

# **Relatório de Avaliação Substitutiva - Programação Concorrente e Distribuída Comparação Experimental de Performance entre Diferentes Configurações de Threads**

**Autor: Lucas Assis Pereira (UC21104934)**

**Instituição: Universidade Católica de Brasília**

**Data: 30/06/2024**

## Sumário

1	Introdução .....	3
2	O que são Threads .....	3
3	Como Threads Funcionam Computacionalmente .....	3
3.1	Componentes Essenciais de uma Thread .....	3
3.2	Gerenciamento de Threads .....	3
3.3	Mecanismos de Sincronização .....	4
4	Como o Uso de Threads Pode Afetar o Tempo de Execução de um Algoritmo.....	4
4.1	Execução Paralela .....	4
4.2	Overhead.....	4
4.3	Condições de Corrida.....	4
5	Relação entre Computação Concorrente e Paralela e a Performance dos Algoritmos	
5		
5.1	Computação Concorrente .....	5
5.2	Computação Paralela .....	5
5.3	Performance dos Algoritmos .....	5
6	Experimento .....	6
6.1	Descrição do Experimento:.....	6
6.2	Gráfico dos tempos de execução:.....	6
6.3	Exemplo de resultado do output: .....	7
6.4	Imagens dos tempos de execução de cada experimento: .....	7
6.4.1	Execução do experimento com 1 thread .....	7
6.4.2	Execução do experimento com 3 threads .....	7
6.4.3	Execução do experimento com 9 threads.....	8
6.4.4	Execução do experimento com 27 threads.....	8
7	Conclusão.....	8
8	Referências .....	8

# 1 Introdução

Na era da informação, onde a demanda por processamento rápido e responsivo cresce exponencialmente, as threads emergem como ferramentas essenciais para orquestrar a execução eficiente de tarefas complexas. Este guia aborda os conceitos fundamentais das threads, seu funcionamento computacional, os impactos no tempo de execução dos algoritmos e a relação entre computação concorrente e paralela na performance dos algoritmos.

## 2 O que são Threads

Threads são unidades de execução dentro de um processo, permitindo que múltiplas operações sejam realizadas de maneira concorrente. Imagine um programa como um restaurante movimentado. As threads são como diferentes cozinheiros, cada um trabalhando em um prato específico. Cada cozinheiro (thread) possui suas próprias ferramentas (conjunto de registradores) e instruções (pilha de execução), mas todos dividem a mesma cozinha (espaço de memória do processo). Essa divisão do trabalho permite que o restaurante (programa) atenda a vários pedidos simultaneamente, proporcionando uma experiência mais rápida e eficiente para seus clientes.

## 3 Como Threads Funcionam Computacionalmente

### 3.1 Componentes Essenciais de uma Thread

Cada thread possui seus próprios componentes essenciais para funcionar:

- **Pilha de Execução:** Armazena o contexto da thread, incluindo instruções em andamento, variáveis locais e dados da pilha de chamadas. É como a lista de tarefas do cozinheiro, detalhando o que ele precisa fazer para preparar o prato.
- **Conjunto de Registradores:** Contém valores de variáveis e endereços de memória frequentemente utilizados pela thread. São como as ferramentas e ingredientes que o cozinheiro tem à mão para realizar seu trabalho.
- **Contador de Programa:** Indica a próxima instrução a ser executada pela thread. É como um ponteiro que indica ao cozinheiro qual passo da receita seguir.

### 3.2 Gerenciamento de Threads

O sistema operacional assume o papel de maestro, gerenciando as threads de forma eficiente. Ele aloca tempo de CPU para cada thread, garante a sincronização do acesso a recursos compartilhados e resolve conflitos entre elas. Imagine o maestro coordenando os cozinheiros, garantindo que cada um tenha tempo para preparar seu prato sem atrapalhar os outros.

