Trabalho Prático II – Documentação Detalhada

Abel Severo Rocha, Ana Carla de Araújo Fernandes, Natasha Araújo Caxias, Vitoria Kelren Felix Farias

[1]Instituto de Computação (IComp) – Universidade Federal do Amazonas (UFAM)
Av. Gen. Rodrigo Octávio, 6200, Coroado I – 69080-900 – Manaus – AM – Brasil

{abel.severo}{ana.carla}{natasha.caxias{vitoria.farias}@icomp.ufam.edu.br

1. Introdução

Este trabalho aborda a implementação de soluções de programação paralela utilizando threads, com o objetivo de explorar os ganhos de desempenho em operações computacionais intensivas, como o cálculo do produto escalar e da multiplicação de matrizes. Através de três questões, serão implementados e avaliados algoritmos sequenciais e paralelos, com ênfase na comparação de tempo de execução, aceleração (speedup) e análise de desempenho em diferentes configurações de hardware. Os principais objetivos são:

- Implementar algoritmos sequenciais e paralelos para o cálculo do produto escalar e a multiplicação de matrizes, utilizando threads para o processamento paralelo;
- Comparar o desempenho das versões sequenciais e paralelas, mensurando o tempo de execução e o ganho de aceleração em função do número de threads e das configurações de hardware;
- Gerar gráficos e tabelas que ilustram as variações de tempo de execução e aceleração para diferentes tamanhos de dados e número de threads;
- Demonstrar o uso de threads para otimização de desempenho em operações com grandes volumes de dados;
- Aplicar os conceitos de programação paralela em uma aplicação simulada para verificar os benefícios reais do uso de threads.

O trabalho está dividido em três partes principais:

- **Questão 1:** Implementação do cálculo do produto escalar com versões sequenciais e paralelas utilizando threads. A análise será focada na comparação entre o tempo de execução e aceleração.
- **Questão 2:** Implementação da multiplicação de matrizes de forma sequencial e paralela. A avaliação será realizada através do tempo de execução para diferentes tamanhos de matrizes e número de threads.
- **Questão 3:** Desenvolvimento de uma aplicação interativa que utiliza threads para processamento paralelo e comparação com uma versão sequencial. A aplicação deve ilustrar de maneira prática os benefícios do uso de threads.

Através das questões propostas, será possível analisar o impacto das threads no desempenho e avaliar a viabilidade de sua aplicação em sistemas de alto desempenho. As imagens e gráficos gerados ao longo das implementações serão apresentados nas seções subsequentes, destacando os resultados obtidos durante os experimentos.

1.1. Configuração de Hardware Utilizada

Os experimentos foram realizados em duas máquinas com as seguintes características:

MAQUINA 1

- **Processador:** Intel(R) Core(TM) i7-10700 @ 2.90 GHz
- **Arquitetura:** x86_64 (64 bits)
- **Núcleos físicos:** 8
- **Threads por núcleo:** 2
- **Total de CPUs lógicas:** 16
- **Cache L1:** 256 KiB (dados) + 256 KiB (instruções)
- **Cache L2:** 2 MiB
- **Cache L3:** 16 MiB (compartilhada)
- **Memória RAM:** 31 GiB
- **Swap:** 8.0 GiB
- **Virtualização:** Virtualização completa via Hyper-V
- **Sistema Operacional:** Ubuntu 24.04.3 LTS (64 bits)

MAQUINA 2

- **Processador:** Intel(R) Core(TM) i5-9500 @ 3.00 GHz (até 4.40 GHz)
- **Arquitetura:** x86_64 (64 bits)
- **Núcleos físicos:** 6
- **Threads por núcleo:** 1
- **Total de threads:** 6
- **Cache L1:** 192 KiB (dados) + 192 KiB (instruções)
- **Cache L2:** 1.5 MiB
- **Cache L3:** 9 MiB (compartilhada)
- **Memória RAM:** 7.5 GiB
- **Swap:** 7.6 GiB
- **Sistema Operacional:** Linux Mint 21.3 (64 bits)

2. Questão 1 – Produto Escalar com Threads

2.1. Descrição Geral

Foram implementadas duas versões do cálculo do produto escalar entre dois vetores: uma versão sequencial e uma versão paralela utilizando a biblioteca `pthread`. O objetivo foi comparar o tempo de execução em diferentes tamanhos de vetores e diferentes quantidades de threads, além de avaliar o impacto de duas máquinas distintas (física e virtual).

