Conventional profilers collect cumulative data over a whole execution.

No information about how performance of single portions of code scales as a function of the input size.

How can we compute efficiently the read memory size?

Two data structures:
1) shadow runtime stack, where each entry contains:
- ID of pending routine
- total routine invocation cost
- partial read memory size
- - more efficient/compact - equal to the RMS upon invocation completion
2) shadow memory

Input-Sensitive Profiling: aggregate routine times by input sizes

For routine f, collect a set of tuples, where each tuple contains:
- an estimate of an input size
- number of invocations on this input size
- max/min/avg execution cost

We need a metric for estimating the input size of a routine invocation automatically?

Profiling algorithm:

Case study: wf

We discuss wf, a simple word frequency counter included in the current development head of Fedora Linux.

Our goal: study how the performance of individual routines scales as a function of the input size. To do so, for each routine of wf, we plot a chart with k points.

We analyze wf with:

aprof

For each point of a chart we need to perform a separate run of wf.

aprof can collects several points for a chart from the same execution of a program by aggregating routine times by input sizes.

Read Memory Size: number of distinct memory cells first accessed by a routine, or by a descendent in the call tree, with a read operation.

Profiles of CPU SPEC 2006 benchmarks: examples

Input-Sensitive Profiling (or how to find the big-Oh of a program?)

Emilio Coppa, Camil Demetrescu, and Irene Finocchi

http://code.google.com/p/aprof/