### 3.3 Mecanismos de Sincronização

Em um ambiente multithread, onde vários cozinheiros trabalham simultaneamente, a sincronização é crucial para evitar problemas como condições de corrida, que podem corromper dados e gerar resultados inconsistentes. Mecanismos como mutex, semáforos e barreiras garantem que as threads acessem os recursos compartilhados de forma ordenada e segura. Imagine os cozinheiros usando um sistema de senhas para acessar os fornos, evitando que colidam uns com os outros.<sup>4</sup>

## 4 Como o Uso de Threads Pode Afetar o Tempo de Execução de um Algoritmo

### 4.1 Execução Paralela

Em sistemas com múltiplos processadores ou núcleos, as threads podem ser executadas simultaneamente, dividindo o trabalho entre as unidades de processamento. Isso leva a uma redução significativa no tempo total de execução, especialmente para tarefas computacionalmente intensivas. Imagine os pratos sendo preparados em diferentes fogões ao mesmo tempo, reduzindo o tempo total para servir a refeição.

### 4.2 Overhead

A criação e o gerenciamento de threads possuem um custo associado, conhecido como overhead. Este overhead inclui o tempo gasto pelo sistema operacional para criar, agendar e gerenciar threads, bem como para sincronizar o acesso a recursos compartilhados. Embora as threads possam melhorar significativamente o desempenho, é crucial equilibrar os benefícios do paralelismo com os custos do overhead.

### 4.3 Condições de Corrida

As condições de corrida ocorrem quando duas ou mais threads tentam acessar e modificar os mesmos dados simultaneamente, levando a resultados imprevisíveis e possivelmente incorretos. Para evitar essas condições, é necessário utilizar mecanismos de sincronização, como mutexes e semáforos, que garantem que apenas uma thread acesse os dados compartilhados por vez.

## 5 Relação entre Computação Concorrente e Paralela e a Performance dos Algoritmos

### 5.1 Computação Concorrente

A computação concorrente refere-se à execução de múltiplas tarefas de forma que elas possam progredir simultaneamente, mas não necessariamente ao mesmo tempo. Em um ambiente concorrente, as threads são gerenciadas de forma que compartilhem o tempo de CPU, proporcionando uma interface mais responsiva e eficiente.

### 5.2 Computação Paralela

A computação paralela, por outro lado, envolve a execução simultânea de múltiplas tarefas em múltiplas unidades de processamento. Este modelo é particularmente eficaz para algoritmos que podem ser divididos em sub-tarefas independentes, permitindo que cada thread execute uma parte do trabalho em paralelo, resultando em uma significativa redução no tempo de execução.

### 5.3 Performance dos Algoritmos

A escolha entre computação concorrente e paralela depende da natureza do algoritmo e dos recursos disponíveis. Algoritmos que podem ser facilmente divididos em sub-tarefas independentes se beneficiam mais da computação paralela, enquanto tarefas que requerem frequente interação e compartilhamento de dados podem ser mais adequadas para um modelo concorrente. A chave para maximizar a performance é compreender as características do problema e aplicar o modelo adequado de threads.

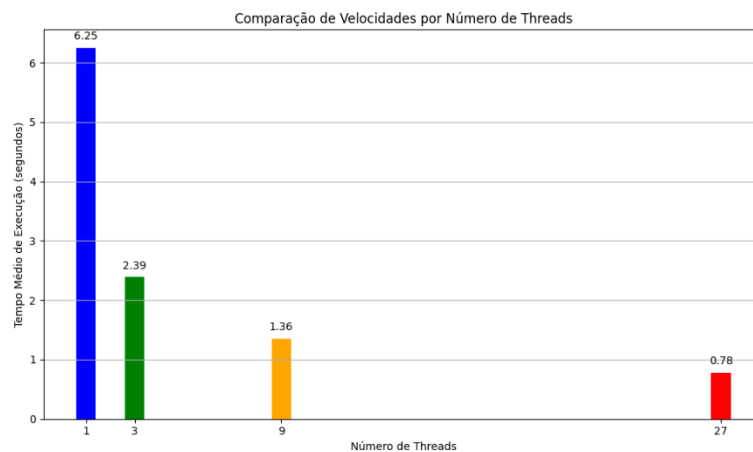
## 6 Experimento

### 6.1 Descrição do Experimento:

Os experimentos foram conduzidos com o objetivo de verificar como um mesmo sistema se comporta ao utilizar diferentes configurações de threads. A análise focou no impacto dessas configurações no processamento eficiente de requisições HTTP para coletar dados de temperatura das capitais do Brasil.

