá afinando una o dos rutinas. Una regla de oro citada frecuentemente es la *Ley de Amdahl*, obtenida a partir de las observaciones realizadas en 1967 por uno de los diseñadores de la serie 360 de IBM y fundador de Amdahl Computer, Gene Amdahl. Estrictamente hablando, sus observaciones fueron acerca del potencial de rendimiento de las computadoras paralelas, pero la gente ha adaptado la Ley de Amdahl para describir otras cosas por igual. Para nuestros propósitos, se puede citar así: digamos que tiene un programa con dos partes, una que puede optimizarse hasta hacerla infinitamente rápida, y otra que no puede optimizarse en absoluto. Incluso si la porción optimizable supone como mucho el 50% del tiempo de ejecución inicial, cuando mucho será usted capaz de recortar el tiempo de ejecución total a la mitad. Esto es, su tiempo de ejecución estará dominado eventualmente por la porción que no puede optimizarse. Esto pone un límite máximo a sus expectativas de afinación.

Incluso a pesar del limitado retorno de inversión que sugiere la Ley de Amdahl, afinar un programa con un perfil agudo puede dar sus recompensas. Los programas con perfiles planos son mucho más difíciles de afinar. A menudo se trata de códigos de sistema, aplicaciones no numéricas y variedades de códigos numéricos sin soluciones matriciales. Se requiere de un enfoque de afinación global para reducir, a un nivel justificable, el tiempo de ejecución de un programa con un perfil plano. Por ejemplo, a veces puede usted optimizar el

uso de la cache de instrucciones, lo cuál resulta complicado por la distribución equitativa de actividades del programa entre un gran número de rutinas. También puede ayudar a reducir la sobrecarga del llamado a subrutinas, plegando los invocados dentro de los invocadores. Ocasionalmente, puede encontrar un problema de referencia a memoria que es endémico al programa completo - y uno que puede arreglarse de un solo golpe.

Cuando estudie un perfil, puede que encuentre un porcentaje inusualmente grande de tiempo gastado en rutinas de biblioteca, tales como `log`, `exp` o `sin`. A menudo tales funciones se están realizando en rutinas de software, en vez de en línea. Puede reescribir su código para eliminar algunas de estas operaciones. Otro patrón importante que debe buscarse es aquel en el cual una rutina consume mucho más tiempo del que usted espera. Un tiempo de ejecución inesperado puede indicar que está accediendo a la memoria en un patrón que resulta malo desde el punto de vista del rendimiento, o que algún aspecto del código no puede optimizarse apropiadamente.

En cualquier caso, para obtener un perfil se requiere de una herramienta de perfilado. Uno o dos *perfiladores de subrutinas* vienen de forma estándar con los ambientes de desarrollo de software de todas las máquinas UNIX. Discutiremos dos de ellos: *prof* y *gprof*. Además, mencionaremos unos pocos perfiladores que actúan línea por línea. Los perfiladores de subrutina pueden proporcionarle a usted una vista global general acerca de dónde está consumiéndose el tiempo. Probablemente deba comenzar con *prof*, si lo tiene (lo cuál es común en muchas máquinas). De otra forma, use *gprof*. Después de eso, puede moverse a un perfilador de línea por línea, si requiere conocer cuáles sentencias requieren de más tiempo.

1 prof

prof es la herramienta de perfilado más común en UNIX. En cierto sentido, es una extensión del compilador, enlazador y de las bibliotecas de código objeto, más unas pocas utilidades extras, de forma que es difícil hallar una sola cosa y decir "esto es lo que perfila su código." *prof* trabaja muestreando periódicamente el contador de programa conforme se ejecuta su aplicación. Para permitir el perfilado, usted debe recompilarla y re-enlazarla usando la bandera `-p`. Por ejemplo, si su programa tiene dos módulos, *stuff.c* y *junk.c*, requiere compilar y enlazar de acuerdo al código siguiente:

```
% cc stuff.c -p -O -c
% cc junk.c -p -O -c
% cc stuff.o junk.o -p -o stuff
```

Esto crea un binario *stuff* que está listo para el perfilado. Usted no requiere hacer nada especial para ejecutarlo. Sólo trátelo normalmente, tecleando `stuff`. Como se están recolectando las estadísticas a tiempo de ejecución, toma un poco más de lo usual en ejecutarse. [2] Una vez completado, habrá un nuevo archivo llamado *mon.out* en el directorio donde lo ejecutó. Este archivo contiene la historia de *stuff* en forma binaria, así que no puede observarla directamente. Use la utilidad *prof* para leer *mon.out* y crear un perfil de *stuff*. Por defecto, la información se escribe a su pantalla porque es la salida estándar, aunque puede redirigirla fácilmente a un archivo:

```
% prof stuff > stuff.prof
```

[2] Recuerde: el código con el perfilado activo toma más en ejecutarse. Debe recompilar y re-enlazar el programa completo *sin* la bandera `-p` cuando haya finalizado el perfilado.

