Analyzing and Improving Performance issues with Go applications

My Rep and why you should read this

- wrote a programming language and made it 8x faster[1]
- go performance analysis and improvement using a leetcode example[2]
- Deep dive into Garbage collection at the examples of go and java[3]
- implemented a jit compiler in go, made the runtime 14x faster [4]
- currently writing a JSON parser not substantially slower (± 1ms) than `encoding/json`[5]

The Three Considerations

- Runtime
- Allocations
- I/O

Analyzing Applications

```go
package main;import p"runtime/pprof"
func main() {
    f, _ := os.Create("cpu.pprof")
    p.StartCPUProfile(f)
    defer p.StopCPUProfile()
}
```

General Performance Hints for Go

- always preallocate slices and maps and benchmark optimal values
- benchmark all changes and note their improvements
- use `strings.Builder`, its faster than `bytes.Buffer`
- if `strings.Builder` is too slow, buffer in your own `[]byte` and use `*(*string)(unsafe.Pointer(&buf))` or `unsafe.String(unsafe.SliceData(buf), len(buf))` this reuses the memory already stored at `[]byte`
- use `bufio.Reader` for batched I/O
- if you make a lot of long living copies, as is often the case with interpreters and parsers, either use an arena or pointers, it can help
- replace generic functions with specifically typed functions
- always search for fast paths, the goal is to always do less

[1]https://xnacly.me/posts/2023/language-performance/
[2]https://xnacly.me/posts/2023/leetcode-optimization/
[3]https://xnacly.me/papers/modern_algorithms_for_gc.pdf
[4]https://xnacly.me/papers/tree-walk-vs-go-jit.pdf
[5]https://github.com/xNaCly/libjson