Advanced Functional Programming

Profiling in GHC

GHC Profiling

- The GHC compiler has extensive support for profiling.
- Links to documentation
  - http://www.haskell.org/ghc/docs/latest/html/users_guide/profiling.html
- GHC can perform two kinds of profiling
  - time profiling
  - space profiling

Time Profiling

- Used to instrument code to see where a program is spending its time.
- Useful to find bottlenecks
- Overview of use
  - Compile code with profiling flags
    - `ghc -prof -auto-all sort1.hs`
  - Run code with profiling command-line options
    - `./main.exe +RTS -p -RTS`
  - Inspect the profile-information file produced
    - `edit main.exe.prof`

Lets try it.

- Our goal is to write a quik-sort routine
- We expect it to have n-log(n) behavior
- We write it in Haskell (see next page)
- It appears to behave like an n^2 algorithm
- We want to know why?

First some tests

```
test1 = check "null list"
          (quik []) ([]::[Int])
test2 = check "one list"
          (quik [3]) [3]
test3 = check "1 to 10"
          (quik [1,9,2,8,3,7,4,6,5,10])
          [1,2,3,4,5,6,7,8,9,10]

tests = [test1,test2,test3]
test = runTestTT (TestList tests)
```

helper functions

```
merge [] xs = xs
merge xs [] = xs
merge (x:xs) (y:ys) | x<y = x : merge xs (y:ys)
merge (x:xs) (y:ys) = y : merge (x:xs) ys

smaller x [] = []
smaller x (y:ys) =
   if x>y
      then y : smaller x ys
      else smaller x ys

larger x [] = []
larger x (y:ys) =
   if x<=y
      then y : larger x ys
      else larger x ys
```

```
quik [] = []
quik [x] = [x]
quik (x:xs) = merge (merge small [x]) large
  where small = quik (smaller x xs)
        large = quik (larger x xs)

main =
  do { putStrLn ("N = "++show n)
     ; let l = last(quik xs)
     ; putStrLn ("The last element is: "++show l)
     }
```

Run the tests

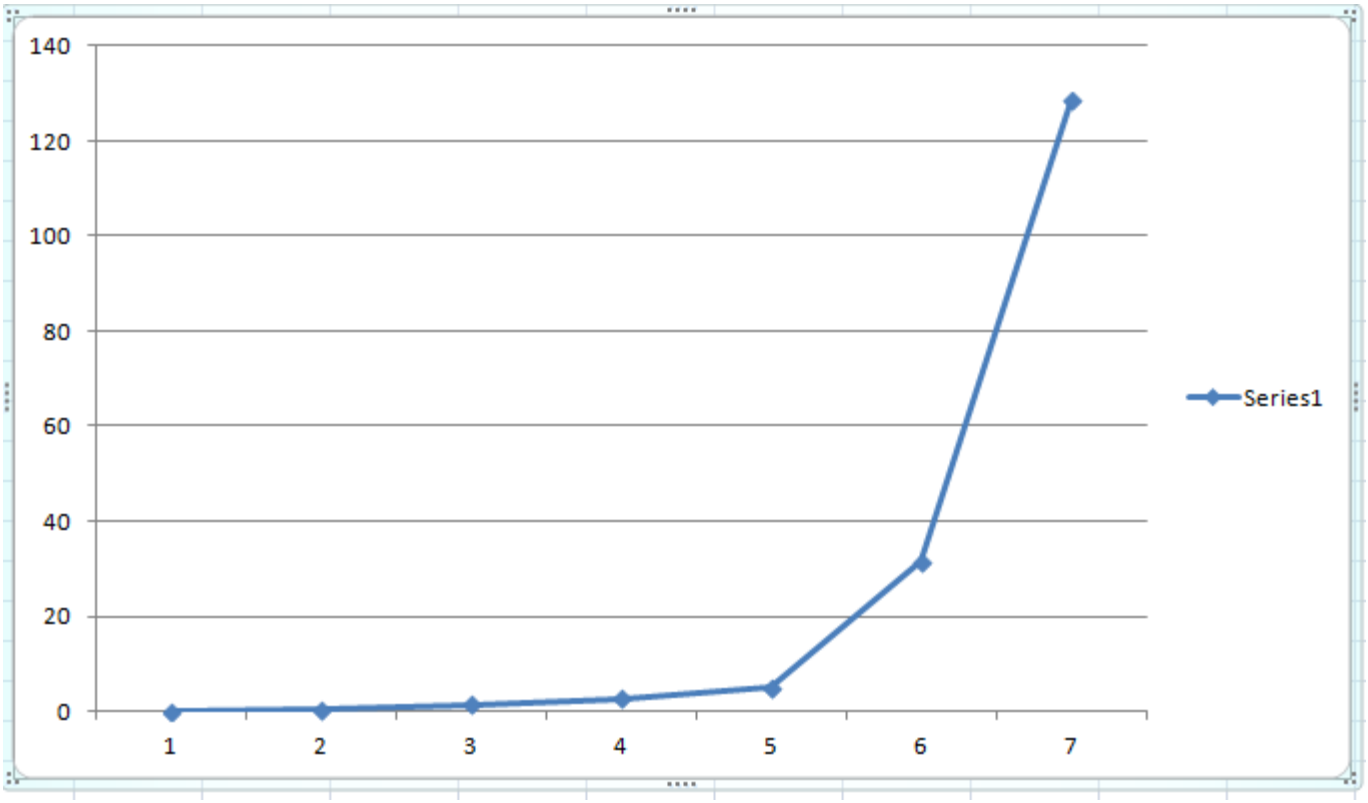
- Type :? for help
- Main> testall
- Cases: 3 Tried: 3 Errors: 0 Failures: 0
- Main>