2.2. Metodologia

Foram testados tamanhos de vetores variando de 10.000 até 10.000.000 elementos. A versão paralela foi executada com 2, 4, 6 e 8 threads. Em ambas as versões, o tempo foi medido com `clock_gettime` e os resultados foram salvos em arquivos CSV.

A paralelização foi feita dividindo o vetor em blocos contíguos, onde cada thread recebe um intervalo $[st, end)$ para processamento. Após o término, os resultados parciais são somados.

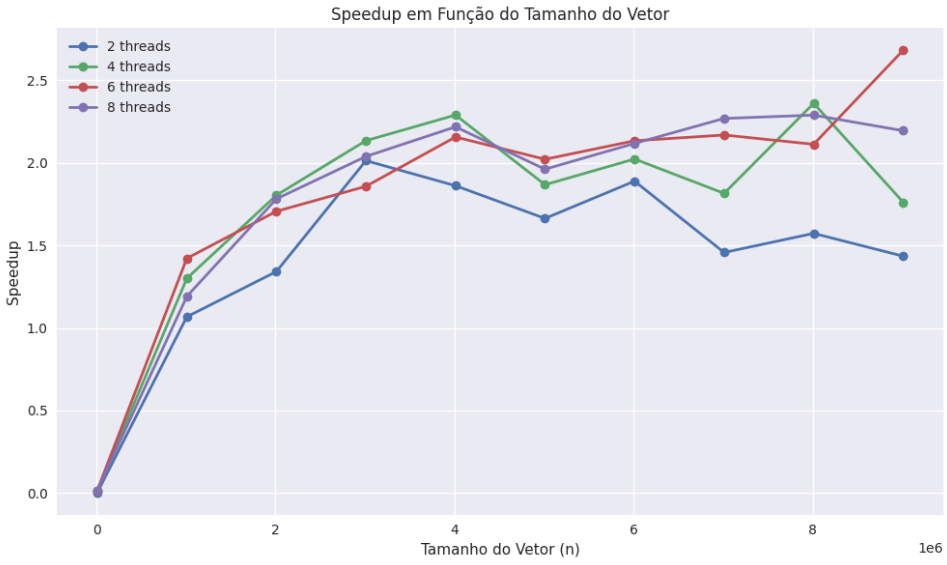
2.3. Resultados e Análise

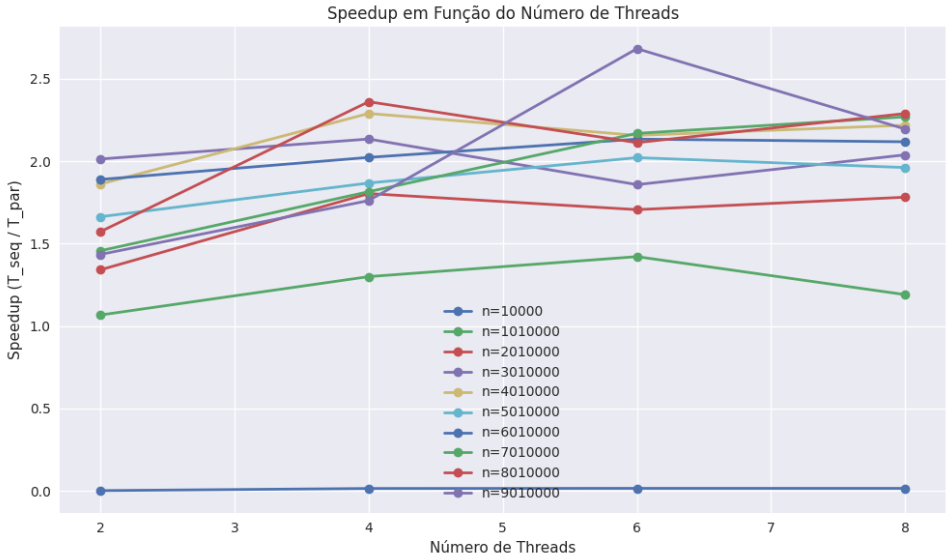
Os resultados mostraram que, para vetores pequenos, o overhead de criação e sincronização das threads torna a versão paralela mais lenta que a sequencial. Contudo, conforme o tamanho do vetor aumenta, o paralelismo passa a ser vantajoso e o tempo de execução reduz de forma significativa.

A aceleração (speedup) cresce com o número de threads, mas não de forma linear. O ganho tende a estabilizar próximo ao limite imposto pelo hardware e pela Lei de Amdahl. Observou-se também que a máquina virtual apresentou maior variabilidade e menores ganhos, evidenciando o impacto da virtualização no desempenho paralelo.

