Análise de desempenho: Processamento de imagem

Artur Leal - albs
Felipe Souza - frs3
Romero Ramsés - rrcb

Centro de Informática UFPE
Universidade Federal de Pernambuco

Sumário

Introdução

Centro de Informática UFPE

- A aplicação desenvolvida consiste em selecionar uma imagem e aplicar um algoritmo para trocar os pixels coloridos por pixels preto e cinza.
- Para cada pixel no tensor que representa a imagem, substitui-se as componentes RGB por um cinza (gerado a partir das cores originais), mantendo o valor do alfa original

Introdução - Recapitulação Exercício 1

Centro de Informática UFPE

- **Versão Original**

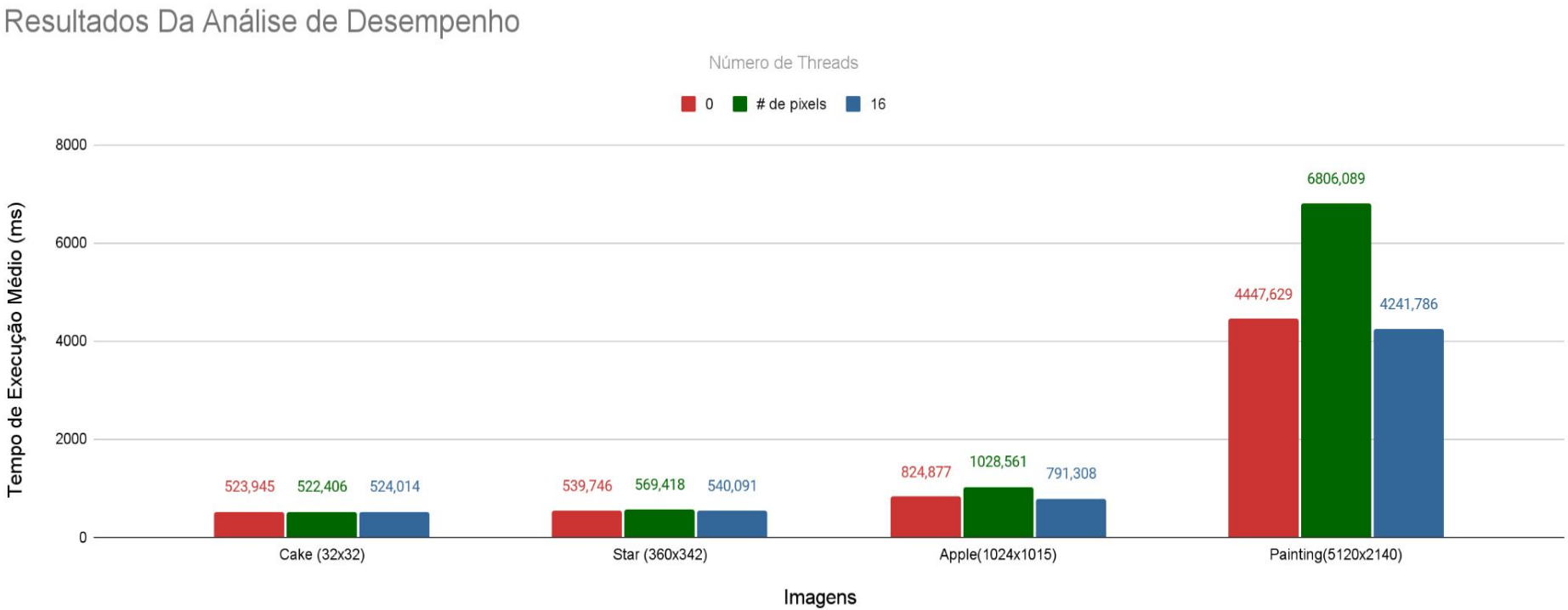
```go
func greyScaleConcV1(pixels *[][]color.Color) {
    ppixels := *pixels
    xLen := len(ppixels)
    yLen := len(ppixels[0])

    wg := sync.WaitGroup{}
    for x := 0; x < xLen; x++ {
        for y := 0; y < yLen; y++ {
            wg.Add(1)
            go func(x, y int) {
                pixel := ppixels[x][y]
                originalColor, ok := color.RGBAModel.Convert(pixel).(color.RGBA)
                if !ok {
                    fmt.Println("type conversion went wrong")
                }
                grey := uint8(float64(originalColor.R)*0.21 + float64(originalColor.G)*0.72 + float64(originalColor.B)*0.07)
                col := color.RGBA{
                    grey,
                    grey,
                    grey,
                    originalColor.A,
                }
                ppixels[x][y] = col
                wg.Done()
            }(x, y)

        }
    }
    wg.Wait()
    *pixels = ppixels
}
```