Para poder explorar cómo funciona el comando *prof*, hemos creado la siguiente aplicación, *bucles.c*, que es pequeña y ridícula. Contiene una rutina principal y tres subrutinas para las cuáles puede usted predecir la distribución de tiempo, con sólo observar el código.

```
main () {
    int l;
    for (l=0;l<1000;l++) {
        if (l == 2*(l/2)) foo ();
        bar();
        baz();
    }
}
foo (){
    int j;
    for (j=0;j<200;j++)
}
bar () {
    int i;
    for (i=0;i<200;i++);
}
baz () {
    int k;
    for (k=0;k<300;k++);
}
```

Nuevamente, debe compilar y enlazar *bucles* con la bandera `-p`, ejecutar el programa, y luego ejecutar la utilidad *prof* para extraer un perfil, tal como sigue:

```
% cc bucles.c -p -o bucles
% ./bucles
% prof bucles > bucles.prof
```

El siguiente ejemplo muestra cómo debe verse *bucles.prof*. Hay seis columnas.

```
%Time Seconds Cumsecs #Calls msec/call Name
 56.8    0.50    0.50   1000    0.500  _baz
 27.3    0.24    0.74   1000    0.240  _bar
 15.9    0.14    0.88    500    0.28   _foo
  0.0    0.00    0.88      1    0.     _creat
  0.0    0.00    0.88      2    0.     _profil
  0.0    0.00    0.88      1    0.     _main
  0.0    0.00    0.88      3    0.     _getenv
```

```
 0.0    0.00    0.88       1    0.      _strcpy
 0.0    0.00    0.88       1    0.      _write
```

Las columnas pueden describirse como sigue:

- `%Time` Porcentaje de tiempo de CPU consumido por esta rutina
- `Seconds` Tiempo de CPU consumido por esta rutina
- `Cumsecs` Un acumulado del tiempo consumida por esta rutina y todas las que la preceden en la lista
- `Calls` El número de veces que fue llamada esta rutina en particular
- `msec/call` Segundos divididos entre el número de llamadas, dando el promedio de tiempo tomado por cada invocación de la rutina
- `Name` El nombre de esta rutina

Las tres rutinas superiores listadas son del propio *bucles.c*. Puede observar una entrada que representa a la rutina principal (main), ubicada en la segunda mitad de la lista. Dependiendo del vendedor, los nombres de las rutinas pueden contener caracteres de guión bajo como prefijo o sufijo, y siempre habrá listadas algunas rutinas que usted no reconozca. Son contribuciones de la biblioteca C y posiblemente de las bibliotecas de FORTRAN, si está usando este lenguaje. El perfilado también introduce algo de sobrecarga en la ejecución, que a menudo se muestra como una o dos subrutinas en la salida de *prof*. En este caso, la entrada llamada `_profil` representa el código insertado por el enlazador para recolectar los datos de perfilado a tiempo de ejecución.

Si fuera nuestra intención afinar *bucles*, deberíamos considerar que un perfil como el mostrado en la figura superior es un buen signo. La primera rutina en la lista toma 50% del tiempo, así que cuando menos hay una posibilidad de que hagamos algo con ella que pueda tener un impacto significativo en el rendimiento global. (Por supuesto, con un programa tan trivial como *bucles*, hay mucho que podemos hacer, puesto que *bucles* no hace nada.)

2 gprof

Así como es importante conocer cómo se distribuye el tiempo durante la ejecución de su programa, también es valioso ser capaz de decir quién invocó a quién en la lista de rutinas. Imagine, por ejemplo, si algo etiquetado como `_exp` aparece posicionado alto en la lista de salida de prof. Usted puede decir: "Mmmm, no recuerdo haber invocado a nada llamado `exp()`. ¡No tengo ni idea de dónde viene!" Un árbol de llamados a subrutinas le ayudará a encontrarlo.

