Hot path optimizations for latency-critical applications in GO

Benchmarking

- Don’t trust your intuition
- The tooling does almost everything for you
- Working with assumptions is bad for you
- Reporting allocations will help you catch the big fishes

bg:50%

The charm of GO's `io` Package

- The `Reader` interface makes a good base for stream processing
- A proof is the well known `bufio` package and `json` package's `Decoder` struct

```go
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

Use your own buffer

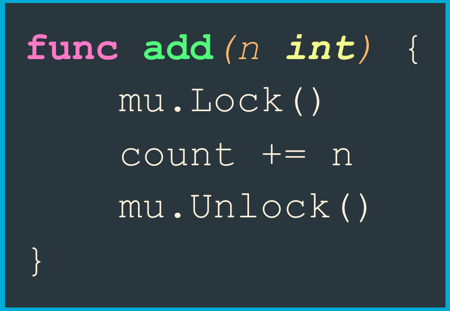
- You can use your own buffer to save allocations

```
func processLines(r io.Reader, do func([]byte)) {
    buf := make([]byte, 4096)
    for {
        n, err = r.Read(buf)
        // do something with buf...
        do(buf[:n])
    }
}
```

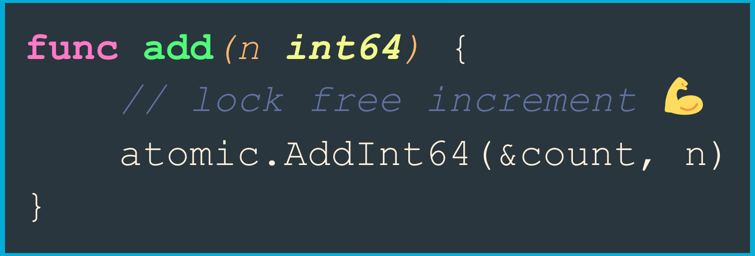
Count word occurrences in a 5M file

```
bufio.Reader    226   5250797 ns/op    4144 B/op    1 allocs/op
own_buffer      301   3972222 ns/op    0 B/op       0 allocs/op
```

Concurrency bottlenecks and resolutions



```go
func add(n int) {
    mu.Lock()
    count += n
    mu.Unlock()
}
```



```go
func add(n int64) {
    // lock free increment 💪
    atomic.AddInt64(&count, n)
}
```

```go
mu.Lock()
defer mu.Unlock()
if _, ok := check(key); ok {
    return
}
do()
set(key)
```

```go
mu.RLock()
_, ok := check(key)
mu.RUnlock()
if ok { return } // early return 💪

mu.Lock()
defer mu.Unlock()
_, ok := check(key)
if ok { return }
do()
set(key)
```

```go
res := process(<-ch)
mu.Lock()
update(res)
mu.Unlock()
```

```go
agg := make(chan *Res, 256)
agg <- process(<-ch)
for {
    var batch []*Res
    batchLoop: for {
        select {
        case res := <-agg:
            batch = append(batch, res)
        default:
            mu.Lock()
            update(batch) // batch update 💪
            mu.Unlock()
            break batchLoop
        }
    }
}
```

The difference between parallelism and concurrency

Concurrency

- The ability of a program or part of it to be executed out-of-order or in partial order, without affecting the outcome.

Parallelism (data)

- Distribution of data across different processor nodes, which operate on the data in parallel

You are not doing parallelism

- Multiple worker routines that take jobs from a channel is not necessarily parallelism
- Using a global mutex because you need to concurrently update the same object creates unpredictable scalability
- Ignoring CPU count misses the point of parallelism and creates an illusion of efficiency

bg:50%

Examples of parallelism

- Spark
- Kafka
- Partitions in databases

How can you implement parallelism

- Identify where your data can be partitioned
- Shard your tasks pseudo-randomly
- Calculate shards according to available processing cores

Recap

- Sometimes it is worth it to process by your own, optimize resource usage and earn performance improvement
- Eliminate bottlenecks
- Try to partition your processing and parallelize it

Thank You!

❤