Análise de Desempenho de Algoritmos

Antonio Neto

2024

Resumo

Este relatório tem como objetivo explorar os conceitos de análise de desempenho de algoritmos, utilizando ferramentas como *time* e *perf* em um ambiente Linux, especificamente na distribuição Ubuntu 20.04 do Docker (comandos para execução no README.md). O estudo foca na implementação de algoritmos para cálculo do fatorial de um número, utilizando métodos recursivo e iterativo, bem como na comparação de desempenho entre eles. Adicionalmente, foram analisados algoritmos de divisão e operações de escrita em arquivo. O DockerRun.txt tem os dados brutos da execução do container.

Sumário

1 Desenvolvimento

Os experimentos foram realizados em um ambiente virtualizado utilizando o Docker. Os códigos em C foram desenvolvidos para calcular o fatorial de um número de forma recursiva e iterativa, realizar operações de divisão de inteiros e floats, e operações de escrita em arquivo. Scripts em Python foram utilizados para automatizar a execução dos testes, coletar e analisar os dados, e gerar gráficos comparativos.

1.1 Implementação dos Algoritmos

Os algoritmos foram implementados conforme descrito abaixo:

1.1.1 Algoritmo Recursivo

```
#include <stdio.h>
#include <stdlib.h>

unsigned long long int factorial(unsigned long long int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <number>\n", argv[0]);
        return 1;
    }

    unsigned long long int num = atoll(argv[1]);
    unsigned long long int result = factorial(num);
    printf("Fatorial de %llu é %llu\n", num, result);

    return 0;
}
```

1.1.2 Algoritmo Iterativo

```
#include <stdio.h>
#include <stdlib.h>

unsigned long long int factorial(unsigned long long int n) {
    unsigned long long int result = 1;
    for (unsigned long long int i = 1; i <= n; ++i) {
        result *= i;
    }
    return result;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <number>\n", argv[0]);
        return 1;
    }

    unsigned long long int num = atoll(argv[1]);
    unsigned long long int result = factorial(num);
    printf("Fatorial de %llu é %llu\n", num, result);

    return 0;
}
```

1.1.3 Divisão de Inteiros

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 1000000; ++i) {
        int a = i, b = i + 1;
        int c = a / b;
        int d = b / a;
        int e = a / a;
        int f = b / b;
    }
    return 0;
}
```

1.1.4 Divisão de Floats

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 1000000; ++i) {
        float a = (float)i, b = (float)(i + 1);
        float c = a / b;
        float d = b / a;
        float e = a / a;
        float f = b / b;
    }
    return 0;
}
```

1.1.5 Operações de Escrita em Arquivo

```
#include <stdio.h>

int main() {
    FILE *file = fopen("output.txt", "w");
    if (!file) {
        perror("Erro ao abrir o arquivo");
        return 1;
    }

    for (int i = 1; i <= 1000000; ++i) {
        int a = i, b = i + 1;
        int c = a / b;
        fprintf(file, "%d\n", c);
    }

    fclose(file);
    return 0;
}
```

1.2 Execução dos Testes

Os testes foram realizados para entradas variando de 20 a 25 no caso dos algoritmos de fatorial, e para um loop de 1 a 1 milhão nos casos das divisões e operações de escrita. As ferramentas *time* e *perf* foram utilizadas para medir o tempo de execução e coletar métricas de desempenho. O script *run_tests.sh* foi utilizado para automatizar a execução dos testes e salvar os resultados.

1.3 Coleta e Análise dos Dados

Os resultados dos testes foram coletados em um arquivo JSON, que foi posteriormente analisado utilizando um script Python. Este script utilizou a biblioteca *matplotlib* para gerar gráficos comparativos dos tempos de execução e outras métricas de desempenho.