Puede pensarse en las subrutinas y funciones como miembros de un árbol familiar. En la parte más alta del árbol, o raíz, está una rutina que precede a la rutina principal que usted codificó en su aplicación; es quien llama a su rutina principal, que a su vez llama a otras, y así sucesivamente, descendiendo hasta los nodos hoja del árbol. El nombre apropiado para este árbol es un *grafo de llamadas*.[3] La relación entre rutinas y nodos en el grafo es uno de padres e hijos. Nos referimos a los nodos separados por más de un salto como ancestros y descendientes.

La Figura 6-4 muestra gráficamente la clase de grafo de llamadas que puede verse en una aplicación pequeña. `main` es el padre o ancestro de la mayoría de las demás rutinas. `G` tiene dos padres, `E` y `C`. Otra rutina, `A`, no parece tener ningún ancestro o descendiente. Este problema puede suceder cuando no se compilan las rutinas con el perfilado activo, o cuando no han sido invocadas - tal como pudiera ser el caso si `A` fuera un manejador de excepciones.

El software de perfilado de UNIX que puede extraer esta clase de información se llama *gprof*. Replica las habilidades de *prof*, además de proporcionar un perfil de grafo de llamados, de forma que pueda usted observar quién llama a quién, y cuán frecuentemente. El perfil del grafo de llamadas es útil si está tratando de averiguar cómo trabaja un fragmento de código, o de dónde proviene una rutina desconocida, o si está buscando candidatos para hacer una subrutina en línea.

[3] No tiene por qué ser un árbol. Cualquier subrutina puede tener más de un padre. Es más, las rutinas recursivas introducen ciclos en el grafo, durante los cuales los hijos llaman a uno de sus padres.

Para usar el perfilado del grafo de llamados, usted debe seguir los mismos pasos que con *prof*, excepto que la bandera `-pg` se sustituye por la bandera `-p`.[4] Adicionalmente, cuando llega el momento de producir el perfil, debe usted usar el programa de utilidad *gprof* en vez de *prof*. Una diferencia más es que el nombre del archivo de estadísticas es *gmon.out* en vez de *mon.out*:

```
% cc -pg stuff.c -c
% cc stuff.o -pg -o stuff
% stuff
% gprof stuff > stuff.gprof
```

