Core_bench: micro-benchmarking for OCaml

Christopher S. Hardin and Roshan P. James

Jane Street

September 24, 2013, OUD Workshop
Precise measurement is essential for writing performance sensitive code.

Objective: Measure the execution cost of functions that are relatively cheap.

- Functions with execution times on the order of nanoseconds to a tens or hundreds of milli-seconds.
- A 3.4 GHz cpu runs several simple instructions per nanosecond.
let t1 = Time.now () in f ();
let t2 = Time.now () in report (t2 - t1)

- Time.now is often too imprecise (about 1 microsec).
- Asking for current time also takes time.
let t1 = Time.now () in
f ();
let t2 = Time.now () in
report (t2 - t1)

- Time.now is often too imprecise (about 1 microsec).
- Asking for current time also takes time.
let t1 = Time.now () in
for i = 1 to batch_size do
  f ();
done;
let t2 = Time.now () in
report batch_size (t2 - t1)

- Compute a batch size to account for the timer.
- Criterion for Haskell.
- Mean, Std deviation to account for system noise.
let t1 = Time.now () in
for i = 1 to batch_size do
    f ();
done;
let t2 = Time.now () in
report batch_size (t2 - t1)

- Compute a batch size to account for the timer.
- Criterion for Haskell.
- Mean, Std deviation to account for system noise.
Micro-benchmarking: Noise

- System noise from other processes and OS activity.
- More importantly, there are delayed costs due to GC.
- Variance in execution times is influenced by batch size.
Core_bench: Linear regression

- Treats micro-benchmarking as a linear regression.
  - Simple case: fit of execution time to batch size.
- Data of larger batch sizes have smaller %-error.
  - Geometric sampling of batch sizes to get a better linear fit.
Core_bench: Linear regression

- No need to estimate the clock and other constant errors:
  - Constant overheads are accounted for in the y-intercept.

- Predict other costs in the same way.
  - Estimate memory allocations and promotions using batch size.
  - Estimate garbage collection using batch size.

- User specifies how much sampling time is allowed.
  - More data allows better estimates.
  - Error estimation, goodness of fit by
    - Bootstrapping
    - $R^2$
open Core.Std
open Core_bench.Std

let t1 = Bench.Test.create ~name:"id" (fun () -> ())

let t2 = Bench.Test.create ~name:"Time.now"
  (fun () -> ignore (Time.now ()))

let t3 = Bench.Test.create ~name:"Array.create300"
  (fun () -> ignore (Array.create ~len:300 0))

let () = Command.run (Bench.make_command [t1; t2; t3])

<table>
<thead>
<tr>
<th>Name</th>
<th>Time/Run</th>
<th>Minor</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>3.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time.now</td>
<td>843</td>
<td>2.00</td>
<td></td>
</tr>
<tr>
<td>Array.create300</td>
<td>3_971</td>
<td></td>
<td>301</td>
</tr>
</tbody>
</table>
Some functions have strange execution times

```ocaml
let benchmark = Bench.Test.create ~name:"List.init"
  (fun () -> ignore(List.init 100_000 ~f:id))
```
Multiple predictors

![Graph showing observed runtime, runs, promoted words, and compactions against batch size.](image)
Multiple predictors: fit

Using runs, compactions, promoted as predictors

![Graph showing runtime vs. batch size for observed, 1-predictor model, and 3-predictor model.]

Christopher S. Hardin and Roshan P. James

Core_bench: micro-benchmarking for OCaml
Runtime cost decomposition example

\( X = [\text{batch size } x, \text{ minor GCs, compactions}], \ y = \text{runtime (ns)}. \) Solve \( X\beta = y, \ x\gamma = X. \) Suppose we get

\[
\beta = \begin{bmatrix} 1.06 \times 10^4 \\ 1.04 \times 10^6 \\ 2.25 \times 10^6 \end{bmatrix}, \quad \gamma = \begin{bmatrix} 1 & 0.00299 & 0.00149 \end{bmatrix}
\]

Then (predicted) runtime is

\[
\gamma\beta = (1.06 \times 10^4)(1) + (1.04 \times 10^6)(0.00299) + (2.25 \times 10^6)(0.00149)
\]

\[
= 10.6\mu s + 3.1\mu s + 3.4\mu s = 17.4\mu s
\]

(Note: Just solving \( xm = y \) gives 17.4\(\mu s.\))
Conclusion and Future Work

- opam install core_bench

- Expose more predictors
  - Measure the effect of live words on performance.
  - Counters for major collection work per minor GC.

- Accuracy of results
  - Ordinary least-squares is susceptible to outliers. Incorporate the fact that measurement error is heavy-tailed (on the positive side).
  - Automatically select execution time based on error.

- Automatically pick predictors from a set.
Thank you.