1.3.1 Script Python

```
import json
import matplotlib.pyplot as plt

# Load the updated JSON data
with open('DockerRun.json') as f:
    data = json.load(f)

# Extract data
executions = data['executions']
metrics = ['task_clock', 'cycles', 'stalled_cycles_frontend', 'instructions', 'branch

# Organize data by metric
metrics_data = {metric: [] for metric in metrics}
names = []
real_times = []
user_times = []
sys_times = []

for execution in executions:
    names.append(f"{execution['name']} {execution.get('input', '')}".strip())
    real_time_str = execution['real_time']
    user_time_str = execution['user_time']
    sys_time_str = execution['sys_time']
    real_times.append(float(real_time_str.split()[0]))
    user_times.append(float(user_time_str.split()[0]))
    sys_times.append(float(sys_time_str.split()[0]))
    for metric in metrics:
        value_str = execution['perf_output'][metric]
        value = float(value_str.split()[0].replace(',', ''))
        metrics_data[metric].append(value)

# Translate metric names
metric_translations = {
    'task_clock': 'Tempo de Tarefa',
    'cycles': 'Ciclos',
    'stalled_cycles_frontend': 'Ciclos Parados no Frontend',
    'instructions': 'Instruções',
    'branches': 'Ramificações',
    'branch_misses': 'Falhas de Ramificação',
    'l1_dcache_loads': 'Carregamentos L1 DCache',
    'l1_dcache_load_misses': 'Falhas de Carregamento L1 DCache'
}

# Separate factorial executions and others
factorial_indices = [i for i, name in enumerate(names) if 'fatorial' in name]
other_indices = [i for i, name in enumerate(names) if 'fatorial' not in name]
```

```
factorial_names = [names[i] for i in factorial_indices]
factorial_real_times = [real_times[i] for i in factorial_indices]
factorial_user_times = [user_times[i] for i in factorial_indices]
factorial_sys_times = [sys_times[i] for i in factorial_indices]

other_names = [names[i] for i in other_indices]
other_real_times = [real_times[i] for i in other_indices]
other_user_times = [user_times[i] for i in other_indices]
other_sys_times = [sys_times[i] for i in other_indices]

# Plot execution times for factorial functions
plt.figure(figsize=(10, 5))
plt.barh(factorial_names, factorial_real_times, color='lightcoral', label='Tempo Real
plt.barh(factorial_names, factorial_user_times, color='skyblue', label='Tempo de Usuá
plt.barh(factorial_names, factorial_sys_times, color='lightgreen', label='Tempo de Si
plt.xlabel('Tempo (segundos)')
plt.title('Comparação dos Tempos de Execução para Fatorial')
plt.legend()
plt.tight_layout()
plt.savefig('resultados/tempo_execucao_fatorial.png')
plt.close()

# Plot execution times for other functions
plt.figure(figsize=(10, 5))
plt.barh(other_names, other_real_times, color='lightcoral', label='Tempo Real')
plt.barh(other_names, other_user_times, color='skyblue', label='Tempo de Usuário', le
plt.barh(other_names, other_sys_times, color='lightgreen', label='Tempo de Sistema',
plt.xlabel('Tempo (segundos)')
plt.title('Comparação dos Tempos de Execução para Outras Funções')
plt.legend()
plt.tight_layout()
plt.savefig('resultados/tempo_execucao_outras.png')
plt.close()

# Plot each metric
for metric, values in metrics_data.items():
    plt.figure(figsize=(10, 5))
    plt.barh(names, values, color='skyblue')
    plt.xlabel(metric_translations[metric])
    plt.title(f'Comparação de {metric_translations[metric]} entre Execuções')
    plt.tight_layout()
    plt.savefig(f'resultados/{metric}.png')
    plt.close()
```

2 Resultados e Discussão

Os gráficos a seguir apresentam os tempos de execução dos algoritmos recursivo e iterativo para o cálculo do fatorial, bem como métricas de desempenho coletadas com a ferramenta *perf*.

