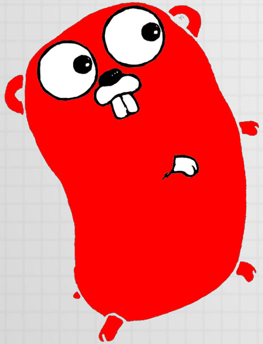
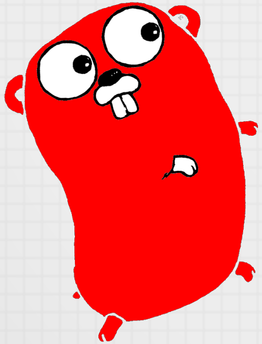
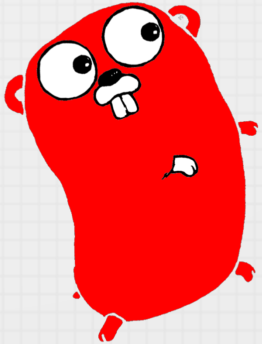
ALL ABOUT SPEED

PRESENTED BY SAMUEL PORTMANN

Parallel Computing and Dynamic Programming

PARALLEL COMPUTING Why?

Speed

Parallel computing can be used to run programs on multiple cores. This improves the performance.

World's Behavior

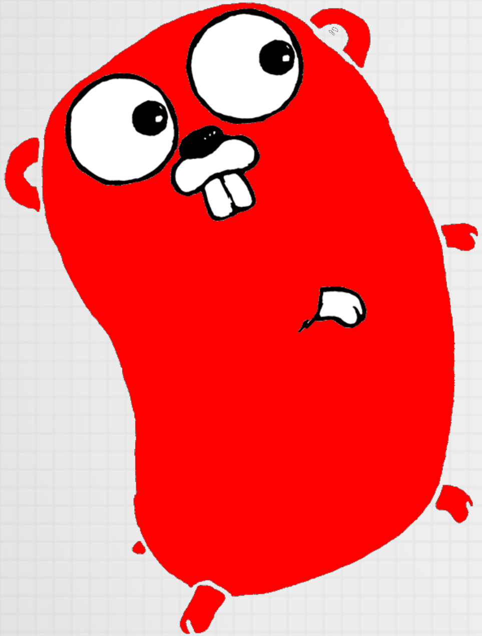
Sequential processing on its own does not model the world's behavior. Out there we see a complex world of interacting and independently behaving pieces.

Differences between Parallel Computing and Concurrent Programming:

Concurrency is not parallelism, although it enables parallelism. If you have only one processor, your program can still be concurrent but it cannot be parallel. On the other hand, a well-written concurrent program might run efficiently in parallel on a multiprocessor.

Parallel Computing is not the same as concurrent programming.

GOLANG Some Facts



Google

Go, also called golang, is a programming language initially developed at Google in 2007.

Characteristics

Go is a statically-typed language with syntax loosely derived from that of C, adding garbage collected memory management, type safety, additional built-in types such as variable-length arrays and key-value maps, and a large standard library.

Concurring Programming

Go's concurrency primitives make it easy to construct streaming data pipelines that make efficient use of I/O and multiple CPUs.

First it was a requirement to program in go. Now I am happy to have seen a new and powerful language.

GOLANG Basics I

func(parameter) return { }

1.anonymous functions - closures
2.recursive functions (multiple return)
3.go functions -> concurrency

func main()

```
func main() {
        fmt.Println("Hello World")
}
```

import "fmt"

"fmt"
"math"
"runtime"
"sync"
"time"

Data-Types

1. integer -> int
2. float -> float64
3. array -> datatype[]

I have mainly explained these elements as they are used in the show program.