Test time in GHCI

```
main =
  do { let l = last(quik xs)
     ; putStrLn
        ("The last element of the sort is: “
         ++show l)
     }
```

- n = 100
- xs = concat (replicate 5 [1..n])

- for n=100, main, takes 0.06 seconds
- for n= 250 about 0.30 seconds
- for n = 500 about 1.37
- for n = 750 about 2.75
- for n = 1000 about 4.95 seconds
- for n = 2500 about 31.53 seconds
- for n= 5000 about 128.4 seconds



Looks quadratic (or worse) to me.

Use GHC profiling

```
$ ghc -prof -auto-all  --make  sort0.hs
[1 of 1] Compiling Main                ( sort0.hs, sort0.o )
Linking sort0.exe ...

$ ./sort0.exe +RTS -p -RTS
N = 2500
The last element of the sort is: 2500
```

- edit sort1.prof

```
    Mon May 05 11:38 2014 Time and Allocation Profiling Report  (Final)

       sort0.exe +RTS -p -RTS

    total time  =        5.91 secs   (5911 ticks @ 1000 us, 1 processor)
    total alloc = 2,010,526,712 bytes  (excludes profiling overheads)

COST CENTRE MODULE    %time %alloc

larger      Main       60.9   99.5
smaller     Main       38.8    0.0


                                                         individual     inherited
COST CENTRE     MODULE                    no.     entries  %time %alloc   %time %alloc

MAIN            MAIN                       37           0    0.0    0.0   100.0  100.0
 CAF            GHC.IO.Encoding.CodePage   59           0    0.0    0.0     0.0    0.0
 CAF            GHC.IO.Encoding            56           0    0.0    0.0     0.0    0.0
 CAF            GHC.IO.Handle.FD           51           0    0.0    0.0     0.0    0.0
 CAF            Main                       44           0    0.0    0.0   100.0  100.0
  xs            Main                       78           1    0.0    0.0     0.0    0.0
  n             Main                       75           1    0.0    0.0     0.0    0.0
  main          Main                       74           1    0.0    0.0   100.0  100.0
   main.l       Main                       76           1    0.0    0.0   100.0  100.0
    quik        Main                       77       40001    0.1    0.1    99.9  100.0
     quik.large Main                       82       20000    0.0    0.0    60.9   99.6
      larger    Main                       83    62577496   60.9   99.5    60.9   99.5
     quik.small Main                       80       20000    0.0    0.0    38.8    0.1
      smaller   Main                       81    62577496   38.8    0.0    38.8    0.0
     merge      Main                       79      134991    0.2    0.2     0.2    0.2
```

Try for finer detail

```
merge [] xs = xs
merge xs [] = xs
merge (x:xs) (y:ys) | x<y = add1 x (merge xs (y:ys))
merge (x:xs) (y:ys) = add1 y (merge (x:xs) ys)

add1 x xs = x:xs
add2 x xs = x:xs

smaller x [] = []
smaller x (y:ys) =
   if x>y
      then add2 y (smaller x ys)
      else smaller x ys

larger x [] = []
larger x (y:ys) =
   if x<=y
      then add2 y (larger x ys)
      else larger x ys
```

```
    Mon May 05 11:42 2014 Time and Allocation Profiling Report  (Final)

       sort1.exe +RTS -p -RTS

    total time  =        1.76 secs   (1763 ticks @ 1000 us, 1 processor)
    total alloc = 505,286,712 bytes  (excludes profiling overheads)

COST CENTRE MODULE    %time %alloc

larger      Main       65.2   99.1
smaller     Main       34.2    0.1


                                                     individual     inherited
COST CENTRE     MODULE                    no.     entries  %time %alloc   %time %alloc

MAIN            MAIN                       37           0    0.1    0.0   100.0  100.0
 CAF            GHC.IO.Encoding.CodePage   59           0    0.0    0.0     0.0    0.0
 CAF            GHC.IO.Encoding            56           0    0.0    0.0     0.0    0.0
 CAF            GHC.IO.Handle.FD           51           0    0.0    0.0     0.0    0.0
 CAF            Main                       44           0    0.0    0.0    99.9  100.0
  xs            Main                       78           1    0.0    0.1     0.0    0.1
  n             Main                       75           1    0.0    0.0     0.0    0.0
  main          Main                       74           1    0.0    0.0    99.9   99.9
   main.l       Main                       76           1    0.0    0.0    99.9   99.9
    quik        Main                       77       20001    0.1    0.2    99.9   99.9
     quik.large Main                       82       10000    0.0    0.1    65.2   99.1
      larger    Main                       83    15663746   65.2   99.1    65.2   99.1
       add2     Main                       84    15643750    0.0    0.0     0.0    0.0
     quik.small Main                       80       10000    0.0    0.1    34.2    0.1
      smaller   Main                       81    15663746   34.2    0.1    34.2    0.1
       add2     Main                       85        9996    0.0    0.0     0.0    0.0
     merge      Main                       79       67491    0.5    0.5     0.5    0.5
      add1      Main                       86       47491    0.0    0.0     0.0    0.0
```