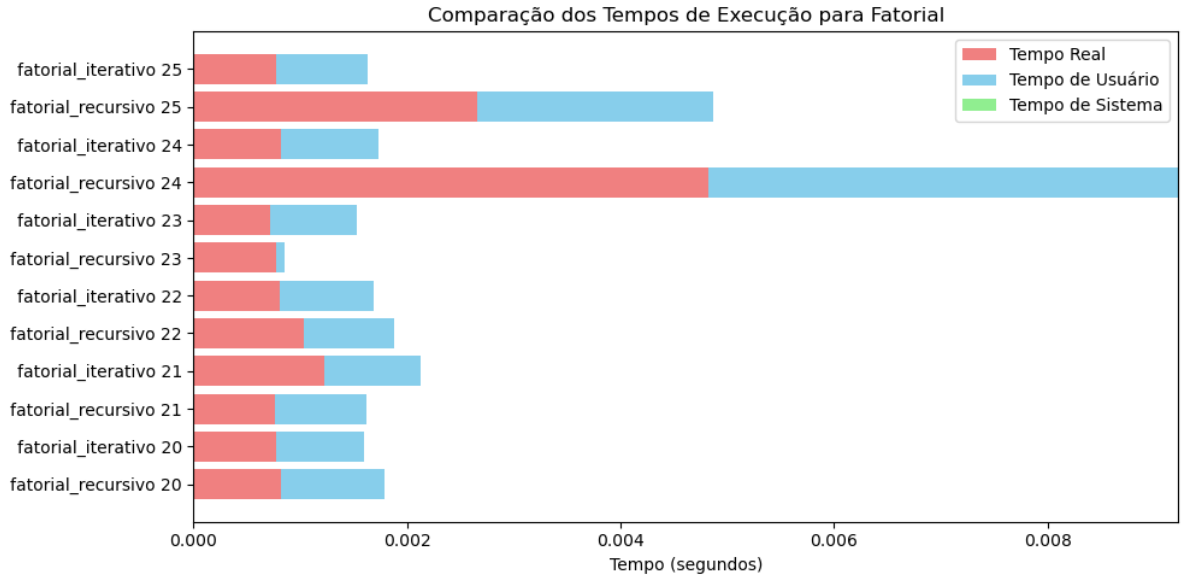

Figura 1: Tempo de Execução dos Algoritmos de Fatorial

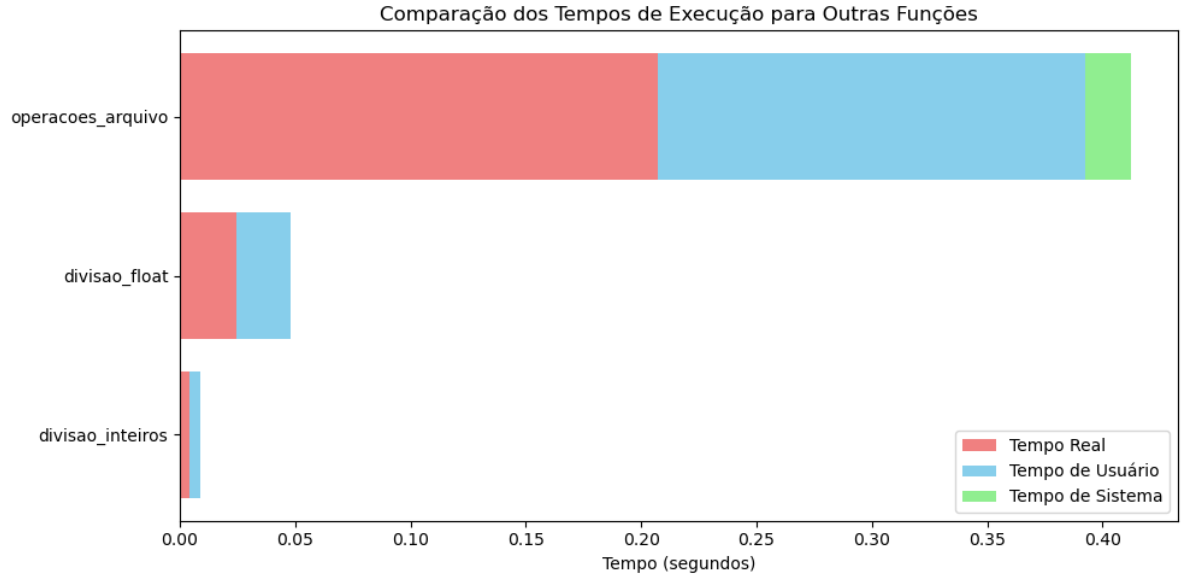

Figura 2: Tempo de Execução de Outras Funções

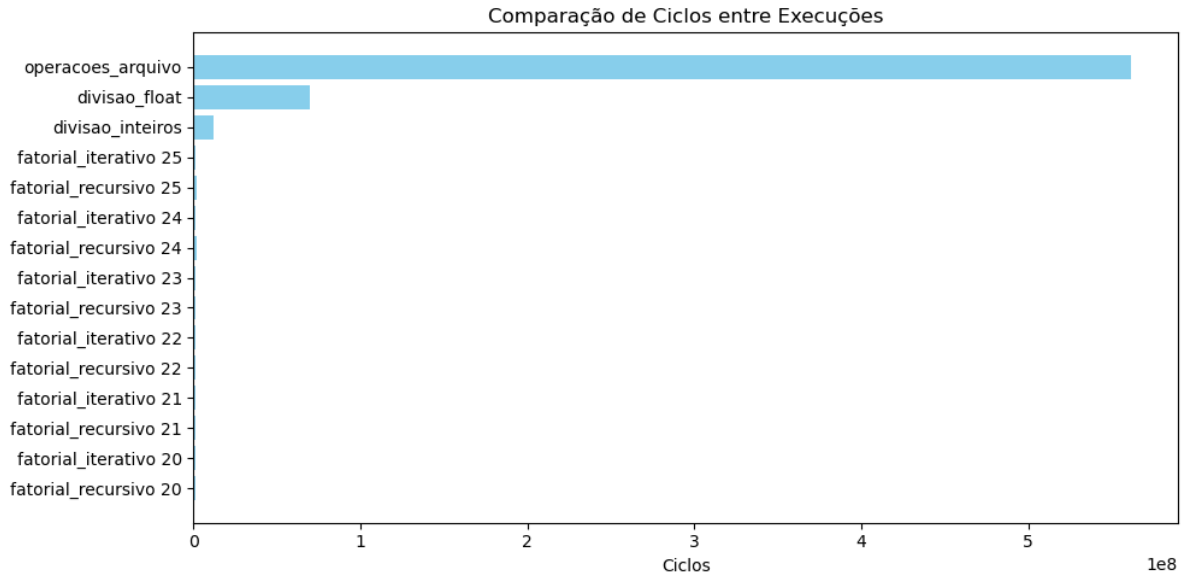

Figura 3: Ciclos de CPU dos Algoritmos

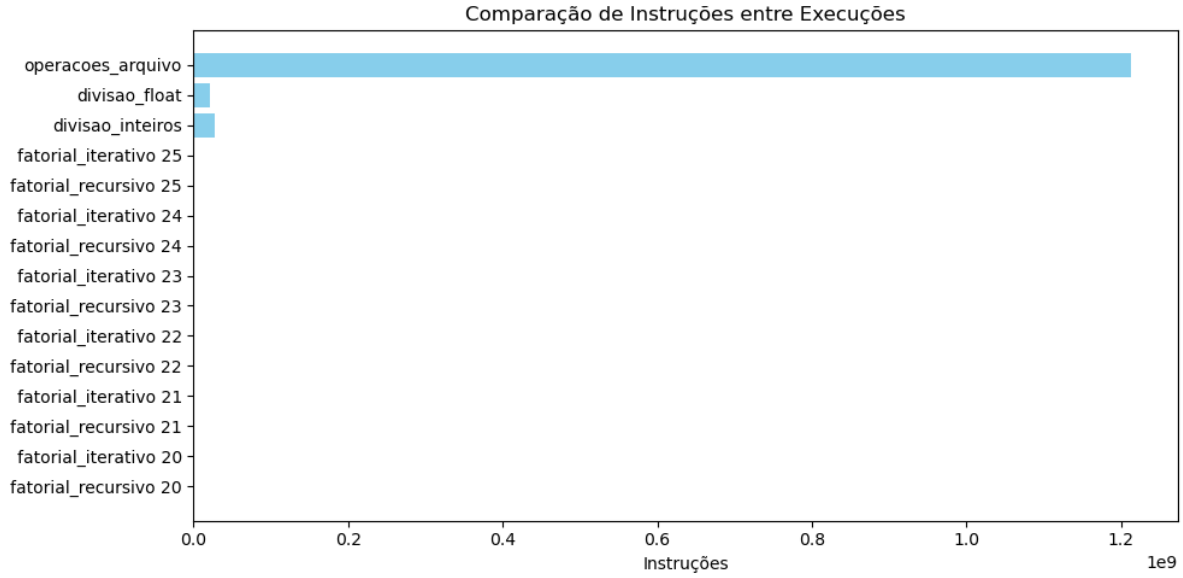

Figura 4: Instruções de CPU dos Algoritmos

Os resultados mostram que o algoritmo iterativo tem um desempenho melhor em termos de tempo de execução e ciclos de CPU quando comparado ao algoritmo recursivo. As operações de divisão de inteiros e floats apresentaram desempenho similar, enquanto a operação de escrita em arquivo adicionou um overhead significativo devido ao I/O. A análise detalhada das métricas forneceu uma visão abrangente do comportamento dos algoritmos.

3 Análise das Métricas