Grafo de llamadas simple

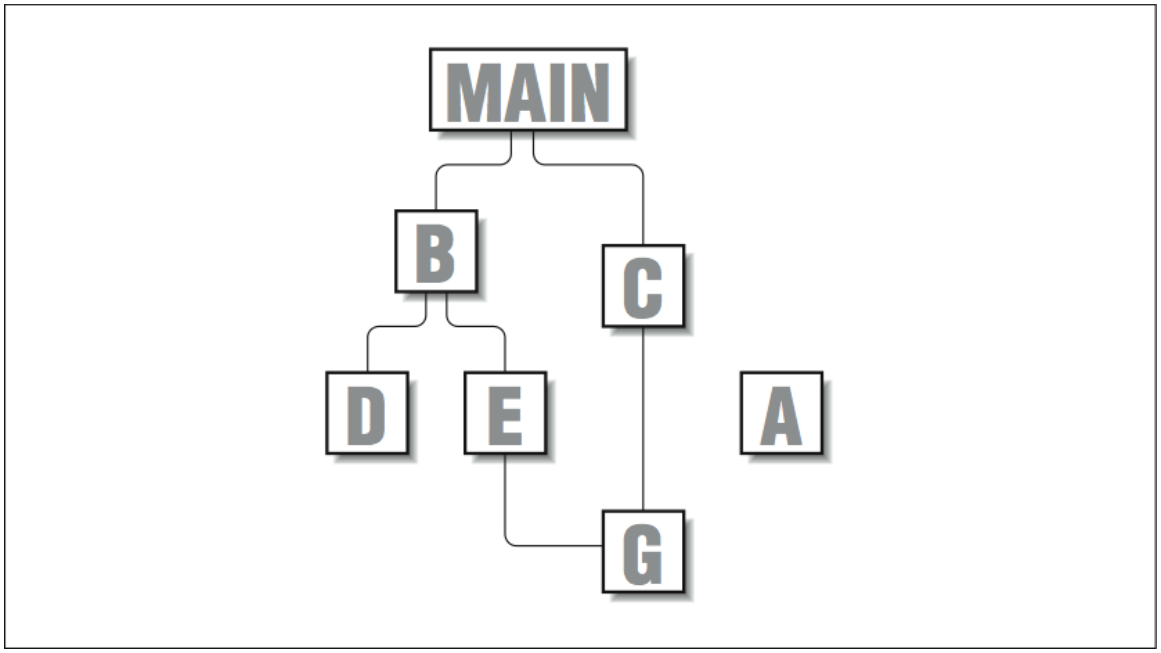

Figure 3

La salida de `gprof` se divide en tres secciones:

- Perfil del grafo de llamadas
- Perfil de cronometraje
- Índice

[4] En las máquinas HP, la bandera es `-G`.

La primera sección es textualmente el mapa del grafo de llamadas. La segunda lista las subrutinas, el porcentaje de tiempo dedicado a cada una, el número de llamadas, etc. (similar a *prof*). La tercera sección es una referencia cruzada, de forma que pueda usted localizar las rutinas por número, en vez de por nombre. Esta sección resulta especialmente útil en aplicaciones grandes, porque las rutinas se ordenan basadas en la cantidad de tiempo que emplean, y puede resultar difícil localizar una en particular buscando su nombre. Inventemos otra aplicación trivial para ilustrar cómo funciona *gprof*. Figure 4 (FORTRAN example) muestra una pequeña pieza de código FORTRAN, junto con un diagrama que indica cómo están conectadas las rutinas. Tanto la subrutina `A` como la `B` son invocadas por `MAIN`, y, a su vez, cada una llama a `C`. El siguiente ejemplo muestra una sección de la salida del perfil del grafo de llamadas de *gprof*:[5]

FORTRAN example

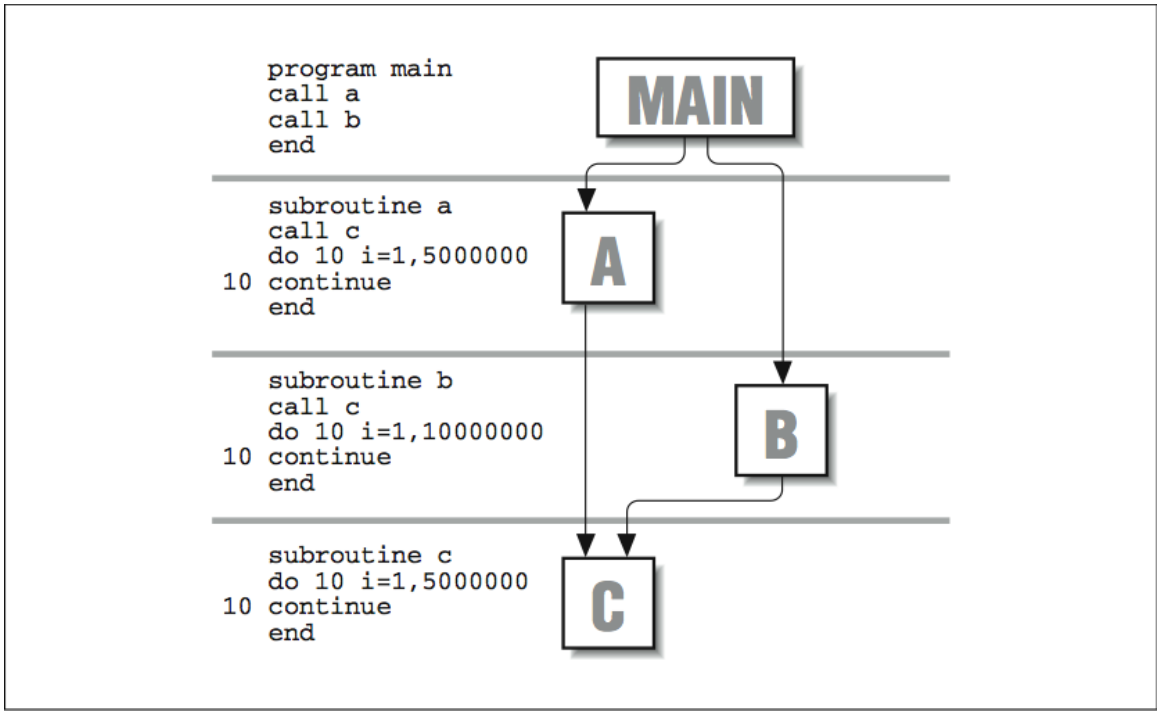

Figure 4

```
                                    called/total       parents
index  %time     self descendants   called+self    name       index
                                    called/total       children
```