Why so many calls to add1

```
merge [] xs = xs
merge xs [] = xs
merge (x:xs) (y:ys) | x<y = add1 x (merge xs (y:ys))
merge (x:xs) (y:ys) = add1 y (merge (x:xs) ys)

quik [] = []
quik [x] = [x]
quik (x:xs) = merge (merge small [x]) large
  where small = quik (smaller x xs)
        large = quik (larger x xs)
```

Fix that

```
quik [] = []
quik [x] = [x]
quik (x:xs) = merge small (x:large)
  where small = quik (smaller x xs)
        large = quik (larger x xs)
```

```
    Mon May 05 11:45 2014 Time and Allocation Profiling Report  (Final)

       sort2.exe +RTS -p -RTS

    total time  =        1.64 secs   (1637 ticks @ 1000 us, 1 processor)
    total alloc = 503,166,952 bytes  (excludes profiling overheads)

COST CENTRE MODULE    %time %alloc

larger      Main       60.8   99.5
smaller     Main       38.9    0.1


                                                     individual     inherited
COST CENTRE     MODULE                   no.     entries  %time %alloc   %time %alloc

MAIN            MAIN                      37           0    0.0    0.0   100.0  100.0
 CAF            GHC.IO.Encoding.CodePage  59           0    0.0    0.0     0.0    0.0
 CAF            GHC.IO.Encoding           56           0    0.0    0.0     0.0    0.0
 CAF            GHC.IO.Handle.FD          51           0    0.0    0.0     0.0    0.0
 CAF            Main                      44           0    0.0    0.0   100.0  100.0
  xs            Main                      78           1    0.0    0.1     0.0    0.1
  n             Main                      75           1    0.0    0.0     0.0    0.0
  main          Main                      74           1    0.1    0.0   100.0   99.9
   main.l       Main                      76           1    0.0    0.0    99.9   99.9
    quik        Main                      77       20001    0.1    0.1    99.9   99.9
     quik.large Main                      82       10000    0.1    0.1    60.9   99.5
      larger    Main                      83    15663746   60.8   99.5    60.8   99.5
       add2     Main                      84    15643750    0.0    0.0     0.0    0.0
     quik.small Main                      80       10000    0.0    0.1    38.9    0.1
      smaller   Main                      81    15663746   38.9    0.1    38.9    0.1
       add2     Main                      85        9996    0.0    0.0     0.0    0.0
     merge      Main                      79       19996    0.0    0.1     0.0    0.1
      add1      Main                      86        9996    0.0    0.0     0.0    0.0
```

Its not the algorithm!

- It’s the data
- N=8

- [1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8]

- Well. Maybe it is the algorithm
- Choose randomly, or try another approach

Merge Sort

```
msort n [] = []
msort n [x] = [x]
msort n xs = merge2 (msort m ys) (msort m zs)
   where m = n `div` 2 - 1
         (ys,zs) = splitAt m xs
         merge2 [] xs = xs
         merge2 xs [] = xs
         merge2 (x:xs) (y:ys)
            | x<y = x : merge2 xs (y:ys)
         merge2 (x:xs) (y:ys) = y : merge2 (x:xs) ys
```

```
$ ghc -prof -auto-all  --make
sort3.hs
[1 of 1] Compiling Main
( sort3.hs, sort3.o )
Linking sort3.exe ...

sheard@freya
/cygdrive/d/work/sheard/Courses/
AdvancedFP/web/code/profiling
$ ./sort3.exe +RTS -p -RTS
N = 8
```

