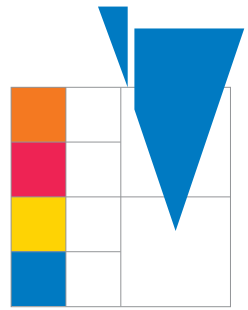
Profiling with GNU Octave

Daniel Kraft



September 21st, 2015

Why Profiling?

It is often surprising where bottlenecks in the code are.

Profiling means to measure the runtime of each part of a program.

This allows to optimise where it really helps.

Using the Profiler

Two stages for using the profiler in Octave:

Data Collection

```
profile on;
runSomeLongScript;
profile off;
lookAtTheData;
profile resume;
doSomeMoreComputations;
profile off;
```

Analysing the Profile

```
info = profile ("info");
profshow (info);
profexplore (info);
```

Cornerstones of the Implementation

Data accumulation:

- `libinterp/corefcn/profiler.[h|cc]`
- Collects data for each individual "function".
- Keeps a call stack and provides enter/exit API.

Data collection:

- Applies to function and operator classes.
- They define `class::profiler_name () const`.
- Use `BEGIN_PROFILER_BLOCK` and `END_PROFILER_BLOCK`.

Front end:

- Functions `profile`, `profshow` and `profexplore`.
- Defined as m-files in `scripts/general`.
- They use built-in functions `__profiler_*` from `profiler.cc`.