

A close-up photograph of several reddish-brown ants walking along a rough, greyish-brown tree trunk. The background is a soft, out-of-focus green, suggesting foliage. The ants are in various positions, some facing left and some right, creating a sense of movement.

Otimização por colônias de formigas

DAVI TEIXEIRA, ENZO LEMES MARQUES, LISANDRA MENEZES E
PEDRO RIBEIRO FERNANDES

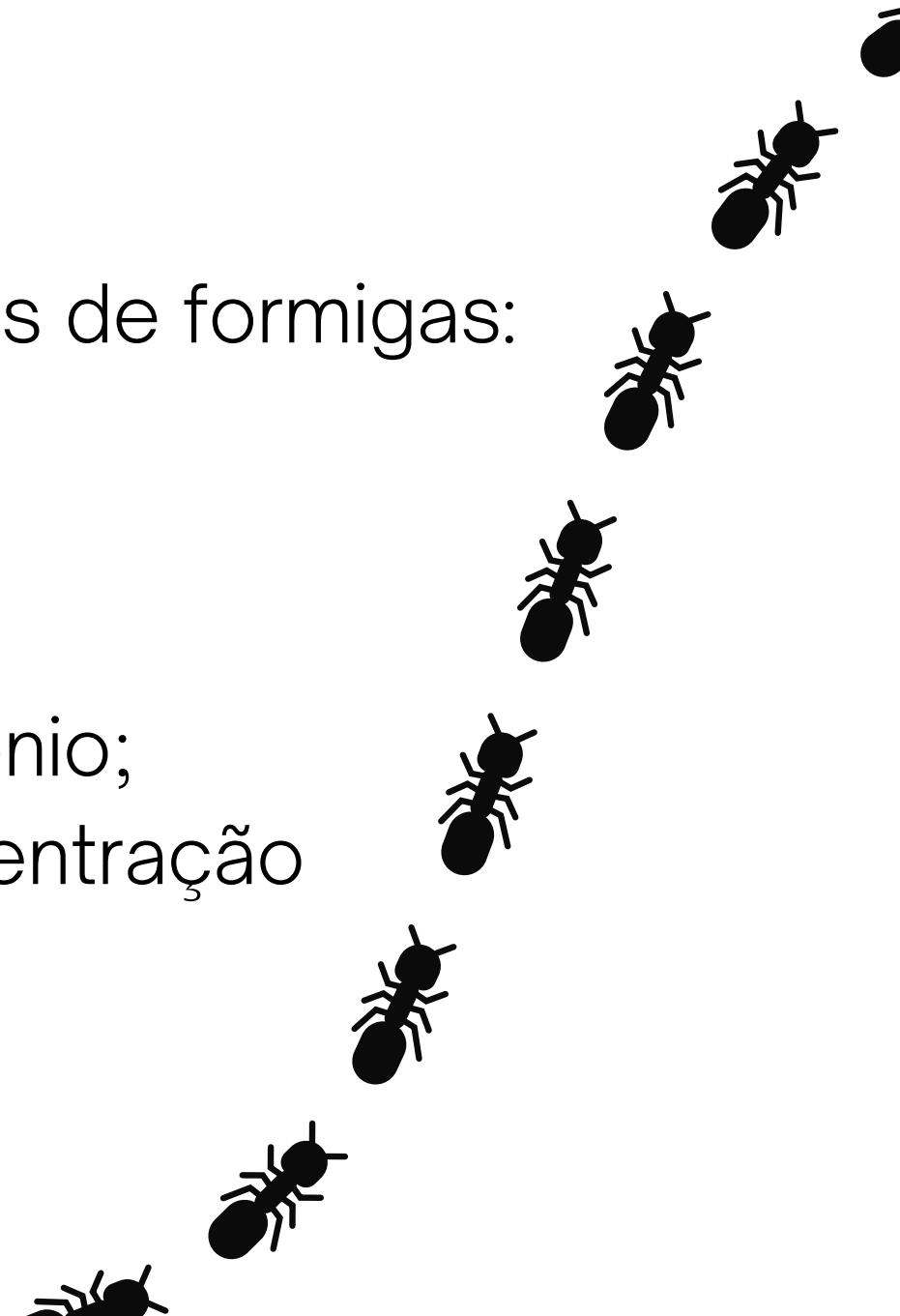
Sumário

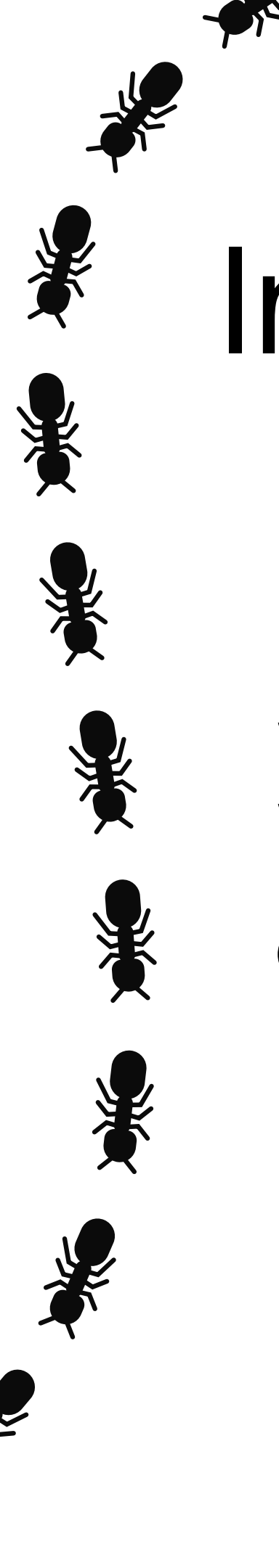
- Introdução ao problema e algoritmo
- Por que paralelizar
- Metodologia
- Resultados
- Conclusões

Introdução

Otimização por Colônia de Formigas (ACO)

- Metaheurística inspirada no comportamento natural de colônias de formigas:
 - Início aleatória em busca de comida;
 - Deposição de feromônio no caminho ida/volta;
 - Outras formigas se atraem pelo caminho com mais feromônio;
 - Com o tempo, caminhos mais curtos recebem maior concentração de feromônio.