It doesn’t terminate, so ^C

```
    Mon May 05 11:47 2014 Time and Allocation Profiling Report  (Final)

       sort3.exe +RTS -p -RTS

    total time  =        8.29 secs   (8290 ticks @ 1000 us, 1 processor)
    total alloc = 3,515,897,488 bytes  (excludes profiling overheads)

COST CENTRE MODULE    %time %alloc

msort       Main       47.3   77.3
msort.m     Main       34.3   15.9
msort.(...) Main        6.2    6.8
msort.zs    Main        4.6    0.0
msort.ys    Main        4.5    0.0
merge       Main        2.6    0.0


                                                     individual     inherited
COST CENTRE      MODULE                    no.     entries  %time %alloc   %time %alloc

MAIN             MAIN                       37           0    0.0    0.0   100.0  100.0
 CAF             GHC.IO.Encoding.CodePage   61           0    0.0    0.0     0.0    0.0
 CAF             GHC.IO.Encoding            58           0    0.0    0.0     0.0    0.0
 CAF             GHC.IO.Handle.FD           52           0    0.5    0.0     0.5    0.0
 CAF             GHC.TopHandler             51           0    0.0    0.0     0.0    0.0
 CAF             Main                       44           0    0.0    0.0    99.5  100.0
  xs             Main                       78           1    0.0    0.0     0.0    0.0
  n              Main                       75           1    0.0    0.0     0.0    0.0
  main           Main                       74           1    0.0    0.0    99.5  100.0
   main.l        Main                       76           1    0.0    0.0    99.5  100.0
    msort        Main                       77    39952817   47.3   77.3    99.5  100.0
     msort.zs    Main                       83    19976406    4.6    0.0     4.6    0.0
     msort.m     Main                       82    19976410   34.3   15.9    34.3   15.9
     msort.(...) Main                       81    19976410    6.2    6.8     6.2    6.8
     msort.ys    Main                       80    19976410    4.5    0.0     4.5    0.0
     merge       Main                       79    19976410    2.6    0.0     2.6    0.0
```

Faulty when n==2

```
msort n xs = merge2 (msort m ys) (msort m zs)
   where m = n `div` 2 - 1
         (ys,zs) = splitAt m xs
```

Better

```
msort n xs = merge2 (msort m ys) (msort (n-m) zs)
   where m = n `div` 2
         (ys,zs) = splitAt m xs
```

```
    Mon May 05 11:26 2014 Time and Allocation Profiling Report  (Final)

       sort4.exe +RTS -p -RTS

    total time  =        0.00 secs   (2 ticks @ 1000 us, 1 processor)
    total alloc =  13,181,536 bytes  (excludes profiling overheads)

COST CENTRE  MODULE           %time %alloc

msort.(...)  Main              50.0   41.5
CAF          GHC.IO.Handle.FD  50.0    0.3
msort        Main               0.0   16.4
msort.merge2 Main               0.0   37.5
xs           Main               0.0    3.4


                                                     individual     inherited
COST CENTRE       MODULE                    no.     entries  %time %alloc   %time %alloc

MAIN              MAIN                       37           0    0.0    0.0   100.0  100.0
 CAF              GHC.IO.Encoding.CodePage   61           0    0.0    0.0     0.0    0.0
 CAF              GHC.IO.Encoding            58           0    0.0    0.0     0.0    0.0
 CAF              GHC.IO.Handle.FD           52           0   50.0    0.3    50.0    0.3
 CAF              Main                       44           0    0.0    0.0    50.0   99.7
  xs              Main                       78           1    0.0    3.4     0.0    3.4
  n               Main                       75           1    0.0    0.0     0.0    0.0
  main            Main                       74           1    0.0    0.1    50.0   96.3
   main.l         Main                       76           1    0.0    0.0    50.0   96.2
    msort         Main                       77       24999    0.0   16.4    50.0   96.2
     msort.zs     Main                       83       12499    0.0    0.0     0.0    0.0
     msort.m      Main                       82       12499    0.0    0.8     0.0    0.8
     msort.(...)  Main                       81       12499   50.0   41.5    50.0   41.5
     msort.ys     Main                       80       12499    0.0    0.0     0.0    0.0
     msort.merge2 Main                       79      115597    0.0   37.5     0.0   37.5
```

Space Profiling

- It can be quite difficult to tell the lifetime of an object in the heap.(why GC is nice)
- Lazy evaluation makes thing even more difficult because we don’t necessarily know when a thunk/closure will be evaluated.
- Solution: instrumented programs that record their own space/time behavior.

