microbenchmark
A package to accurately time R expressions

O. Mersmann and S. Krey
TU Dortmund University

Image courtesy of William Warby
SURGEON GENERAL'S WARNING: Microbenchmarks can lead to a distorted view of reality and massive loss of productivity.

Want to benchmark “regular” R code?
Check out the rbenchmark package by Wacek Kusnircyzk on CRAN.
So why another timing package?

Isn't this enough

```r
> system.time(function_under_test())

  user  system elapsed
12.236  0.520  13.092
```

or possibly

```r
> system.time(replicate(100, function_under_test())) / 100

  user  system elapsed
 0.820  0.145  0.965
```
> f_nothing <- function() NULL
> f_something <- function() 1 + 1
> n <- 1000000L
> (tr1 <- system.time(replicate(n, f_nothing())) / n)
user      system     elapsed
3.372e-06  5.600e-08  3.429e-06
> (tr2 <- system.time(replicate(n, f_something())) / n)
user      system     elapsed
5.488e-06  5.200e-08  5.543e-06
> (tr3 <- system.time(replicate(n, NULL)) / n)
user      system     elapsed
1.176e-06  7.600e-08  1.252e-06
> (tr4 <- system.time(replicate(n, 1 + 1)) / n)
user      system     elapsed
3.912e-06   0.000e+00  3.913e-06
> s <- seq_len(n)
> (tf1 <- system.time(for(i in s) f_nothing()) / n)
  user  system elapsed
2.88e-07 1.60e-08 3.05e-07
> (tf2 <- system.time(for(i in s) f_something()) / n)
  user  system elapsed
5.65e-07 0.00e+00 5.67e-07
> (tf3 <- system.time(for(i in s) NULL) / n)
  user  system elapsed
6.8e-08 0.0e+00 6.8e-08
> (tf4 <- system.time(for(i in s) 1 + 1) / n)
  user  system elapsed
3.20e-07 0.00e+00 3.19e-07
Which timings are “correct”?

```r
> cast(expr ~ method, data=r, value="time")
  expr replicate for
1  1 + 1 3.912e-06 3.20e-07
2  NULL 1.176e-06 6.80e-08
3  f_nothing() 3.372e-06 2.88e-07
4  f_something() 5.488e-06 5.65e-07
```

Table contains the *elapsed* time in seconds.
```r
> res <- microbenchmark(f_nothing(), f_something(), NULL, 1 + 1,
+                      times=200L)
>
> print(res, unit="s")

Unit: seconds

<table>
<thead>
<tr>
<th>expr</th>
<th>min</th>
<th>lq</th>
<th>median</th>
<th>uq</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 + 1</td>
<td>1.84e-07</td>
<td>2.10e-07</td>
<td>2.14e-07</td>
<td>2.400e-07</td>
<td>6.420e-07</td>
</tr>
<tr>
<td>NULL</td>
<td>1.50e-08</td>
<td>2.60e-08</td>
<td>2.70e-08</td>
<td>2.700e-08</td>
<td>4.600e-08</td>
</tr>
<tr>
<td>f_nothing()</td>
<td>1.80e-07</td>
<td>1.96e-07</td>
<td>2.14e-07</td>
<td>2.410e-07</td>
<td>1.077e-06</td>
</tr>
<tr>
<td>f_something()</td>
<td>3.75e-07</td>
<td>4.05e-07</td>
<td>4.32e-07</td>
<td>5.365e-07</td>
<td>6.498e-06</td>
</tr>
</tbody>
</table>
```
Let's check...
### Lets check...

```r
> cast(expr ~ method, data=rr, value="time")

<table>
<thead>
<tr>
<th>expr</th>
<th>replicate</th>
<th>for microbenchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 + 1</td>
<td>3.912e-06</td>
<td>3.20e-07</td>
</tr>
<tr>
<td>NULL</td>
<td>1.176e-06</td>
<td>6.80e-08</td>
</tr>
<tr>
<td>f_nothing()</td>
<td>3.372e-06</td>
<td>2.88e-07</td>
</tr>
<tr>
<td>f_something()</td>
<td>5.488e-06</td>
<td>5.65e-07</td>
</tr>
</tbody>
</table>
```
How it works