### 6.2 Gráfico dos tempos de execução:

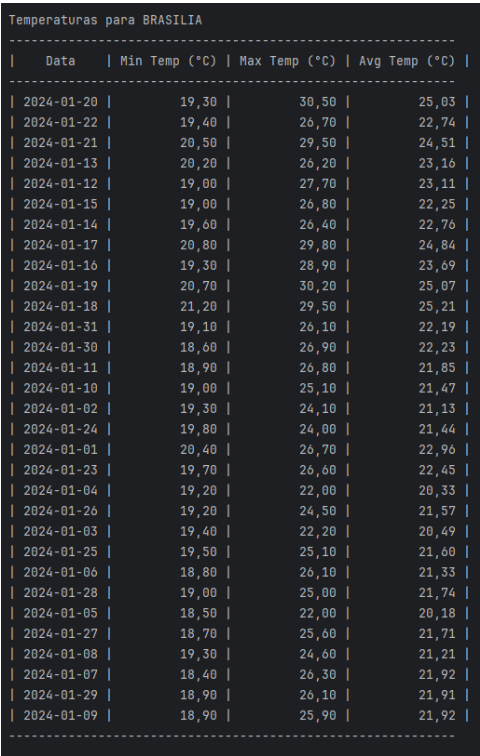
Como apontado pelo gráfico abaixo, a quantidade de threads em um sistema pode influenciar significativamente na velocidade do processamento. Entretanto, durante os testes realizados, foi observado que esse impacto também depende consideravelmente da velocidade da internet do usuário que está executando o código. Isso ocorre porque, ao utilizar múltiplas threads, o sistema pode distribuir tarefas entre os núcleos do processador de forma mais eficiente, aumentando a capacidade de processamento geral. No entanto, se a conexão de internet do usuário for lenta, pode haver atrasos na comunicação entre as threads ou na obtenção de recursos externos pela aplicação, o que pode mitigar os ganhos de desempenho esperados pelo aumento no número de threads.



*Figura 1 Imagem do gráfico dá média de tempo por número de threads*

### 6.3 Exemplo de resultado do output:

A imagem abaixo é um print de como o sistema mostra as temperaturas de uma capital

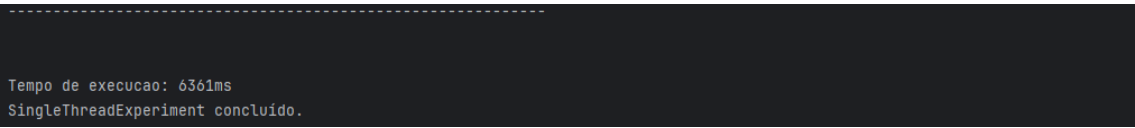
A terminal window with a dark background displaying a table of temperature data for Brasília. The table has four columns: 'Data', 'Min Temp (°C)', 'Max Temp (°C)', and 'Avg Temp (°C)'. It contains 30 rows of data, each representing a date in January 2024 and its corresponding temperature statistics.

Data	Min Temp (°C)	Max Temp (°C)	Avg Temp (°C)
2024-01-20	19,30	30,50	25,03
2024-01-22	19,40	26,70	22,74
2024-01-21	20,50	29,50	24,51
2024-01-13	20,20	26,20	23,16
2024-01-12	19,00	27,70	23,11
2024-01-15	19,00	26,80	22,25
2024-01-14	19,60	26,40	22,76
2024-01-17	20,80	29,80	24,84
2024-01-16	19,30	28,90	23,69
2024-01-19	20,70	30,20	25,07
2024-01-18	21,20	29,50	25,21
2024-01-31	19,10	26,10	22,19
2024-01-30	18,60	26,90	22,23
2024-01-11	18,90	26,80	21,85
2024-01-10	19,00	25,10	21,47
2024-01-02	19,30	24,10	21,13
2024-01-24	19,80	24,00	21,44
2024-01-01	20,40	26,70	22,96
2024-01-23	19,70	26,60	22,45
2024-01-04	19,20	22,00	20,33
2024-01-26	19,20	24,50	21,57
2024-01-03	19,40	22,20	20,49
2024-01-25	19,50	25,10	21,60
2024-01-06	18,80	26,10	21,33
2024-01-28	19,00	25,00	21,74
2024-01-05	18,50	22,00	20,18
2024-01-27	18,70	25,60	21,71
2024-01-08	19,30	24,60	21,21
2024-01-07	18,40	26,30	21,92
2024-01-29	18,90	26,10	21,91
2024-01-09	18,90	25,90	21,92