GHC profiler overview

- Compile with “–prof” to instrument the code

  (1) ghc –prof Main.hs –o Main

- Run with cues to the runtime system to generate a heap profile (*.hp)

  (2) ./Main +RTS *{options}*

- Convert the heap profile to Postscript (*.ps) and view it

  (3) hp2ps Main.hp

  (4) gv Main.ps

GHC profiler options

./Main +RTS *{options}*

- One breakdown option (see right)
- And one option to restrict the profile to a specific part of the program (see GHC User's Guide online)

Option	Breakdown
-hc	By function
-hm	By module
-hy	By type
-hb	By thunk behavior

Thunk behaviors

- Output from the –hb option

LAG	between creation and first use
USE	between first and last use
DRAG	from final use until GC’ed
VOID	never used

- Most suspicious are DRAG and VOID

Program 1

```
mean :: [Float] -> Float
mean xs = sum xs /
     (fromIntegral (length xs))

main = print (mean [0.0 .. 1000000])
```

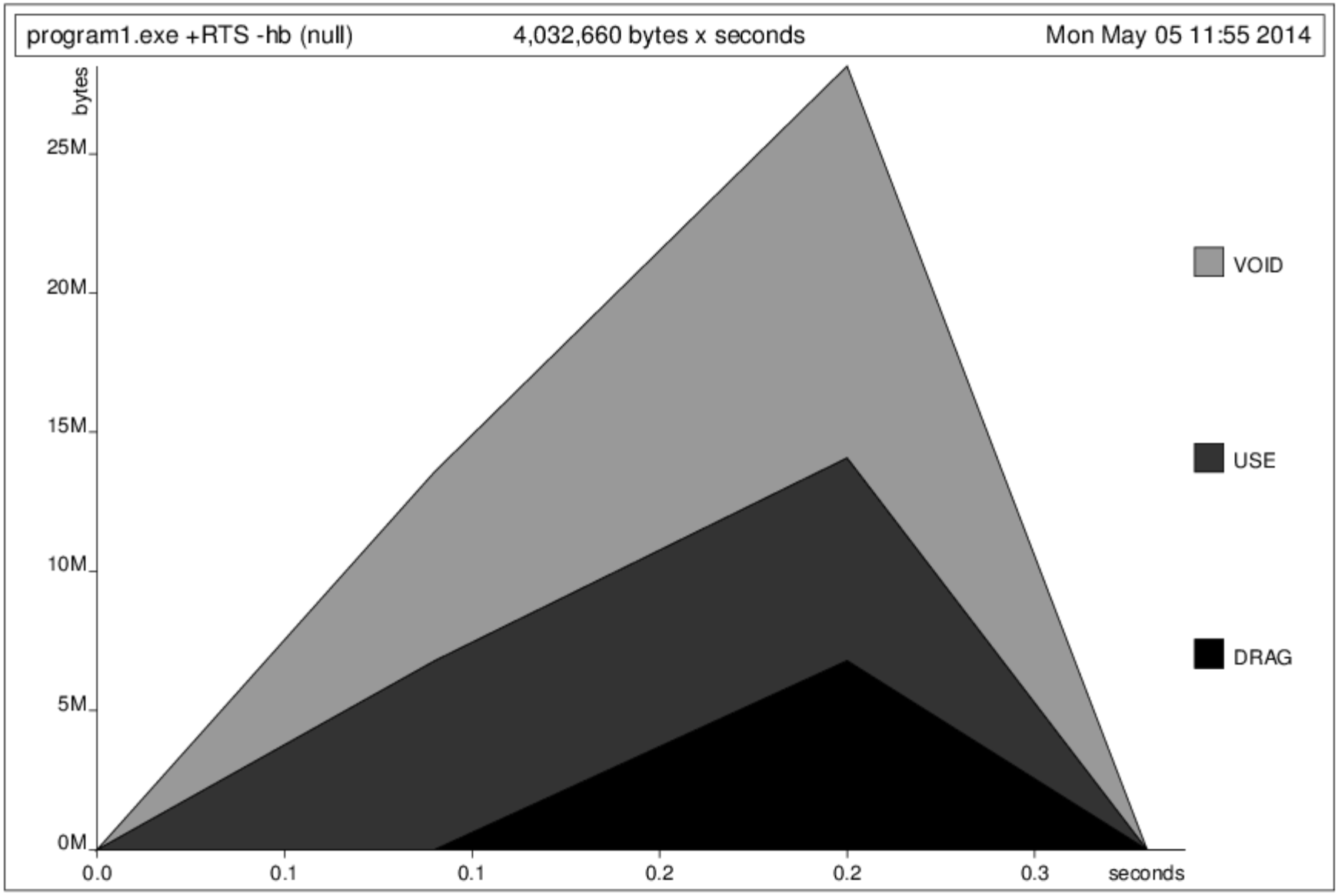
- **`xs`** is lazily computed, must be stored until both **`sum`** and **`length`** finish.
- Program runs out of memory and crashes!

```
$ ghc -prof program1.hs
```