Nesta seção, apresentamos as métricas de desempenho detalhadas para diferentes algoritmos e entradas. Cada métrica é explicada abaixo com suas respectivas fórmulas:

- **Percentual do Tempo Total Gasto pela CPU**: Representa a porcentagem do tempo total de execução que a CPU utilizou para processar o algoritmo. É calculado como:

$$\text{Percentual do Tempo CPU} = \frac{\text{Tempo de Usuário} + \text{Tempo de Sistema}}{\text{Tempo Real}} \tag{1}$$

- **MIPS (Milhões de Instruções por Segundo)**: Mede o número de instruções executadas pela CPU por segundo. É calculado como:

$$\text{MIPS} = \frac{\text{Número de Instruções}}{\text{Tempo de CPU} \times 10^6} \tag{2}$$

- **MFLOPS (Milhões de Operações de Ponto Flutuante por Segundo)**: Mede o número de operações de ponto flutuante executadas pela CPU por segundo. É calculado como:

$$\text{MFLOPS} = \frac{\text{Número de Operações de Ponto Flutuante}}{\text{Tempo de CPU} \times 10^6} \tag{3}$$

- **Tempo de CPU**: Tempo total de uso da CPU para executar o algoritmo, calculado com base no número de ciclos de clock e a frequência do clock:

$$\text{Tempo de CPU} = \frac{\text{Número de Ciclos de Clock}}{\text{Frequência do Clock}} \tag{4}$$

- **Performance**: Uma medida geral de desempenho, calculada como o inverso do tempo de execução:

$$\text{Performance} = \frac{1}{\text{Tempo de Execução}} \tag{5}$$

3.1 Métricas Detalhadas

- **Métricas para fatorial_recursivo 20**:
  - Percentual do tempo total gasto pela CPU: 1.18
  - MIPS: 682.60
  - MFLOPS: 1031.99
  - Tempo de CPU: 0.000541 segundos
  - Performance: 1214.742110
- **Métricas para fatorial_iterativo 20**:
  - Percentual do tempo total gasto pela CPU: 1.05
  - MIPS: 833.61

  - MFLOPS: 1222.49
  - Tempo de CPU: 0.000529 segundos
  - Performance: 1284.985332