A decorative vertical line of black ant icons runs down the left side of the slide, starting from the top left and ending near the bottom left.

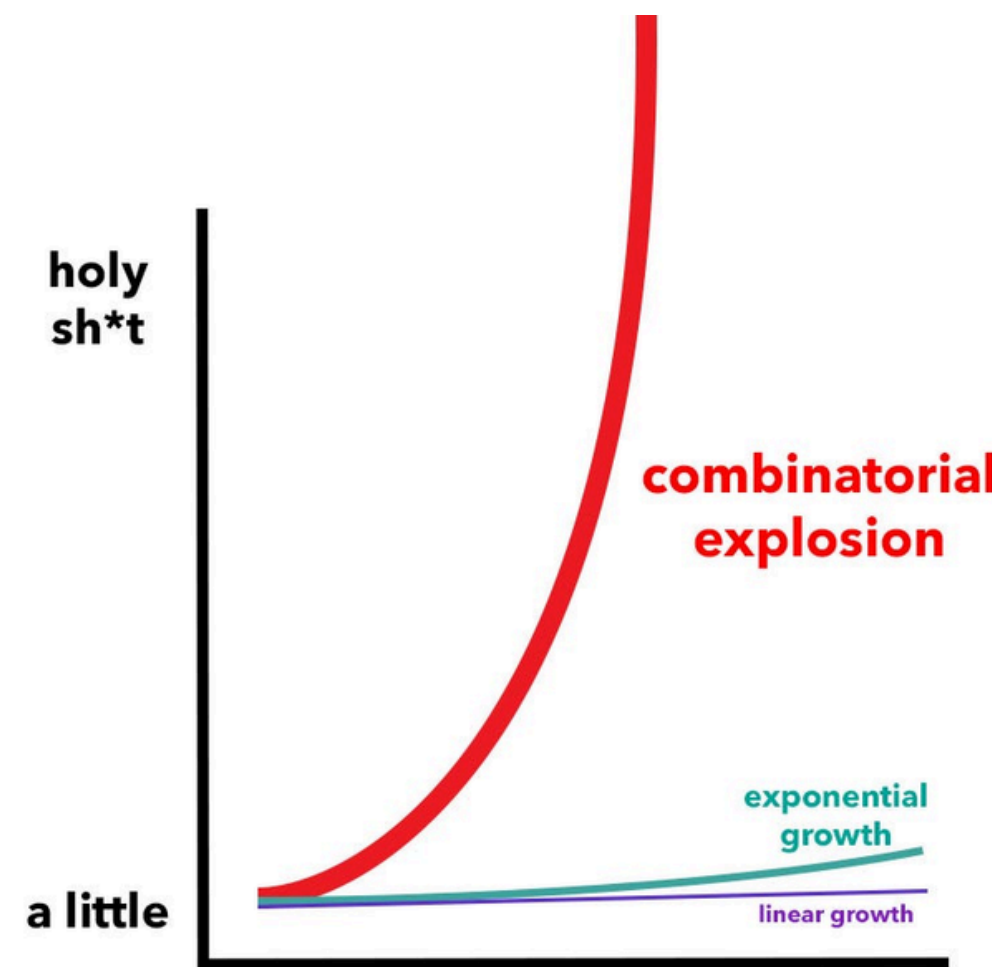
Introdução

Problemas de Otimização Combinatória envolvem encontrar uma solução satisfatória a partir de um conjunto finito, mas possivelmente muito grande, de soluções possíveis.

Características:

- Discretos e Finitos: As soluções são compostas de combinações ou permutações de um conjunto finito de elementos.
- Exemplo Clássico: Problema do Caixeiro Viajante (TSP)

Otimização Combinatória



Taxa de crescimento de um problema de otimização combinatória.