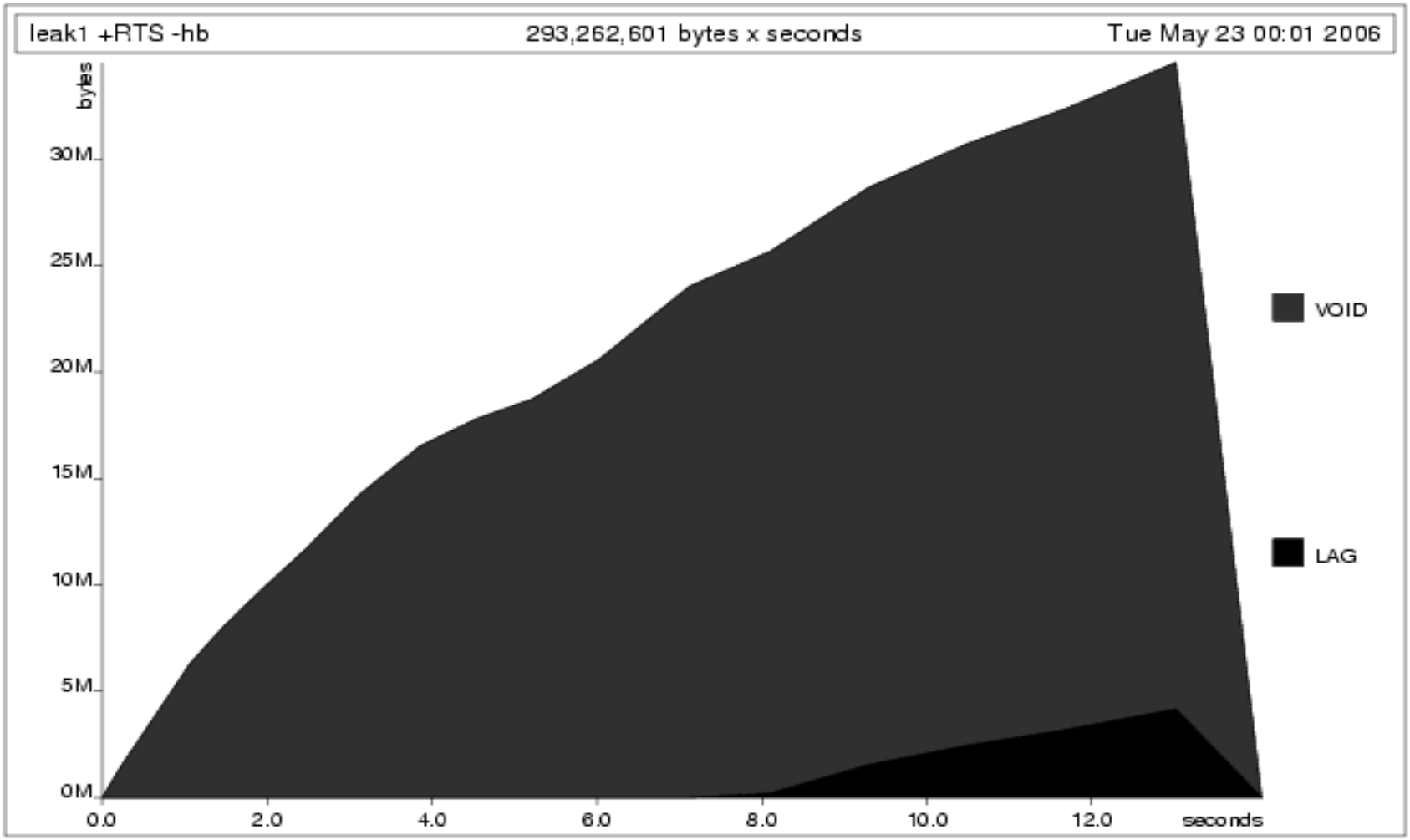
```
$ ./program1 +RTS  -hb
Stack space overflow: current size 8388608 bytes.
Use `+RTS -Ksize -RTS' to increase it.
```

```
$ hp2ps program1.hp
```

```
$ ls
program1.aux  program1.o   sort0.o     sort1.o     sort2.o     sort3.o     sort4.o
program1.exe  program1.ps  sort0.prof  sort1.prof  sort2.prof  sort3.prof  sort4.prof
program1.hi   sort0.exe    sort1.exe   sort2.exe   sort3.exe   sort4.exe   sortA.hs
program1.hp   sort0.hi     sort1.hi    sort2.hi    sort3.hi    sort4.hi    sortAA.hs
program1.hs   sort0.hs     sort1.hs    sort2.hs    sort3.hs    sort4.hs
```


program1.exe +RTS -hb (null)
4,032,660 bytes x seconds
Mon May 05 11:55 2014
bytes
25M
20M
15M
10M
5M
0M
0.0
0.1
0.1
0.2
0.2
0.3
seconds
VOID
USE
DRAG

Program 1 (by behavior)

