Unnecessary allocations in maximum and mapreduce #28752

Open **legendre6891** opened this issue on 19 Aug 2018 · 1 comment

Labels iteration performance

legendre6891 commented on 19 Aug 2018

In the function `f!` below, the memory allocation scales with `n`. In comparison, `g!` allocates no memory. The two are functionally the same (I think?).

```
function f!(x,y,z)
  n = length(x)
  for i in 1:n
    z[i] = maximum(a*b - b*i for (a,b) in zip(x,y))
  end
end

function g!(x,y,z)
  n = length(x)
  for i in 1:n

    m = -Inf
    for (a,b) in zip(x,y)
      v = a * b - b * i
      if v > m
        m = v
      end
    end
    z[i] = m

  end
end
```

Here is the test

```
julia> n = 500; x = randn(n); y = randn(n); z = similar(x);

julia> @allocated f!(x,y,z)
32000
```

I posted to Slack about this and it was suggested that I post file an issue. One theory is that `zip` is allocating in every iteration. In any case, `mapreduce` exhibits the same allocation behavior:

```
julia> function f!(x,y,z)
          for (i, v) in enumerate(x)
              z[i] = mapreduce(a->(a[1]*a[2]), max, zip(x,y), init = 0.0)
          end
      end
f! (generic function with 1 method)

julia> @benchmark f!($a, $b, $c)
BenchmarkTools.Trial:
 memory estimate:  31.25 KiB
 allocs estimate:  1000
 --------------
 minimum time:     2.480 ms (0.00% GC)
 median time:      2.785 ms (0.00% GC)
 mean time:        2.922 ms (0.99% GC)
 maximum time:     39.308 ms (92.63% GC)
 --------------
 samples:          1708
 evals/sample:     1
```

```
julia> versioninfo(verbose=true)
Julia Version 1.0.0
Commit 5d4eaca0c9 (2018-08-08 20:58 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  uname: Linux 4.14.61 #2 SMP Mon Aug 6 23:42:49 CDT 2018 x86_64 Intel(R) Core(TM) i7-8700K
CPU @ 3.70GHz
  CPU: Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz:
               speed         user         nice          sys         idle          irq
       #1-12  4700 MHz     779941 s        134 s     273512 s   29072680 s          0 s

  Memory: 62.75409698486328 GB (60294.4453125 MB free)
  Uptime: 25193.0 sec
  Load Avg:  0.57763671875  0.50048828125  0.48681640625
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.0 (ORCJIT, skylake)
```

legendre6891 changed the title ~~Unnecessary allocations maximum and mapreduce~~ **Unnecessary allocations in maximum and mapreduce** on 19 Aug 2018

Some insightful experiments.

```
julia> versioninfo()
Julia Version 1.0.0
Commit 5d4eaca0c9 (2018-08-08 20:58 UTC)
Platform Info:
  OS: Windows (x86_64-w64-mingw32)
  CPU: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.0 (ORCJIT, skylake)

julia> using BenchmarkTools

julia> a = rand(1000); b = rand(1000); c = rand(1000);
```

1. Firstly with the `zip` outside.

```
julia> function f!(x,y,z)
           n = length(x)
           xy = zip(x, y)
           for i in 1:n
               z[i] = maximum(a->(a[1]*a[2] - a[2]*i), xy)
           end
       end
f! (generic function with 1 method)

julia> @benchmark f!($a, $b, $c)
BenchmarkTools.Trial:
  memory estimate:  32 bytes
  allocs estimate:  1
  --------------
  minimum time:     2.459 ms (0.00% GC)
  median time:      2.672 ms (0.00% GC)
  mean time:        2.752 ms (0.00% GC)
  maximum time:     5.569 ms (0.00% GC)
  --------------
  samples:          1815
  evals/sample:     1

julia> function g!(x,y,z)
           n = length(x)
           xy = zip(x, y)
           for i in 1:n
               z[i] = maximum((a[1]*a[2] - a[2] * i) for a in xy)
           end
       end
g! (generic function with 1 method)

julia> @benchmark g!($a, $b, $c)
BenchmarkTools.Trial:
  memory estimate:  31.28 KiB
  allocs estimate:  1001
```

```
  mean time:      2.953 ms (0.92% GC)
  maximum time:   38.495 ms (91.85% GC)
  --------------
  samples:        1701
  evals/sample:   1

julia> function h!(x,y,z)
           n = length(x)
           xy = zip(x, y)
           for i in 1:n
               z[i] = mapreduce(a->(a[1]*a[2] - a[2]*i), max, xy, init=0.0)
           end
       end
h! (generic function with 1 method)

julia> @benchmark h!($a, $b, $c)
BenchmarkTools.Trial:
  memory estimate:  32 bytes
  allocs estimate:  1
  --------------
  minimum time:     2.441 ms (0.00% GC)
  median time:      2.711 ms (0.00% GC)
  mean time:        2.900 ms (0.00% GC)
  maximum time:     7.452 ms (0.00% GC)
  --------------
  samples:          1721
  evals/sample:     1
```

As you can see, `maximum(i for i in I)` allocates in every iteration when semantically it shouldn't.

2. Further putting `zip` inside the loop causes even more allocations.

```
julia> function p!(x,y,z)
           n = length(x)
           for i in 1:n
               z[i] = mapreduce(a->(a[1]*a[2] - a[2]*i), max, zip(x, y), init=0.0)
           end
       end
p! (generic function with 1 method)

julia> @benchmark p!($a, $b, $c)
BenchmarkTools.Trial:
  memory estimate:  31.25 KiB
  allocs estimate:  1000
  --------------
  minimum time:     2.485 ms (0.00% GC)
  median time:      2.721 ms (0.00% GC)
  mean time:        2.818 ms (0.90% GC)
  maximum time:     36.950 ms (92.30% GC)
  --------------
  samples:          1773
  evals/sample:     1

julia> function m!(x,y,z)
           n = length(x)
           for i in 1:n
```

```
m! (generic function with 1 method)

julia> @benchmark m!($a, $b, $c)
BenchmarkTools.Trial:
  memory estimate:  31.25 KiB
  allocs estimate:  1000
  --------------
  minimum time:     2.421 ms (0.00% GC)
  median time:      2.669 ms (0.00% GC)
  mean time:        2.807 ms (0.96% GC)
  maximum time:     38.248 ms (93.12% GC)
  --------------
  samples:          1780
  evals/sample:     1

julia> function n!(x,y,z)
           n = length(x)
           for i in 1:n
               z[i] = maximum((a[1]*a[2] - a[2]*i) for a in zip(x, y))
           end
       end
n! (generic function with 1 method)

julia> @benchmark n!($a, $b, $c)
BenchmarkTools.Trial:
  memory estimate:  62.50 KiB
  allocs estimate:  2000
  --------------
  minimum time:     2.564 ms (0.00% GC)
  median time:      2.848 ms (0.00% GC)
  mean time:        2.951 ms (1.03% GC)
  maximum time:     41.008 ms (93.47% GC)
  --------------
  samples:          1693
  evals/sample:     1
```

2

StefanKarpinski added performance iteration labels on 21 Aug 2018

Assignees

No one assigned

Labels

iteration performance

Projects

None yet

No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

3 participants