Por que paralelizar

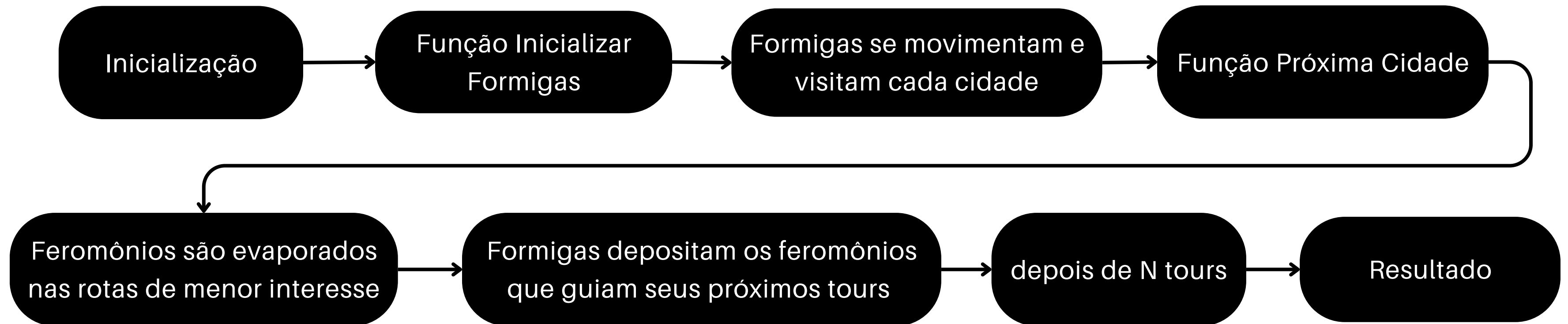
1. **Problemas de otimização combinatória**, apesar de serem aparentemente simples, se tornam extremamente complexos à medida que a quantidade de elementos combinatórios aumenta.
2. A paralelização do algoritmo de colônia de formigas é interessante pois é um **algoritmo computacionalmente intensivo** de acordo com o número de instâncias do problema.

Metodologia: Sequencial

Definição de constantes, variáveis globais e matrizes (distâncias entre as cidades e feromônio).

Inicializar cada formiga com cidade inicial, caminho, visitado e comprimento do tour

Calcula a probabilidade de cada cidade a ser visitada com base na atratividade e quantidade de feromônio.



Ao se movimentarem, selecionam uma cidade inicial aleatória, enquanto houver cidade a ser visita utiliza a função probabilística para escolher a próxima cidade.

Ao completar o tour, aplica-se uma taxa de evaporação na matriz de feromônio e atualiza o feromônio com base no tour da formiga.

Inicializa as funções e imprime o melhor resultado obtido e o tempo de execução.

Resultados: Sequencial

O tempo de execução e os custos associados às matrizes foram:

Matriz 100x100

- Melhor distância: 845.702881
- Tempo para gerar: 43.423530 segundos

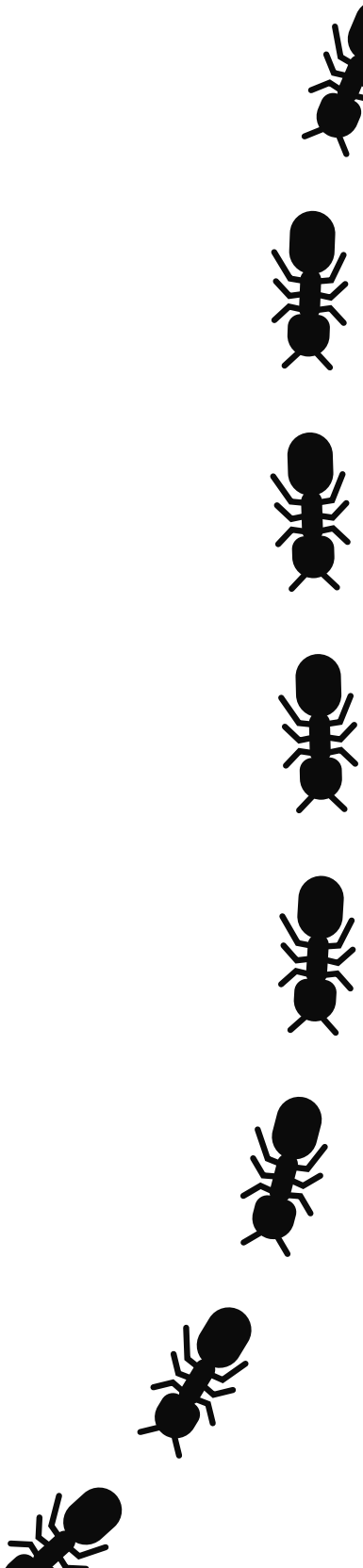
Matriz 50x50

- Melhor distância: 572.472900
- Tempo para gerar: 12.063116 segundos

Matriz 25x25

- Melhor distância: 408.462555
- Tempo para gerar: 3.407926 segundos

Intel® Core™ i7-11800H de 11ª geração, cache de 24 MB, 8 núcleos, até 4,60 GHz



Metodologia: Open MP

Divisão do Trabalho:

- Regiões paralelas garantem que cada thread do gereencie uma formiga.
 - Inclui inicialização, movimento e atualização de feromônios.