- **Versão Aprimorada**

```go
func greyScaleConcV2(pixels *[][]color.Color) {
    ppixels := *pixels
    xLen := len(ppixels)
    yLen := len(ppixels[0])

    numThreads := runtime.NumCPU()

    var wg sync.WaitGroup
    wg.Add(numThreads)

    processSection := func(startX, endX int) {
        defer wg.Done()
        for x := startX; x < endX; x++ {
            for y := 0; y < yLen; y++ {
                pixel := ppixels[x][y]
                originalColor, ok := color.RGBAModel.Convert(pixel).(color.RGBA)
                if !ok {
                    fmt.Println("type conversion went wrong")
                }
                grey := uint8(float64(originalColor.R)*0.21 + float64(originalColor.G)*0.72 + float64(originalColor.B)*0.07)
                col := color.RGBA{
                    grey,
                    grey,
                    grey,
                    originalColor.A,
                }
                ppixels[x][y] = col
            }
        }
    }

    for i := 0; i < numThreads; i++ {
        startX := (xLen * i) / numThreads
        endX := (xLen * (i + 1)) / numThreads
        go processSection(startX, endX)
    }

    wg.Wait()
}
```

Introdução - Recapitulação Exercício 1

Centro de Informática
UFPE



Introdução - Nova Proposta

Centro de Informática UFPE

- **A versão melhorada foi originalmente implementada com canais**
- **A fim de comparar especificamente o impacto do número de threads, ela foi alterada para utilizar o *sync.WaitGroup()***
- **Nesta nova análise de desempenho, será comparado, através da medição do tempo de execução das aplicações, como esses diferentes mecanismos de concorrência impactam na performance**

Introdução - Nova Proposta

Centro de Informática
UFPE

```go
func greyScaleConcWG(pixels *[][]color.Color) {
    ppixels := *pixels
    xLen := len(ppixels)
    yLen := len(ppixels[0])

    numThreads := runtime.NumCPU()

    var wg sync.WaitGroup
    wg.Add(numThreads)

    processSection := func(startX, endX int) {
        defer wg.Done()
        for x := startX; x < endX; x++ {
            for y := 0; y < yLen; y++ {
                pixel := ppixels[x][y]
                originalColor, ok := color.RGBAModel.Convert(pixel).(color.RGBA)
                if !ok {
                    fmt.Println("type conversion went wrong")
                }
                grey := uint8(float64(originalColor.R)*0.21 + float64(originalColor.G)*0.72 + float64(originalColor.B)*0.07)
                col := color.RGBA{
                    grey,
                    grey,
                    grey,
                    originalColor.A,
                }
                ppixels[x][y] = col
            }
        }
    }

    for i := 0; i < numThreads; i++ {
        startX := (xLen * i) / numThreads
        endX := (xLen * (i + 1)) / numThreads
        go processSection(startX, endX)
    }

    wg.Wait()
}
```

```go
func greyScaleConcCH(pixels *[][]color.Color) {
    ppixels := *pixels
    xLen := len(ppixels)
    yLen := len(ppixels[0])

    numThreads := runtime.NumCPU()

    ch := make(chan int, numThreads)

    processSection := func(startX, endX int) {
        for x := startX; x < endX; x++ {
            for y := 0; y < yLen; y++ {
                pixel := ppixels[x][y]
                originalColor, ok := color.RGBAModel.Convert(pixel).(color.RGBA)
                if !ok {
                    fmt.Println("type conversion went wrong")
                }
                grey := uint8(float64(originalColor.R)*0.21 + float64(originalColor.G)*0.72 + float64(originalColor.B)*0.07)
                col := color.RGBA{
                    grey,
                    grey,
                    grey,
                    originalColor.A,
                }
                ppixels[x][y] = col
            }
        }
        ch <- 1
    }

    for i := 0; i < numThreads; i++ {
        startX := (xLen * i) / numThreads
        endX := (xLen * (i + 1)) / numThreads
        go processSection(startX, endX)
    }

    for i := 0; i < numThreads; i++ {
        <-ch
    }
}
```