GOLANG Basics II

if condition { }

```
if matrix[m][n] > (*maxValue)[m] {
        (*maxValue)[m] = matrix[m][n]
}
```

for condition { }

```
func main() {
        fmt.Println("Hello World")
}
```

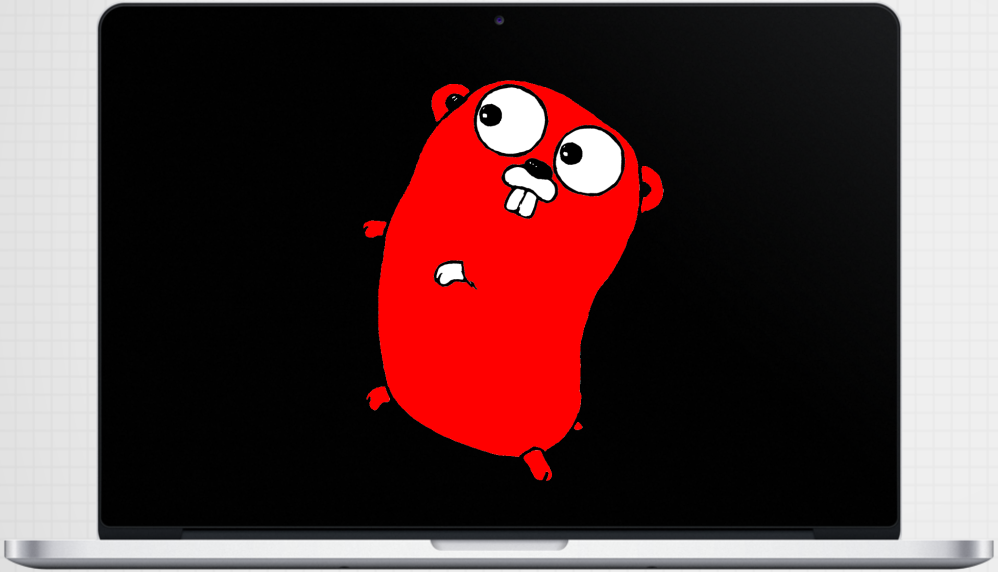
pointer

1. &variable: get the adress
2. *pointer: operate on the value

WaitGroup

1. var wg sync.WaitGroup
2. wg.Add(1)
3. wg.Done()
4. wg.Wait()

GO INTO THE CODE Speed Please



Point Of Attack

Which part of the program should we parallelize?

Code

A simple value iteration cake eating problem.

Let's go through a simple dynamic programming example.

AMDAHL'S LAW Maximum Speed Up

Formula

$$T(n) = T(1)\left(B + \frac{1}{n}(1 - B)\right)$$

$n \in \mathbb{N}$ the number of threads of execution,

$B \in [0, 1]$ the fraction of the algorithm that is strictly serial

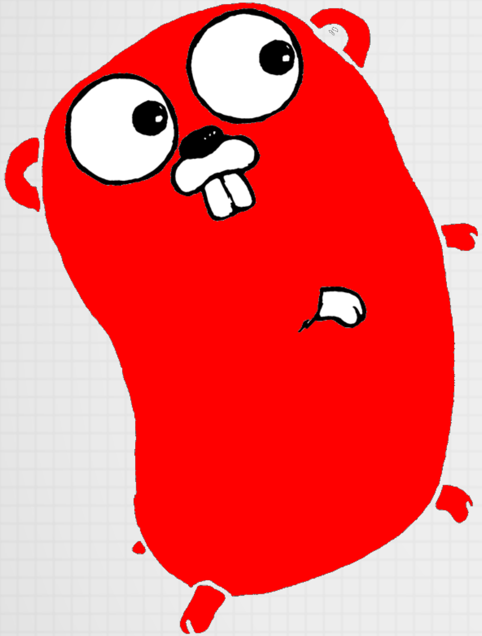
How fast can we get? (60% Parallelized)

With n = 4 and B = 40%

-> T(n) = T(1) x 0.55

We could probably parallelize more of the program and then we would be faster.

ANNOTATIONS Good To Know



Reference

I recommend the ebook available on
http://www.golang-book.com/

Matrix Lib

Skelter John: Parallelized Matrix Library
https://github.com/skelterjohn/go.matrix

GITHub

This presentation and the code:
https://github.com/portmann/go.parallelized

This is the last slide.

THANK YOU
FOR YOUR TIME