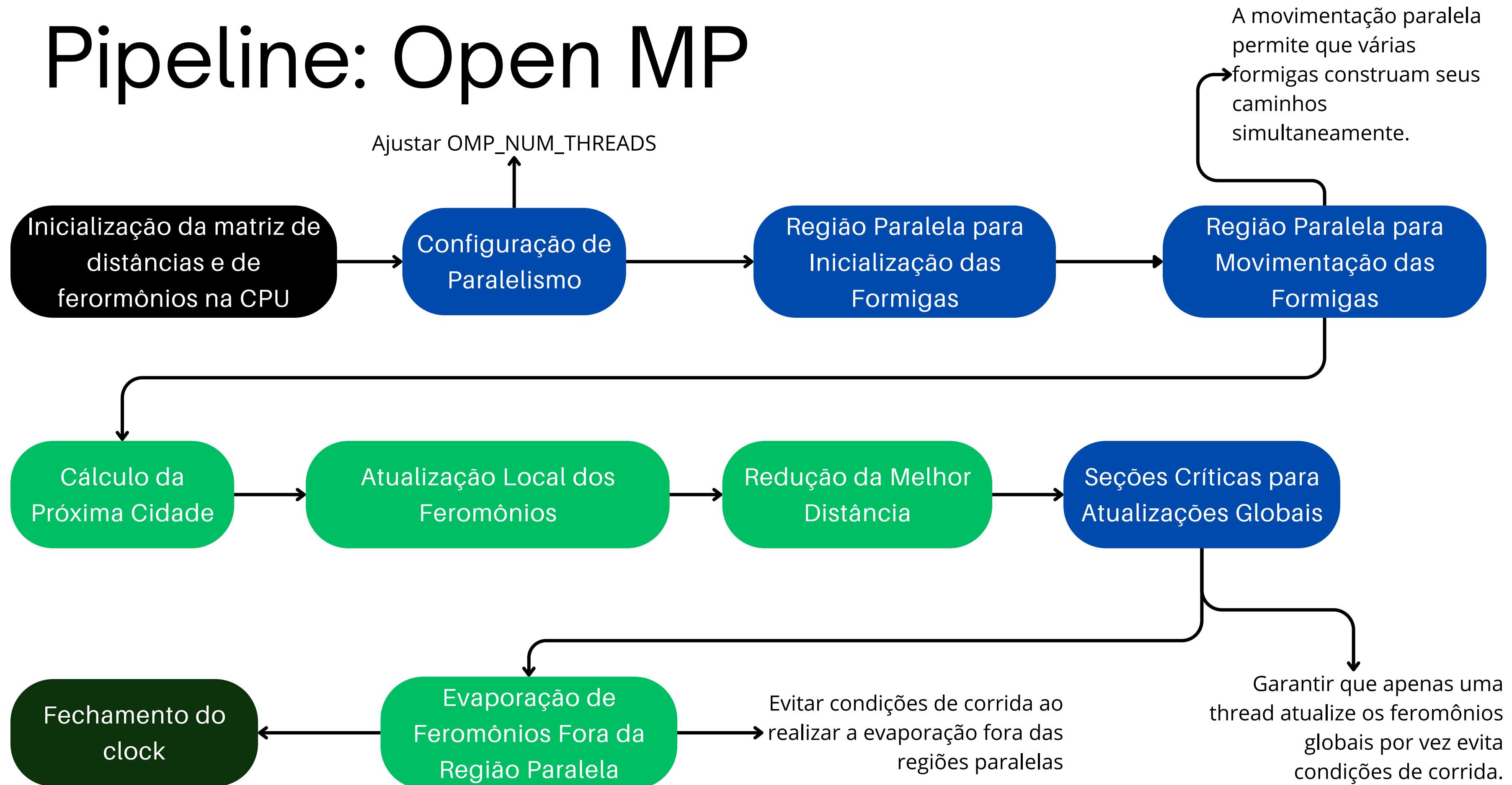
Uso de Memória Compartilhada e Local:

- Matrizes de distância e feromônio são compartilhadas entre threads.
- Atualizações de feromônio são mantidas localmente e combinadas ao final.

Regiões Críticas e Sincronização:

- Seções críticas evitam condições de corrida nas atualizações globais.
 - Cláusulas 'atomic' garantem atualizações seguras de variáveis compartilhadas.

Pipeline: Open MP



Resultados: OpenMP

O tempo de execução e os custos associados às matrizes foram:

Matriz 100x100

- Melhor distância: 906.311829
- Tempo para gerar: 31.147344 segundos

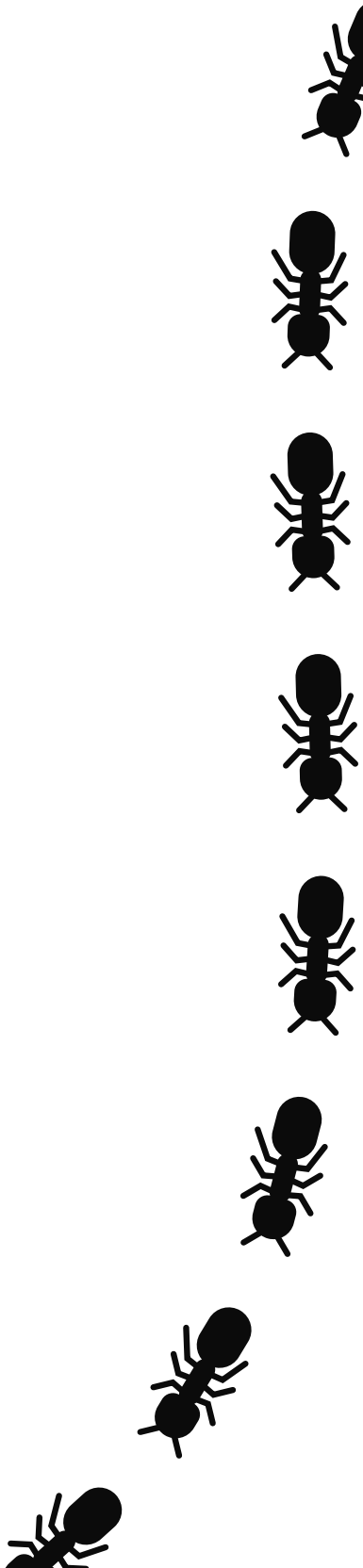
Matriz 50x50

- Melhor distância: 583.176758
- Tempo para gerar: 8.330663 segundos

Matriz 25x25

- Melhor distância: 408.462585
- Tempo para gerar: 2.083050 segundos

Intel® Core™ i7-11800H de 11ª geração, cache de 24 MB, 8 núcleos, até 4,60 GHz



Metodologia: CUDA

Divisão do trabalho:

- Cada thread na GPU gerencia uma formiga individual.
 - Inclui inicialização, simulação de movimentos e atualização de feromônios.

