Universidade Católica de Brasília

TRABALHO FINAL

PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

DETALHAMENTO DO SPEEDUP

Thiago Castro Gonçalves (UC23101157)

Gustavo Nunes Silva (UC23200427)

Moises M X S de Paula (UC23200597)

Renan Augusto Barbosa Inacio da Silva (UC23201116)

Rafael Rodrigues Martins (UC22201129)

Professor: Marcelo Eustáquio

Universidade Católica de Brasília

1. Introdução aos Códigos

Os códigos desenvolvidos buscam organizar e manipular dados, de forma a permitir uma análise rebuscada por meio de gráficos. Há dois algoritmos que desempenham a mesma função, porém um é tratado de forma sequencial, enquanto outro é abordado de forma paralelizada, o que acarreta diferença temporal em suas execuções.

1.1. Código Não Paralelizado

Percorre arquivos CSV de uma pasta chamada Dados, agrupando e processando os dados por ramo da Justiça (como Justiça Federal, do Trabalho, Estadual e Tribunais Superiores). Para cada grupo, aplica funções específicas que calculam metas de desempenho, padronizando os resultados mesmo quando dados estiverem ausentes. Ao final, gera dois DataFrames: um com todos os dados combinados (df_consolidado) e outro com o resumo das metas por tribunal (resumo_metas). Esses resultados são salvos em arquivos CSV e utilizados para gerar gráficos. Todo o processamento é feito de forma sequencial e o tempo de execução é medido e exibido.

A seguir, visualiza-se o tempo de execução do código :

```
Tempo lendo arquivos e calculando metas (n�o paralelo): 32.60123 segundos
Tempo consolidando DataFrame e criando resumo: 2.26127 segundos
Tempo criando Consolidado_NP.csv: 295.20680 segundos
Tempo criando ResumoMetas_NP.csv: 0.01110 segundos
Tempo total de execu��o (Vers�o N�o Paralela): 544.59123 segundos

[Done] exited with code=0 in 547.249 seconds
```

1.2. Código Paralelizado

Este código processa dados de arquivos CSV localizados na pasta Dados, calculando metas de desempenho para diferentes ramos da Justiça (como Federal, Estadual, STJ, TST, etc.), de forma **paralela** usando ThreadPoolExecutor.

A função processar_arquivo_csv é responsável por ler um arquivo, identificar os ramos da Justiça presentes e aplicar a função de cálculo adequada para cada um, gerando uma linha de resumo com os resultados das metas.

A função gerar_metas_paralelizado coordena esse processamento paralelo: ela envia cada arquivo para uma thread separada, coleta os resultados e, ao final, consolida tudo em dois DataFrames : df_consolidado e resumo_metas. Esses dados são exportados como CSV e usados para gerar gráficos. A execução paralela permite maior desempenho, especialmente quando há muitos arquivos para processar. O tempo de execução total é exibido ao final para monitoramento do desempenho.

A seguir, visualiza-se o tempo de execução do código :

```
Tempo lendo arquivos e calculando metas (paralelo): 27.68131 segundos
Tempo consolidando DataFrame e criando resumo: 2.12969 segundos
Tempo criando Consolidado_P.csv: 227.22518 segundos
Tempo criando ResumoMetas_P.csv: 0.00815 segundos
Tempo total de execu��o (Vers�o Paralela): 327.79670 segundos

[Done] exited with code=0 in 331.148 seconds
```

2. Detalhamento do SpeedUp

O SpeedUp é uma métrica que permite avaliar o ganho de desempenho de um processo, à medida em que ele é executado de forma sequencial e paralela.

É possível calcular o SpeedUp com a fórmula :

$$S(n) = \frac{T(1)}{T(n)}$$

- S(n) = speedup com n processadores.
- T(1) = tempo de execução com um processador (sequencial).
- T(n) = tempo de execução com n processadores (paralelo).

Sendo assim:

S(n) = T(1) / T(n) → S(n)=544,59 / 327,79 → S(n)=1,66

Interpretação: Se S(n)>1, significa que o código que rodou em paralelo é mais rápido que o código que rodou sequencialmente. Além disso, o SpeedUp gerado é classificado com **sublinear,** visto que não gera um aumento proporcional ao aumento de processadores, porém acarreta notória melhora de performance. Isso decorre, sobretudo, em virtude de partes não paralelizáveis do algoritmo.

2.1. Lei de Amdahl

Essa lei estima o ganho máximo possível ao paralelizar um código com n núcleos disponíveis. Sua fórmula é dada por :

$$S = \frac{1}{\mathrm{B} + \frac{(1 - \mathrm{B})}{n}}$$

- **B :** Fração do programa que pode ser paralelizada
- **N :** número de threads ou núcleos usados

Conclusão : Assumindo 0.9 para B e 8 para N, no caso do código paralelizado, é possível inferir que o ganho máximo desse código, em relação ao sequencial, seria de aproximadamente 4,7x mais velocidade.