Figura 2 Imagem do resultado do sistema da capital Brasília

### 6.4 Imagens dos tempos de execução de cada experimento:

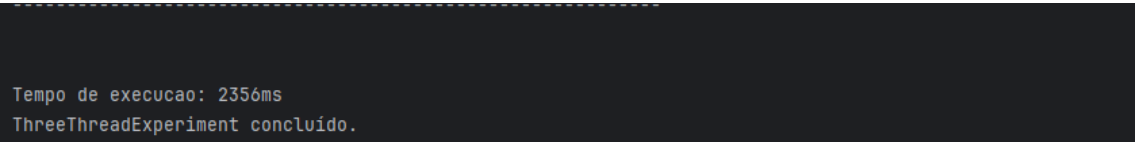
#### 6.4.1 Execução do experimento com 1 thread

A terminal window with a dark background showing the execution time for a single-threaded experiment. The output indicates a duration of 6361ms and that the experiment was concluded.

```
-----  
Tempo de execucao: 6361ms  
SingleThreadExperiment concluído.
```

Figura 3 Imagem do tempo de execução de uma 1 thread

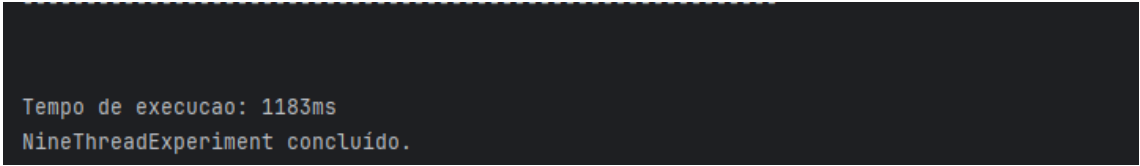
#### 6.4.2 Execução do experimento com 3 threads

A terminal window with a dark background showing the execution time for a three-threaded experiment. The output indicates a duration of 2356ms and that the experiment was concluded.

```
-----  
Tempo de execucao: 2356ms  
ThreeThreadExperiment concluído.
```

Figura 4 Imagem do tempo de execução de 3 threads

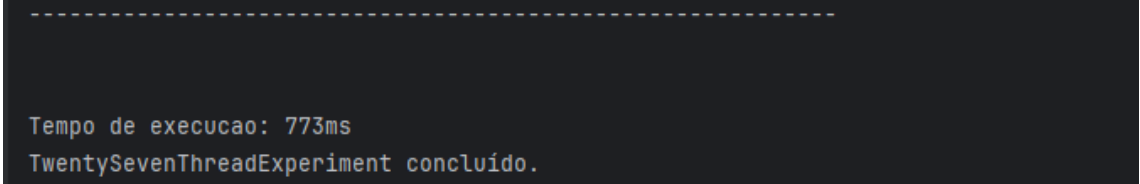
### 6.4.3 Execução do experimento com 9 threads



```
-----  
Tempo de execucao: 1183ms  
NineThreadExperiment concluído.
```

*Figura 5 Imagem do tempo de execução de 9 threads*

### 6.4.4 Execução do experimento com 27 threads



```
-----  
Tempo de execucao: 773ms  
TwentySevenThreadExperiment concluído.
```

*Figura 6 Imagem do tempo de execução de 27 threads*

## 7 Conclusão

Em suma, as threads são ferramentas poderosas para melhorar a eficiência computacional, permitindo a execução concorrente e paralela de tarefas dentro de um sistema. A escolha adequada do número de threads e a implementação correta de mecanismos de sincronização são cruciais para maximizar os benefícios do paralelismo, enquanto se minimiza o overhead e se evita condições de corrida. Além disso, entender a relação entre computação concorrente e paralela ajuda a decidir qual abordagem é mais adequada para diferentes tipos de algoritmos e cenários de uso.

## 8 Referências

Tanenbaum, A. S., & Bos, H. (2014). **Modern Operating Systems** (4th ed.). Pearson.