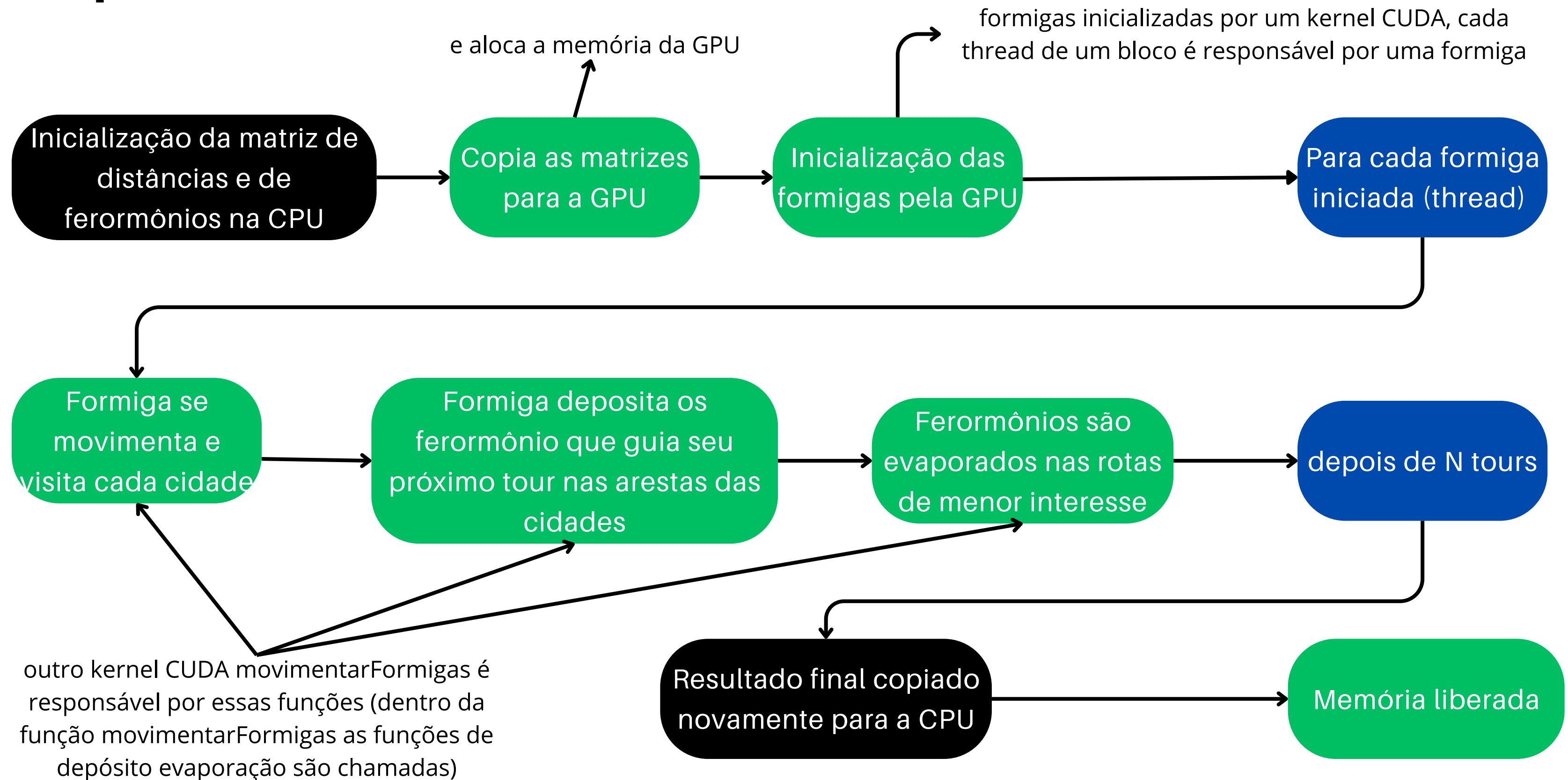
Uso de Memória Compartilhada e Global:

- Matrizes de distância e feromônio armazenadas na memória global.
 - Acesso rápido e simultâneo por múltiplas threads melhora a eficiência e velocidade.

Operações Atômicas:

- Evitam condições de corrida durante atualização dos feromônios.
 - Garantem que as atualizações sejam seguras e não sobrepostas.

Pipeline: CUDA



Resultados: CUDA

O tempo de execução e os custos associados às matrizes foram:

Matriz 100x100

- Melhor distância: 854.655701
- Tempo para gerar: 13.263616 segundos

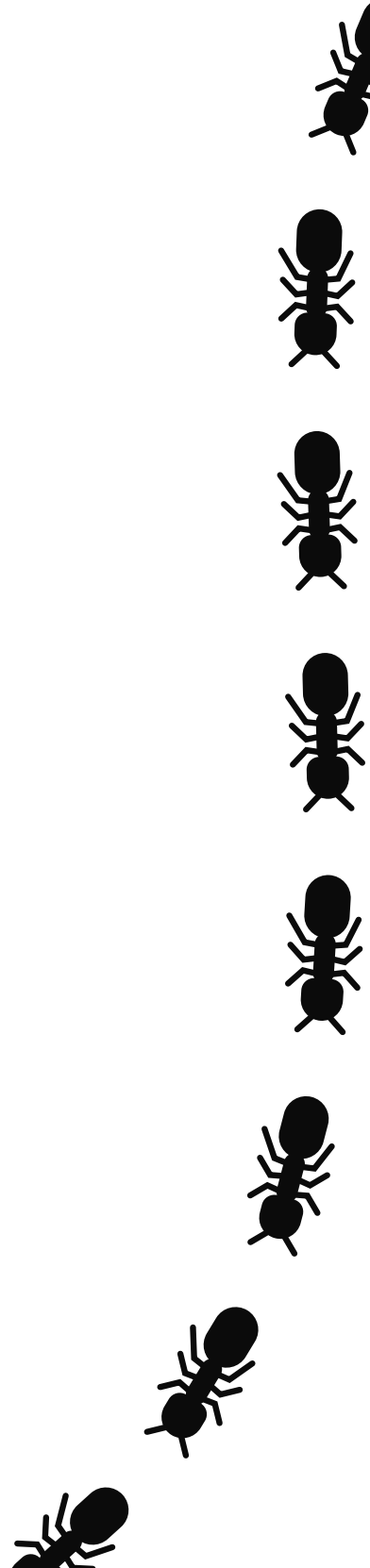
Matriz 50x50

- Melhor distância: 583.426331
- Tempo para gerar: 3.853429 segundos

Matriz 25x25

- Melhor distância: 407.936554
- Tempo para gerar: 0.940340 segundos

Geforce RTX 3070



Profiling da execução:

```
[5/8] Executing 'cuda_api_sum' stats report
```

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
99,0	17.371.864.637	2	8.685.932.318,0	8.685.932.318,0	11.663.656	17.360.200.981	12.267.268.386,0	cudaDeviceSynchronize
0,0	49.564.626	5	9.912.925,0	3.702,0	2.001	49.524.350	22.143.463,0	cudaMalloc
0,0	7.936.574	2	3.968.287,0	3.968.287,0	148.386	7.788.188	5.402.155,0	cudaLaunchKernel
0,0	1.251.164	5	250.232,0	18.868,0	8.271	1.160.649	509.240,0	cudaMemcpy
0,0	249.716	5	49.943,0	6.771,0	2.341	118.243	62.308,0	cudaFree
0,0	2.041	1	2.041,0	2.041,0	2.041	2.041	0,0	cuModuleGetLoadingMode

```
[6/8] Executing 'cuda_gpu_kern_sum' stats report
```

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
99,0	17.360.128.826	1	17.360.128.826,0	17.360.128.826,0	17.360.128.826	17.360.128.826	0,0	moverFormigasCUDA(formiga *, float *, float *, int, int, float, float, float, int)
0,0	10.020.914	1	10.020.914,0	10.020.914,0	10.020.914	10.020.914	0,0	inicializarFormigasCUDA(formiga *, int, int, unsigned int, curandStateXORWOW *)

```
[7/8] Executing 'cuda_gpu_mem_time_sum' stats report
```

Time (%)	Total Time (ns)	Count	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Operation
99,0	787.145	2	393.572,0	393.572,0	1.216	785.929	554.875,0	[CUDA memcpy Device-to-Host]
1,0	8.320	3	2.773,0	3.904,0	417	3.999	2.041,0	[CUDA memcpy Host-to-Device]

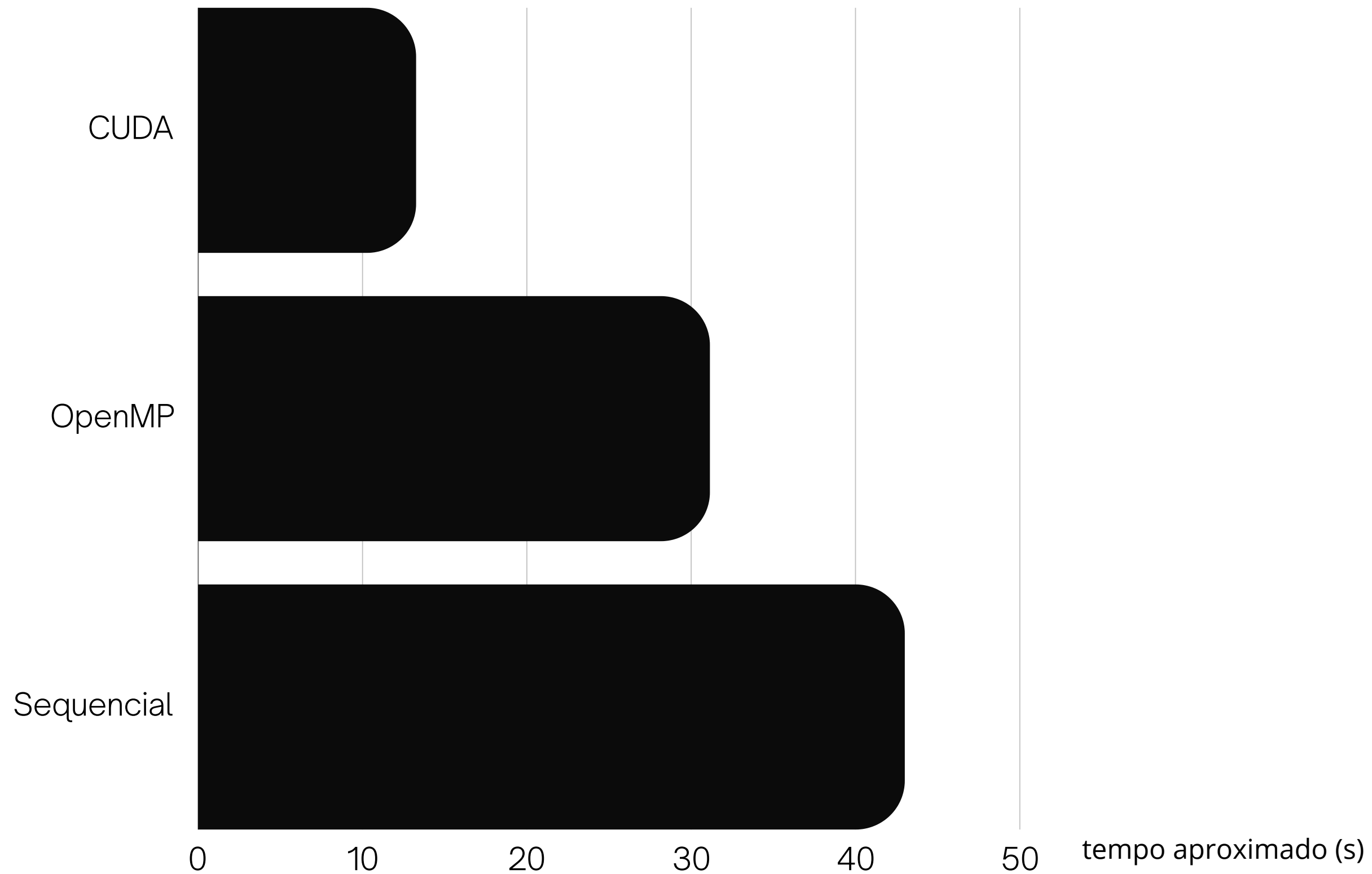
```
[8/8] Executing 'cuda_gpu_mem_size_sum' stats report
```