- **Métricas para fatorial_recursivo 21**:
  - Percentual do tempo total gasto pela CPU: 1.11
  - MIPS: 802.22
  - MFLOPS: 1177.86
  - Tempo de CPU: 0.000531 segundos
  - Performance: 1305.280250

- **Métricas para fatorial_iterativo 21**:
  - Percentual do tempo total gasto pela CPU: 0.74
  - MIPS: 763.74
  - MFLOPS: 1108.65
  - Tempo de CPU: 0.000547 segundos
  - Performance: 819.785692

- **Métricas para fatorial_recursivo 22**:
  - Percentual do tempo total gasto pela CPU: 0.82
  - MIPS: 777.99
  - MFLOPS: 1177.86
  - Tempo de CPU: 0.000519 segundos
  - Performance: 966.626262

- **Métricas para fatorial_iterativo 22**:
  - Percentual do tempo total gasto pela CPU: 1.10
  - MIPS: 768.72
  - MFLOPS: 1129.94
  - Tempo de CPU: 0.000541 segundos
  - Performance: 1243.903319

- **Métricas para fatorial_recursivo 23**:
  - Percentual do tempo total gasto pela CPU: 0.11
  - MIPS: 7792.54
  - MFLOPS: 11904.76
  - Tempo de CPU: 0.000503 segundos
  - Performance: 1299.513072

- **Métricas para fatorial_iterativo 23**:

  - Percentual do tempo total gasto pela CPU: 1.12
  - MIPS: 821.25
  - MFLOPS: 1236.09
  - Tempo de CPU: 0.000499 segundos
  - Performance: 1387.695856
- **Métricas para fatorial_recursivo 24**:
  - Percentual do tempo total gasto pela CPU: 0.91
  - MIPS: 152.17
  - MFLOPS: 227.38
  - Tempo de CPU: 0.000659 segundos
  - Performance: 207.334585
- **Métricas para fatorial_iterativo 24**:
  - Percentual do tempo total gasto pela CPU: 1.11
  - MIPS: 721.11
  - MFLOPS: 1095.29
  - Tempo de CPU: 0.000542 segundos
  - Performance: 1219.034988
- **Métricas para fatorial_recursivo 25**:
  - Percentual do tempo total gasto pela CPU: 0.83
  - MIPS: 308.46
  - MFLOPS: 451.47
  - Tempo de CPU: 0.000638 segundos
  - Performance: 376.609961
- **Métricas para fatorial_iterativo 25**:
  - Percentual do tempo total gasto pela CPU: 1.10
  - MIPS: 786.03
  - MFLOPS: 1176.47
  - Tempo de CPU: 0.000530 segundos
  - Performance: 1289.292810
- **Métricas para divisao_inteiros**:
  - Percentual do tempo total gasto pela CPU: 1.01
  - MIPS: 6042.20
  - MFLOPS: 227.38
  - Tempo de CPU: 0.004896 segundos

  - Performance: 230.621509

- **Métricas para divisao_float**:
  - Percentual do tempo total gasto pela CPU: 0.98
  - MIPS: 906.79
  - MFLOPS: 42.08
  - Tempo de CPU: 0.027868 segundos
  - Performance: 41.145299

- **Métricas para operacoes_arquivo**:
  - Percentual do tempo total gasto pela CPU: 0.99
  - MIPS: 5911.73
  - MFLOPS: 4.87
  - Tempo de CPU: 0.224655 segundos
  - Performance: 4.832881

4 Conclusão

A análise de desempenho realizada revelou que, para o cálculo do fatorial de números relativamente grandes (20 a 25), o método iterativo é mais eficiente em termos de tempo de execução e uso de ciclos de CPU. As operações de divisão mostraram-se eficientes, enquanto as operações de escrita em arquivo apresentaram maior overhead devido ao I/O. A ferramenta *perf* foi essencial para coletar métricas detalhadas de desempenho, possibilitando uma análise mais aprofundada.