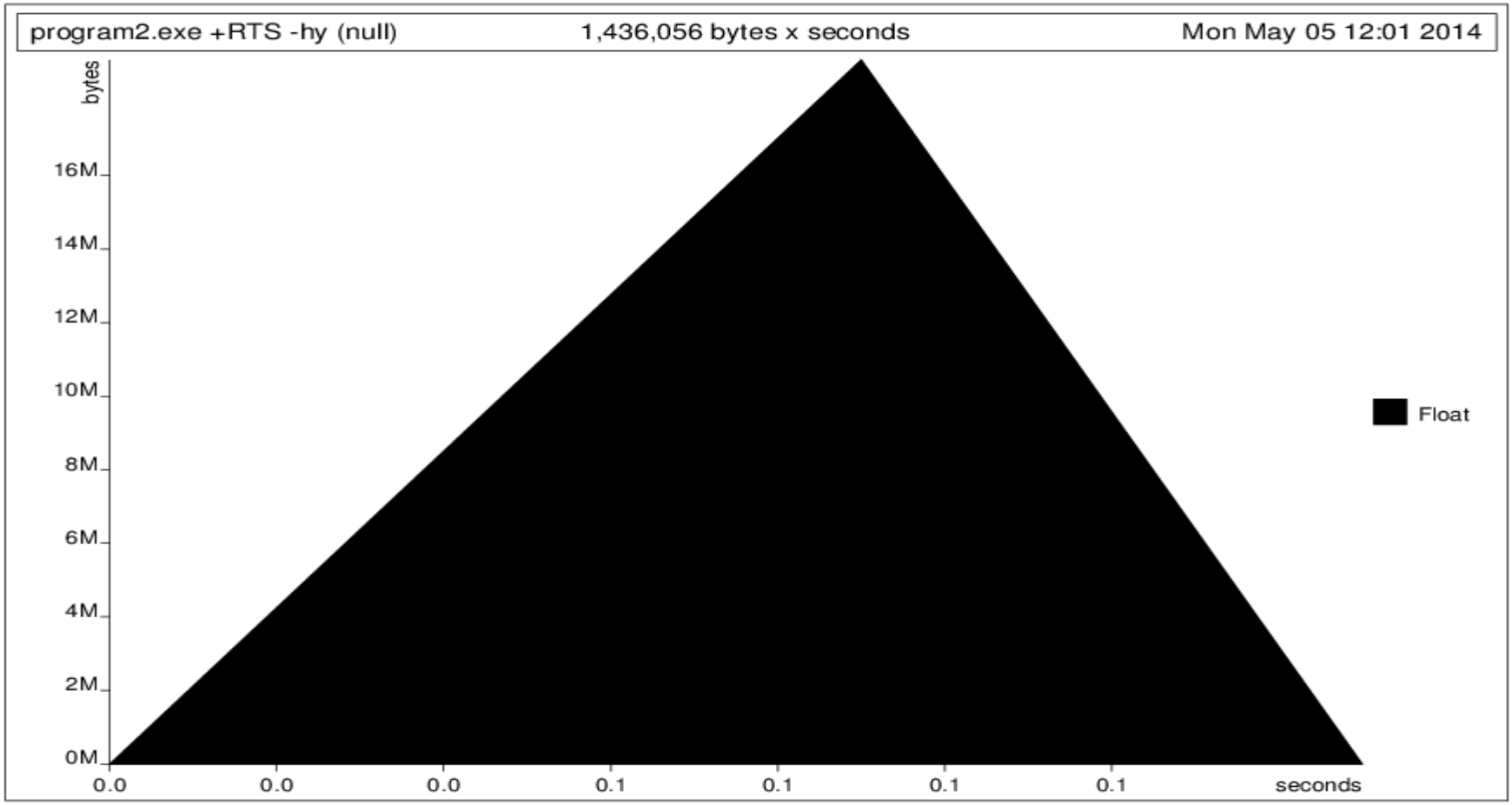


Program 2

```
mean :: [Float] -> Float
mean xs = loop 0 0 xs where
   loop sum len [] = sum / len
   loop sum len (x:xs) =
          loop (sum+x) (len+1) xs

main = print (mean [0.0 .. 1000000])
```

- Now we only traverse the list once.
- But STILL runs out of memory and crashes!