Introdução - Nova Proposta

Centro de Informática UFPE

- **Aprimoramento no script auxiliar de medição de tempo**

```go
func main() {
    // Path to the specific Go program
    goProgramPath := filepath.Join("greyscale", "greyscale.go")

    // Number of times to run the program
    numRuns := 110
    cpuTimes := make([]float64, numRuns-10)

    // Run the Go program and measure CPU time
    for i := 0; i < numRuns; i++ {
        startTime := time.Now()

        // Run the Go program using exec.Command
        cmd := exec.Command("go", "run", goProgramPath)
        cmd.Stdout = os.Stdout
        cmd.Stderr = os.Stderr
        cmd.Run()

        endTime := time.Now()

        // Calculate CPU time in seconds
        cpuTime := endTime.Sub(startTime).Seconds()
        // Disconsidering the first 10 results
        if i > 9 {
            cpuTimes[i-10] = cpuTime
        }
        fmt.Println("Começando Interação: ", i, cpuTime)
    }
    // Calculate the average CPU time
    averageCPUTime := calculateAverage(cpuTimes)
    // Output the results to the end of the file in the "exercicio1" subfolder
    outputFilePath := filepath.Join("greyscale", "average_cpu_time.txt")
    outputFile, err := os.OpenFile(outputFilePath, os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
    if err != nil {
        fmt.Println("Error opening output file:", err)
        return
    }
    defer outputFile.Close()

    _, err = fmt.Fprintf(outputFile, "Average CPU Time: %f seconds\n", averageCPUTime)
    if err != nil {
        fmt.Println("Error writing to output file:", err)
        return
    }

    fmt.Printf("Average CPU Time: %f seconds\n", averageCPUTime)
    fmt.Printf("Results saved to %s\n", outputFilePath)
}

// calculateAverage calculates the average of a slice of float64 values
func calculateAverage(values []float64) float64 {
    sum := 0.0
    for _, value := range values {
        sum += value
    }
    return sum / float64(len(values))
}
```

Objetivos e Métricas

Centro de Informática UFPE

- **Comparar o desempenho de duas versões do algoritmo de processamento de imagem:**
  - **Uma versão implementada com canais**
  - **Uma versão implementada com WaitGroup()**

- **Como métrica de desempenho, será utilizado o tempo de execução do programa *(em milissegundos)***

Parâmetros Utilizados

Centro de Informática UFPE

- **Parâmetros do Sistema:**
  - **CPU : AMD Ryzen 5700x (8C/16T) 3.40 GHZ**
  - **Memória: 32GB 3200Mhz DDR4**
  - **Sistema operacional: Windows 11 Home Versão 22H2**
  - **Wifi: Desligada**
  - **Configuração de Energia: Alto Desempenho *(maior estabilidade de clock)***

- **Parâmetros da Carga de Trabalho:**
  - **Mecanismo de Concorrência**
  - **Tamanho da Imagem**

Fatores Utilizados

Centro de Informática
UFPE

- **Fatores da Carga de Trabalho:**
  - **Mecanismo de Concorrência**
  - **Tamanho da Imagem**