`proc.time()` / `system.time()`

On Unix/Linux: `getrusage()`, `gettimeofday()`, `times()`
On Windows: `GetTickCount()`, `GetProcessTimes()`
How it works

`proc.time()` / `system.time()`

On Unix/Linux: `getrusage()`, `gettimeofday()`, `times()`
On Windows: `GetTickCount()`, `GetProcessTimes()`

`microbenchmark()`

On Unix/Linux: `clock_gettime()`, `gethrtime()`
On MacOS X: `mach_timebase_info()`
On Windows: `QueryPerformanceCounter()`, `QueryPerformanceFrequency()`
Challenges

**Timing is extremely accurate**

We can measure/see

- CPU frequency scaling / throttling and core hopping
- Overhead of C function calls

**Precision of clock is unknown**

- Clock might drift
- Timing might be zero
- Might observe discrete values

**Clock only measures elapsed time**

Don't know where time is spent. Might not even be spent in R process.
Countermeasures in microbenchmark

**Automatic**

- Configurable warm-up phase to wake CPU (warmup setting in control).
- Configurable order of execution (random, inorder, block)
- Estimate minimal overhead in warm-up phase and subtract from all timings.
- Warn if timings underflow (lower than estimated overhead).
- Explicitly avoid inlining of functions in C code.

**Users responsibility**

- Ensure R session is a representative state
- Possibly call gc() or allocate some large objects
- Consider side-effects
Suggestions - Windows

Recommendations

- Bind R process to a CPU (use Task Manager)
- Set an appropriate power scheme to disable CPU frequency scaling (usually Office/Desktop)
- Install Intel/AMD CPU drivers

Windows XP

Precision of clock is usually lacking because of missing HPET.
Suggestions - MacOS X

Problems

- No clean way to temporarily disable CPU frequency scaling (must uninstall kexts!)
- No command line utility to set CPU affinity

Solution

microbenchmark tries hard to spin up CPU before actual timing. No solution for CPU hopping.
Suggestions - Linux

Recommendations

- Bind R process to a CPU (taskset)
- Disable frequency scaling:
  ```bash
  for cpu in /sys/devices/system/cpu/cpu[0-9]* ; do
    echo performance > $cpu/cpufreq/scaling_governor
  done
  
  or if you have the cpufrequtils installed
  
  cpufreq-set -c $core -g performance
  ```

Older systems

No HPET on older x86 Chipsets results in loss of precision.
More examples - `.Call`

```r
> r_do_nothing <- function(x) x
> do_nothing <- getNativeSymbolInfo("do_nothing")
>
> res <- microbenchmark(r=r_do_nothing(NULL),
+                      symbol=.Call(do_nothing, NULL),
+                      string=.Call("do_nothing", NULL),
+                      times=100L,
+                      control=list(warmup=2^16))
>
> print(res, unit="eps")

Unit: evaluations per second

<table>
<thead>
<tr>
<th>expr</th>
<th>min</th>
<th>lq</th>
<th>median</th>
<th>uq</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>775193.80</td>
<td>1509441.7</td>
<td>1564960.6</td>
<td>2998831.0</td>
<td>3597122</td>
</tr>
<tr>
<td>string</td>
<td>18906.81</td>
<td>115048.7</td>
<td>118210.3</td>
<td>120802.1</td>
<td>129584</td>
</tr>
<tr>
<td>symbol</td>
<td>81699.35</td>
<td>991080.3</td>
<td>1041124.7</td>
<td>1686340.6</td>
<td>2016129</td>
</tr>
</tbody>
</table>
```

O. Mersmann and S. Krey
More examples - side effects

```r
> x <- NULL
> res <- microbenchmark(x <- c(x, 1), times=10000L)
> print(res)
Unit: nanoseconds
  expr   min     lq median   uq  max
1 x <- c(x, 1) 477 13492  27128 40552.5 2483183
```
More examples - side effects
Conclusion and Outlook

Other real-world uses

▷ Cost of S3, S4 and proto method calls
▷ Overhead of .C vs. .Call vs pure R function
▷ Measure influence of compiler and compiler flags on speed of interpreter (Warning: microbenchmark!)

Planned features

▷ More plotting functions (currently only boxplots)
▷ Possibly use OS API to set Process Affinity
▷ Better and more diagnostic messages
▷ Estimate clock granularity and overhead more accurately