[5]Con el fin de no consumir demasiado espacio, recortamos la sección más relevante para nuestra discusión y la incluimos en este ejemplo. Hay mucho más, incluyendo llamados a rutinas de configuración y sistema, del tipo de las que ve cuando ejecuta *gprof*.

```
  ....                          ....                          ....

                    0.00          8.08      1/1           _main [2]
[3]     99.9        0.00          8.08      1         _MAIN_ [3]
                    3.23          1.62      1/1           _b_ [4]
                    1.62          1.62      1/1           _a_ [5]

-----------------------------------------------

                    3.23          1.62      1/1           _MAIN_ [3]
[4]     59.9        3.23          1.62      1         _b_ [4]
                    1.62          0.00      1/2           _c_ [6]

-----------------------------------------------

                    1.62          1.62      1/1           _MAIN_ [3]
[5]     40.0        1.62          1.62      1         _a_ [5]
                    1.62          0.00      1/2           _c_ [6]

-----------------------------------------------

                    1.62          0.00      1/2           _a_ [5]
                    1.62          0.00      1/2           _b_ [4]
[6]     39.9        3.23          0.00      2         _c_ [6]
```

Emparedada entre cada conjunto de líneas punteadas está la información que describe una rutina dada y su relación respecto a padres e hijos. Es fácil decir qué rutina representa cada bloque, porque el mismo está desplazado más a la izquierda que los otros. Los padres se listan arriba, los hijos abajo. Como sucedió con `prof`, se agregaron guiones bajos a las etiquetas.[6] A continuación se muestra una descripción de cada una de las columnas:

- `index` Observará que cada nombre de rutina tiene asociado un número entre corchetes (`[n]`). Se trata de una referencia cruzada para ubicar rutina en cualquier parte del perfil. Si, por ejemplo, está usted buscando en el bloque que describe `_MAIN_` y quiere saber más acerca de sus hijos, digamos `_a_`, puede encontrarlo recorriendo hacia abajo la parte izquierda de la página en busca de su índice, `[5]`.
- `%time` El significado del campo `%time` es un poco diferente del que tiene para *prof*. En este caso describe el porcentaje de tiempo gastado en esta rutina *mas* el tiempo gastado en todos sus hijos. Le proporciona una forma rápida de determinar dónde encontrar las partes más ocupadas del grafo de llamadas.
- `self` Listado en segundos, la columna `self` tiene diferentes significados para los padres, la rutina en cuestión y sus hijos. Comenzando con la entrada central -la rutina misma- el valor `self` muestra cuánto tiempo global se dedicó a la rutina. En el caso de `_b_`, por ejemplo, esta cantidad fue de 3.23 segundos. Cada entrada en la columna `self` muestra la cantidad de tiempo que puede atribuirse a llamadas desde los padres. Si observa la rutina `_c_`, por ejemplo, verá que consumió un tiempo total de 3.23 segundos. Pero note que tuvo dos padres: 1.62 segundos del tiempo se pueden atribuir a llamadas provenientes de `_a_`, y 1.62 segundos a las de `_b_`.
  Para el hijo, la cantidad `self` muestra cuánto tiempo se gastó ejecutando cada hijo, debido a llamadas

[6] Puede que haya observado que hay dos rutinas principales: `_MAIN_` y `_main`. En un programa FORTRAN, `_MAIN_` es la verdadera rutina principal, que es invocada como una subrutina por `_main`, que proporciona una biblioteca del sistema a tiempo de enlace. Cuando perfila usted código en C, no verá `_MAIN_`.

provenientes de esta rutina. Los hijos pueden haber consumido más tiempo global, pero el único tiempo contabilizado es aquél atribuíble a llamadas desde esta subrutina. Por ejemplo, `_c_` acumuló 3.23 segundos globales, pero si observa en el bloque describiendo `_b_`, verá a `_c_` listada como un hijo con sólo 1.62 segundos. Este fue el tiempo total gastado ejecutando `_c_` del lado de `_b_`.

