Executive Report : Struct

Andrea Iskander Belkhir id : 511089

Beyza Özdemir id : 576145

GitHub url : https://github.com/andreaIskanderBelkhir/Struct.jl

Goal

What we did for this part of the project was the studies of tasks and an approach to the paradigms of paralelization. What we used to better understand those concept where the concepts explained in classes and the help of the recommended book **Julia High Performance** mainly the chapters 8,9 and 10.

Task

Since the base code hadn't any suggestion on what funcion where to apply some sorte of paralelization, we spent the first part on finding what funcions were candidate to be paralelized. What we choose were the funcion:

* S and T funcion.
* Embededtraversal.
* Traversal.
* Struct2lar.

Choosing the task

What we used as task in the funcions listed before were what was inside the for loop,this because every loop cicle works independently and thats a great indicator for a task. We can see easly in the code below where the task is a assign

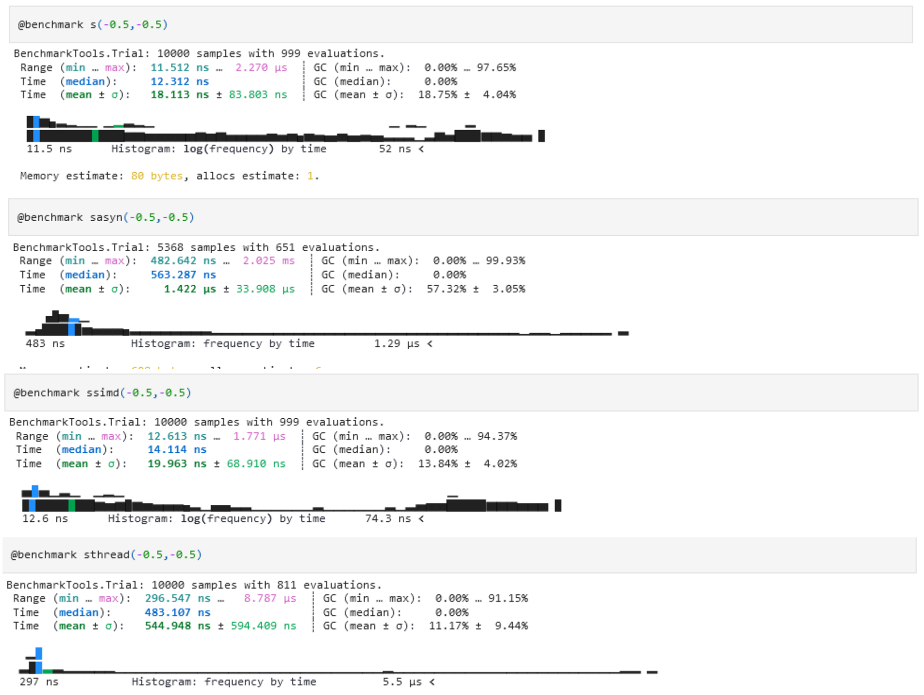
```
function  s(args...)
  d  =  length(args)
  mat  =  MMatrix{d+1,d+1,Float64}(1I)
  #create a task an run it asynchronously
  #on the thread
   @async for  k  in  range(1,  length=d)
                 mat[k,k]=args[k]
            end
  return  mat
end
```

Concurrent or Paralelism

When we worked on the tasks we used both concurrent paradigm with the `@async` annotation and paralelism with the annotations `@simd` and `Threads.@spawn` and we tested with the package Benchmarktools to find the best performance.

Testing

When we tested the funcions we saw improvmenton on three funcions but with the funcion <s(arg..)> and <t(arg..)> the paradigms returned worse performance compared to the `@inbounds`. What we understand with the test of those two funcions is that not all funcion (mostly small funcions) need a paralelism approach.



```
@benchmark s(-0.5,-0.5)
```

```
BenchmarkTools.Trial: 10000 samples with 999 evaluations.
 Range (min … max):  11.512 ns …  2.270 μs  ┊ GC (min … max):  0.00% … 97.65%
 Time  (median):     12.312 ns              ┊ GC (median):     0.00%
 Time  (mean ± σ):   18.113 ns ± 83.803 ns  ┊ GC (mean ± σ):  18.75% ±  4.04%

 11.5 ns      Histogram: log(frequency) by time       52 ns <

 Memory estimate: 80 bytes, allocs estimate: 1.
```

```
@benchmark sasyn(-0.5,-0.5)
```

```
BenchmarkTools.Trial: 5368 samples with 651 evaluations.
 Range (min … max):  482.642 ns …  2.025 ms  ┊ GC (min … max):  0.00% … 99.93%
 Time  (median):     563.287 ns              ┊ GC (median):     0.00%
 Time  (mean ± σ):     1.422 μs ± 33.908 μs  ┊ GC (mean ± σ):  57.32% ±  3.05%

 483 ns          Histogram: frequency by time        1.29 μs <
```

```
@benchmark ssimd(-0.5,-0.5)
```

```
BenchmarkTools.Trial: 10000 samples with 999 evaluations.
 Range (min … max):  12.613 ns …  1.771 μs  ┊ GC (min … max):  0.00% … 94.37%
 Time  (median):     14.114 ns              ┊ GC (median):     0.00%
 Time  (mean ± σ):   19.963 ns ± 68.910 ns  ┊ GC (mean ± σ):  13.84% ±  4.02%

 12.6 ns      Histogram: log(frequency) by time      74.3 ns <
```

```
@benchmark sthread(-0.5,-0.5)
```

```
BenchmarkTools.Trial: 10000 samples with 811 evaluations.
 Range (min … max):  296.547 ns …   8.787 μs  ┊ GC (min … max):  0.00% … 91.15%
 Time  (median):     483.107 ns               ┊ GC (median):     0.00%
 Time  (mean ± σ):   544.948 ns ± 594.409 ns  ┊ GC (mean ± σ):  11.17% ±  9.44%

 297 ns          Histogram: frequency by time          5.5 μs <
```

Next goal

What we need to work for the last submission is improving the parallesim paradigm implemented, this because what we got with the use of four thread is a worse performance then the concurrent paradigm.