

# Run-time profiling with the gprof utility

*Martyn Winn*

July 2002

## Introduction

`gprof` is a commonly-available tool for profiling the execution of jobs. In particular, it gives detailed timing information which can be used for identifying bottlenecks in the program, which can then be addressed by algorithm development or parallelisation. My geek friends tell me that `gprof` is perfectly adequate for analysing serial jobs, though prettier tools exist, and different tools exist for parallel jobs.

## Availability

I am using `gprof` on my Sun under Solaris 2.8 It appears to be available under Digital UNIX V4.0F and RedHat, but I couldn't see it on our IRIX 6.5 system (Kevin says that you can run `prof -gprof` on IRIX).

In any case, a Google search revealed lots of sources for it ...

IRIX also has the tool `pixie` which "takes your executable program and creates a new version and some support files which generates profiling data. The `prof` program reads this data and produces a report." (as found by running "pixie execution" through Google ;-)

## How to use it

### Step 1:

Compile libraries and program with `-xpg` flag (or `-pg` on other compilers).

### Step 2:

Run a job as normal. You should get a file `gmon.out`. This is the "call graph profile file".

### Step 3:

Run `gprof` as:

```
gprof $CPROG/refmac5 gmon.out >! gprof.log
```

i.e. the first argument is the full pathname of the executable, and the second argument is the output file from Step 2.

### Step 4:

Look at `gprof.log`

# Output

## call graph profile

This contains listings such as:

index	%time	self	descendents	called/total	parents	
				called+self	name	index
				called/total	children	
		0.00	198.51	1/1	hkou_ [4]	
[6]	50.6	0.00	198.51	1	tls_refine_ [6]	
		0.00	158.53	20/41	refall_ [5]	
		39.80	0.01	20/20	derivs_of_tls_ [17]	
		0.00	0.13	1/1	tls_init_ [124]	
		0.04	0.00	21/42	tls2anisou_ [148]	
		0.00	0.00	20/1037	eigen_filter_r_ [271]	
		0.00	0.00	1/1	write_tls_file_ [337]	
		0.00	0.00	21/41	add_scale_contr_ [487]	
		0.00	0.00	21/62	make_u_positive_ [478]	
		0.00	0.00	1/1	calc_amat_ [549]	

This gives information for the subroutine `tls_refine`. The program spent 50.6% of its time here (I did 20 cycles of TLS and 20 cycles of restrained refinement). The routine was called from `hkou` and called `refall` etc. No significant time was spent in `tls_refine` itself, but 198.51 seconds were spent in its descendents. This time is split chiefly between `derivs_of_tls` (39.80 seconds) and the *descendents* of `refall` (158.53 seconds). The latter figure can split up further by looking under the entry for `refall`.

The number of times a routine is called is also listed. In this example, `refall` is called 20 times from `tls_refine` (corresponding to the 20 TLS cycles) and 21 times from `residual` (corresponding to the 20 restrained refinement cycles plus the final call after the last parameter shifts).

## flat profile

This lists all routines in descending order of the total time spent in the routine. For example:

%	cumulative	self		self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
24.4	97.16	97.16	41	2369.76	2547.66	prot_shrink_ [13]
10.0	136.96	39.80	20	1990.00	1990.50	derivs_of_tls_ [17]
9.3	174.12	37.16	40	929.00	1013.72	grad_aniso_ [16]

Thus, the most expensive routine is `prot_shrink` which takes up 97.16 seconds distributed over 41 calls. This represents 24.4% of the running time (the routine calculates electron density as part of the procedure for calculating the partial structure factor of the bulk solvent).

## Options

Of course, `gprof` has various options, for example for including or excluding routines from the analysis. Perhaps the most useful is the ability to add "marks" at points in the code to give a finer break down of the timings. This is described for C programs, but I couldn't see how to get it to work for Fortran.

*Last modified: Fri Jul 12 11:50:33 BST 2002*