- `descendants` Como ocurren con la columna `self`, los vaores en la columna de descendientes tienen diferentes significados para la rutina, sus padres y los hijos. Para la rutina misma, muestra el número de segundos gastados en todos sus descendientes.
  Para los padres de la rutina, esta columna describe cuánto tiempo gastado por la rutina puede trazarse a partir de llamadas realizadas por cada padre. Observando nuevamente la rutina `_c_`, puede observar que de su tiempo total, 3.23 segundos, 1.62 segundos son atribuíbles a cada uno de sus padres, `_a_` y `_b_`.
  Para los hijos, la columna de descendientes muestra cuánto del tiempo de los hijos puede atribuirse a llamadas realizadas desde esta rutina. El hijo puede haber acumulado más tiempo global, pero aquí sólo se despliega el tiempo asociado con llamadas desde esta rutina.
- `calls` La columna `calls` muestra el número de veces que se invocó cada rutina, así como la distribución de dichas llamadas asociadas tanto con padres como con hijos. Comenzando con la rutina misma, la cantidad en la columna `calls` muestra el número total de entradas a la rutina. En situaciones donde la rutina se invoca a sí misma, también observará usted un $+n$ agregado inmediatamente, mostrando que se realizaron n llamadas recursivas adicionales.
  Las cantidades de padres e hijos se expresan como tasas. Para los padres, la tasa m/n debe leerse como "de las n veces que se invocó la rutina, m vinieron de su padre." Para el hijo, debe leerse como "de las n veces que fue invocado este hijo, m provinieron de esta rutina."

3 Perfil Plano de gprof

Como mencionamos previamente, *gprof* también produce un perfil de cronometraje (también conocido como un perfil "plano", un término algo confuso) similar al que produce *prof*. Unos pocos campos son diferentes de *prof*, y también hay algo de información extra, así que será de ayuda explicarlo brevemente. El siguiente ejemplo muestra unas pocas de las primeras líneas de un perfil plano de *gprof* para el programa *stuff*. Reconocerá del programa original las tres rutinas superiores. Las otras son funciones de biblioteca incluidas a tiempo de enlace.

```
 %    cumulative     self              self     total
time    seconds     seconds    calls  ms/call  ms/call  name
39.9       3.23        3.23        2  1615.07  1615.07  _c_ [6]
39.9       6.46        3.23        1  3230.14  4845.20  _b_ [4]
20.0       8.08        1.62        1  1620.07  3235.14  _a_ [5]
 0.1       8.09        0.01        3     3.33     3.33  _ioctl [9]
 0.0       8.09        0.00       64     0.00     0.00  .rem [12]
 0.0       8.09        0.00       64     0.00     0.00  _f_clos [177]
 0.0       8.09        0.00       20     0.00     0.00  _sigblock [178]
 ...        ....        ....        .        .        .       ......
```

He aquí el significado de cada columna:

- `%time` Nuevamente, vemos un campo que describe el tiempo de ejecución de cada rutina, como un porcentaje del tiempo global del programa. Como puede usted esperar, todas las entradas esta columna deben totalizar (aproximadamente) un 100%.

- `cumulative` seconds. Para una rutina dada, la columna llamada "segundos acumulados" hace un recuento de la suma del tiempo de ejecución ejecución tomado por todas las rutinas precedentes, más el tiempo propio. Conforme la vaya revisando hacia abajo, verá que los números se aproximan asintóticamente al tiempo total del programa.
- `self` seconds La contribución individual de cada rutina al tiempo total de ejecución.
- `calls` El número de veces que fue invocada esta rutina en particular.
- `self ms/call` Los segundos gastados adentro de la rutina, dividido entre el número de llamadas. Ello nos da una duración promedio del tiempo tomado por cada invocación a la rutina. La cantidad está medida en milisegundos.
- `total ms/call` Los segundos gastados adentro de la rutina más sus descendientes, dividido entre el número de llamados.
- `name` El nombre de la rutina. Observe que de nuevo aparece aquí el número de referencia cruzada.