Total (MB)	Count	Avg (MB)	Med (MB)	Min (MB)	Max (MB)	StdDev (MB)	Operation
3,264	2	1,632	1,632	0,000	3,264	2,308	[CUDA memcpy Device-to-Host]
0,080	3	0,027	0,040	0,000	0,040	0,023	[CUDA memcpy Host-to-Device]

Resultados



Tempo de execução



Speedup absoluto

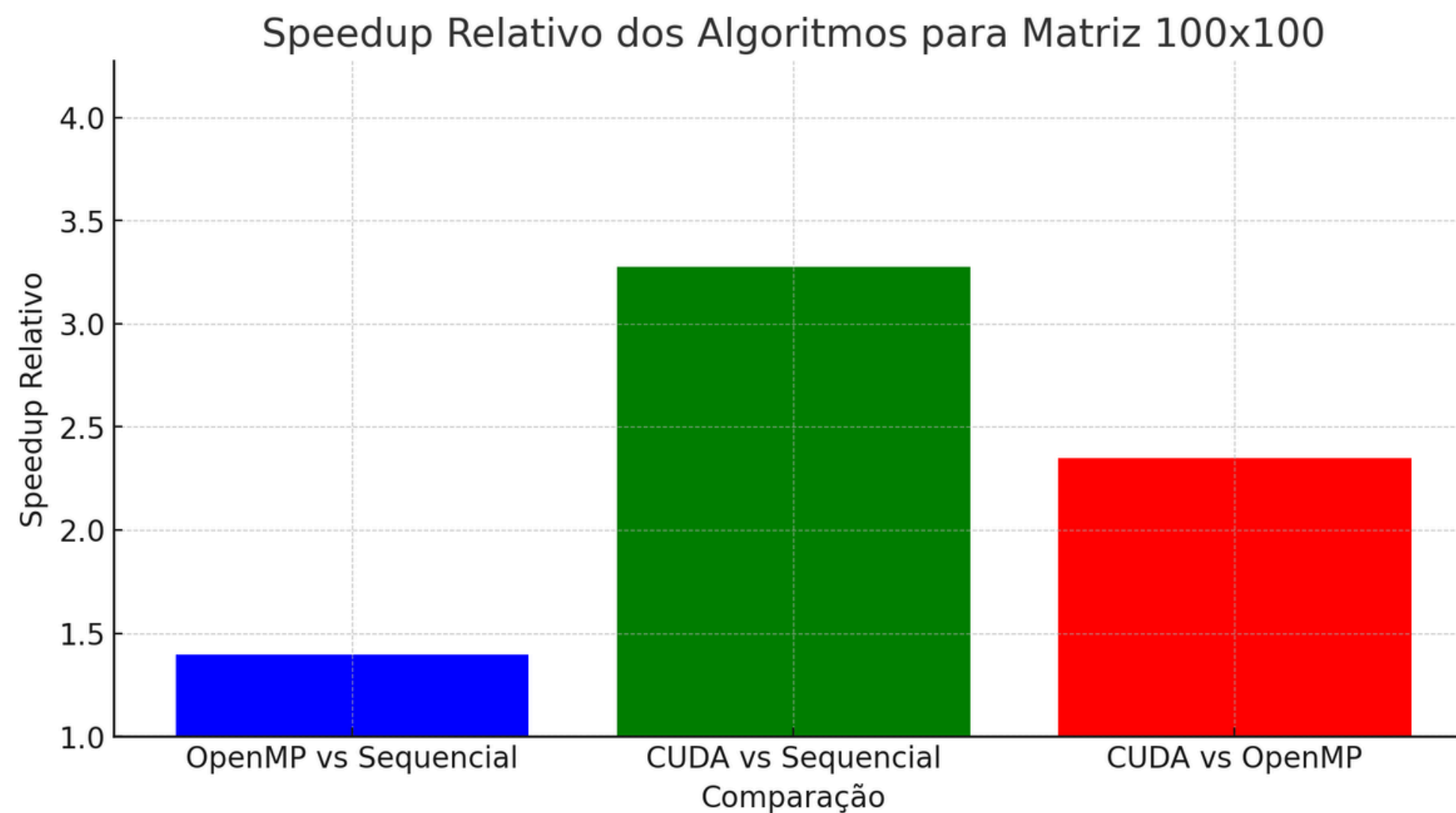
$$S = \frac{\textit{tempoAntigo}}{\textit{tempoNovo}}$$