Program 2 (by type)



Strictness operators

- **`seq :: a -> b -> b  {- primitive -}`**
- Evaluating **`seq`** $\mathbf{e_1}$ $\mathbf{e_2}$ first evaluates $\mathbf{e_1}$ until its top constructor is known and then evaluates $\mathbf{e_2}$ (and returns the value of $\mathbf{e_2}$).
- **`($!) :: (a -> b) -> a -> b`**

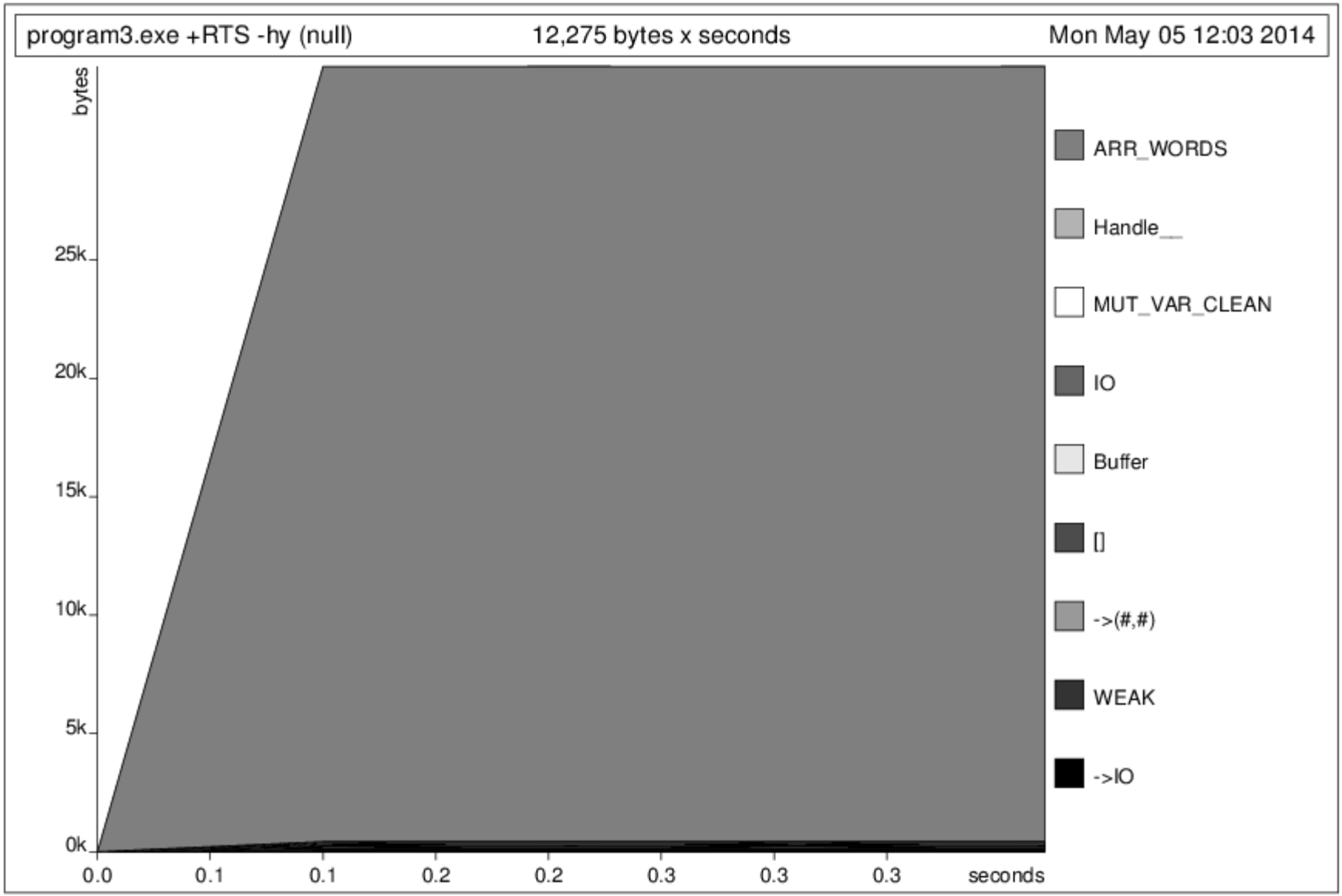
  `f $! x = seq x (f x)`
- **`($!)`** makes any function strict

Program 3

```
mean :: [Float] -> Float
mean xs = loop 0 0 xs where
   loop sum len [] = sum / len
   loop sum len (x:xs) =
      (loop $! (sum+x)) $! (len+1)) xs

main = print (mean [0.0 .. 1000000])
```

- No more senselessly growing arithmetic thunks.
- Program prints **499940.88** and exits normally

```
$ ghc -prof program3.hs
[1 of 1] Compiling Main ( program3.hs, program3.o )
Linking program3.exe ...


$ ./program3 +RTS  -hy
499940.88


$ hp2ps program3.hp
```


program3.exe +RTS -hy (null)
12,275 bytes x seconds
Mon May 05 12:03 2014
bytes
25k
20k
15k
10k
5k
0k
0.0
0.1
0.1
0.2
0.2
0.3
0.3
0.3
seconds
ARR_WORDS
Handle__
MUT_VAR_CLEAN
IO
Buffer
[]
->(#,#)
WEAK
->IO