4 Acumulando los Resultados de Varias Corridas de gprof

Es posible acumular estadísticas provenientes de múltiples corridas, de forma que obtenga usted una instantánea del comportamiento del programa con una variedad de conjuntos de datos. Por ejemplo, digamos que quiere usted perfilar una aplicación - llamémosla *bar* — con tres conjuntos de datos distintos. Puede realizar las corridas separadamente, guardando los archivos *gmon.out* conforme las ejecuta, y luego combinar los resultados en un solo perfil al terminar:

```
% f77 -pg bar.f -o bar
% bar < data1.input
% mv gmon.out gmon.1
% bar < data2.input
% mv gmon.out gmon.2
% bar < data3.input
% gprof bar -s gmon.1 gmon.2 gmon.out > gprof.summary.out
```

En el perfil de ejemplo, cada corrida crea un nuevo archivo *gmon.out* que luego renombramos para que no sea encimado por el siguiente. Al final, *gprof* combina la información de cada uno de los archivos de datos para producir un perfil sumario de *bar* en el archivo *gprof.summary.out*. Adicionalmente (aunque no se muestra aquí), *gprof* crea un archivo llamado *gmon.sum* que contiene los datos mezclados a partir de los tres archivos de datos originales. *gmon.sum* tiene el mismo formato que *gmon.out*, de forma que pueda usted usarlo como entrada para producir otros perfiles mezclados si así lo requiere.

Al menos en forma, la salida del perfil mezclado luce exactamente igual que el de una corrida individual. Pero hay un par de cosas interesantes que debemos señalar. Por un lado, la rutina `main` parece haber sido invocada más de una vez - de hecho, una vez por cada corrida. Además, dependiendo de la aplicación, las múltiples ejecuciones tienden o bien a suavizar el contorno del perfil, o bien a exagerar sus características. Puede imaginar cómo es que esto sucede. Si se está invocando constantemente una única rutina, mientras que las otras vienen y van conforme cambian los datos de entrada, ésta toma una importancia creciente en sus esfuerzos de afinación.

5 Unas Pocas Palabras Acerca de la Exactitud

En aquellos procesadores que se ejecutan a 600 MHz o más, el tiempo que transcurre entre muestras de 60 Hz y 100 Hz es una verdadera eternidad. Es más, puede que experimente usted errores de cuantificación

cuando la frecuencia de muestreo es fija, como es el caso en muestras de 1/100 y 1/60 de segundo. Para usar un ejemplo exagerado, asumamos que la línea de tiempo en Figure 5 (Errores de cuantificación al perfilar) muestra invocaciones alternadas a dos subrutinas, `BAR` y `FOO`. Las marcas cronométricas representan los puntos de muestreo del perfilado.

Errores de cuantificación al perfilar

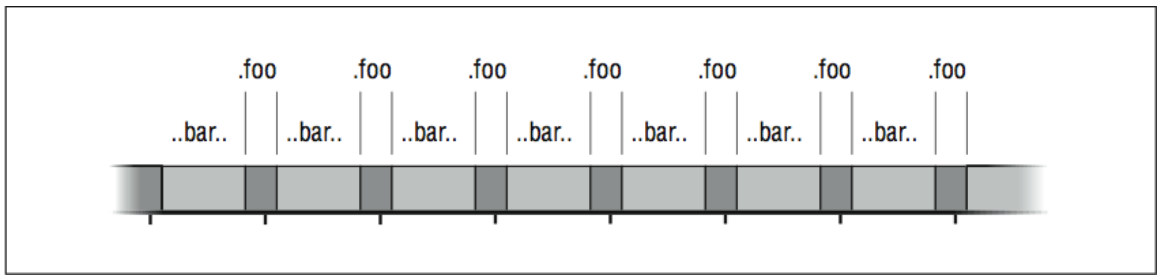

Figure 5

`BAR` y `FOO` se ejecutan por turnos. Pero `BAR` toma más tiempo que `FOO`. Y dado que el intervalo de muestreo es muy cercano a la frecuencia en que ambas se alternan, tenemos un error de cuantificación: la mayoría de las muestras se están tomando mientras `FOO` se está ejecutando. Así, el perfil nos dice que `FOO` usa más tiempo de CPU que `BAR`.

Hemos descrito y probado los perfiladores de subrutinas reales que han estado disponibles en UNIX durante años. En muchos casos, los vendedores tienen herramientas mucho mejores, ya sea a la venta o gratis. Si usted está haciendo un trabajo de afinación serio, pregunte a su representante de ventas qué otras herramientas le ofrece.