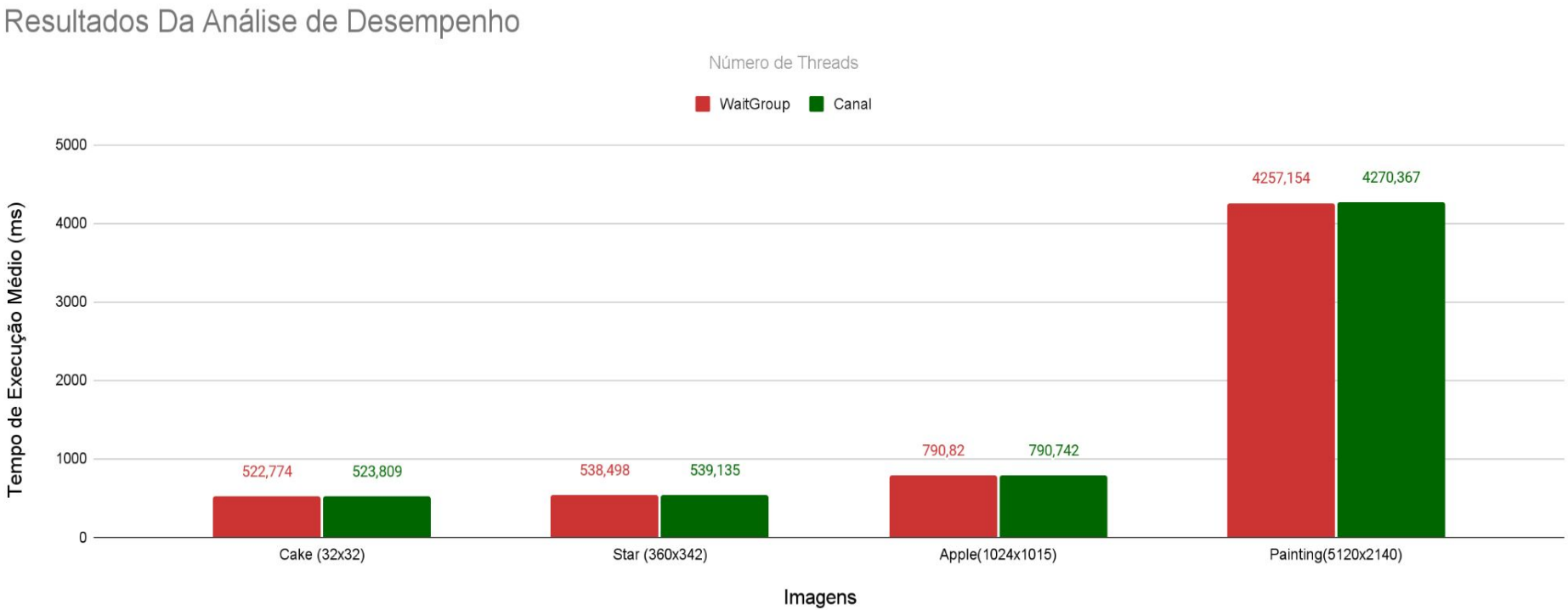
Mecanismo de Concorrência	**WaitGroup, Canal**
Tamanho da Imagem *(pixels)*	**16 x 16, 360 x 342, 1024 x 1015, 5120 x 2160**

Projetando o experimento

- **Técnica de Avaliação: Medição**
- **Um cliente executará o programa no terminal**
  - **Usando um script auxiliar, cada experimento será executado 100 vezes, sendo computado o tempo de execução de cada uma. Ao final da centésima execução, tira-se a média aritmética dos 100 resultados e armazena esse valor como o resultado do algoritmo**
  - **Um experimento consiste em esse script auxiliar com cada um dos algoritmos, de forma a ter o resultado dos três algoritmos para poder compará-los**
  - **Serão realizados 4 experimentos, cada um utilizando uma imagem diferente**
    - ***Apple.png, Cake.png, Painting.png, Star.png***

Resultados

Centro de Informática
UFPE



Resultados

- **Os resultados são praticamente os mesmos, com canais com um desempenho ligeiramente inferior**
  - **O tempo de execução é em média 0,15% superior**
  - **A razão para isso se deve por WaitGroup() ser mais leve que canal**
- **O resultado comprova que é um caso simples de paralelismo, o qual não se beneficia dos recursos extras que o canal propicia**

Obrigado!