Onde:

S: Speedup absoluto

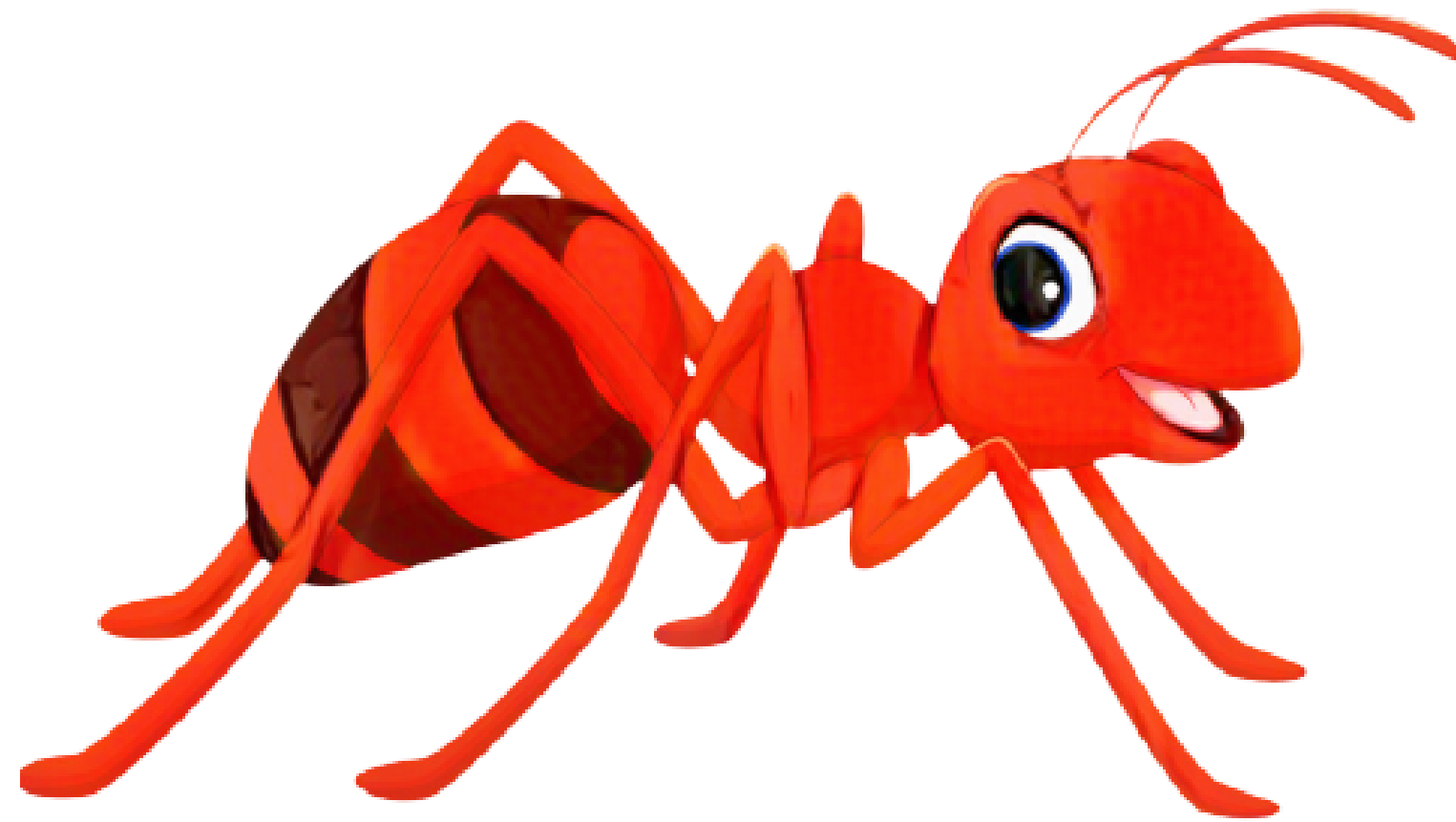
tempoAntigo: tempo de exec. sequencial

tempoNovo: tempo de exec. paralela



Conclusão:

- Percebemos que meta-heurísticas podem se beneficiar de implementações paralelas, pois podemos processar mais instâncias de um problema de uma só vez, assim, aumentando o desempenho dessas técnicas para a resolução de problemas reais.
- Além disso, ficou clara a vantagem competitiva do CUDA dentre as opções de técnicas que possibilitam o paralelismo, ressaltando a importância das GPGPUs para o processamento computacional hodierno.



Obrigado pela atenção!