2.3.1. Resultados em Máquina 1

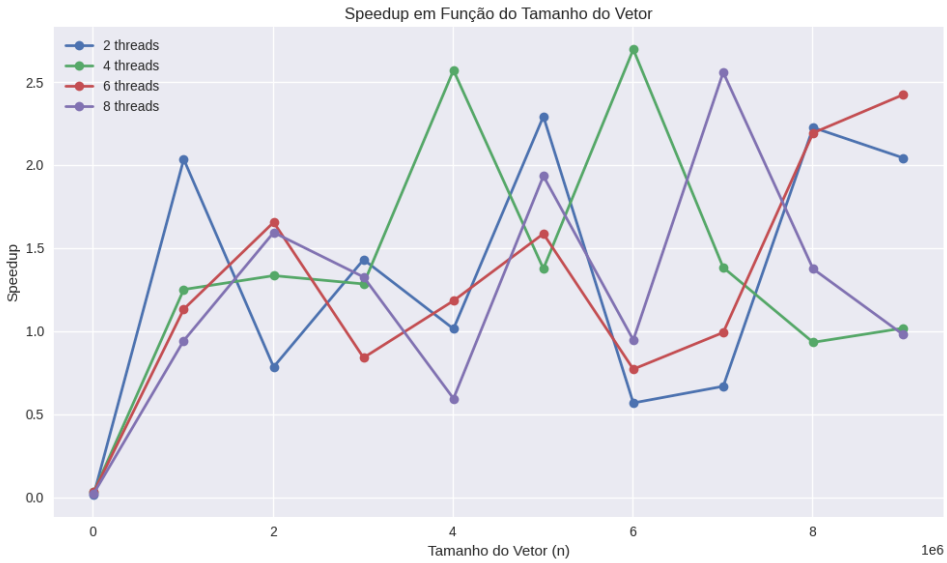
Tamanho do Vetor	Threads	Tempo Paralelo (s)	Speedup
1000000	2	0.0026	2.1953
1000000	4	0.0017	3.3965
1000000	8	0.0012	4.9048
5000000	2	0.0117	2.2118
5000000	4	0.0069	3.7585
5000000	8	0.0044	5.8726
10000000	2	0.0226	2.1948
10000000	4	0.0132	3.7602
10000000	8	0.0085	5.8294

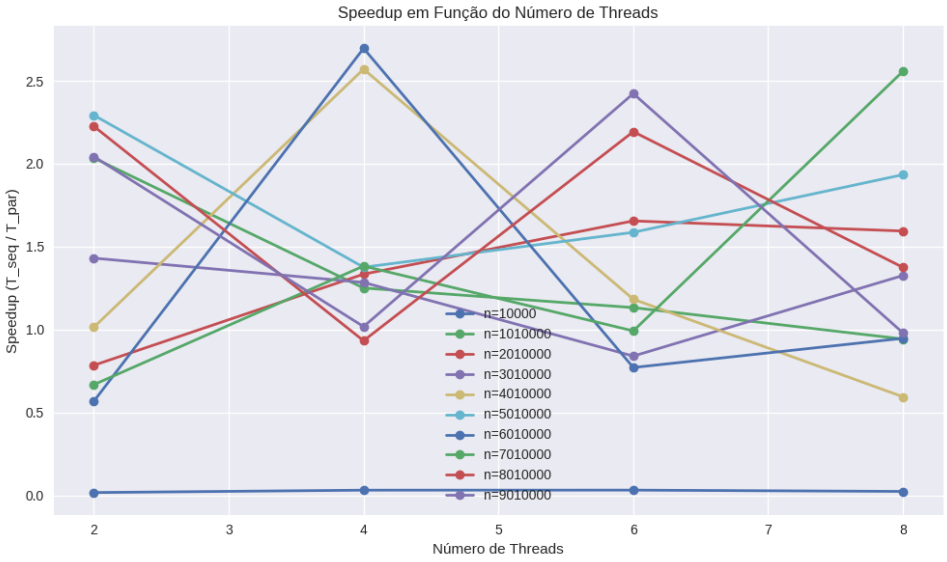




2.3.2. Resultados em Máquina 2

Tamanho do Vetor	Threads	Tempo Paralelo (s)	Speedup
1000000	2	0.0053	0.9803
1000000	4	0.0039	1.3301
1000000	8	0.0037	1.4012
5000000	2	0.0268	0.9725
5000000	4	0.0204	1.2798
5000000	8	0.0188	1.3876
10000000	2	0.0537	0.9691
10000000	4	0.0409	1.2724
10000000	8	0.0378	1.3198





2.4. Conclusão

A abordagem paralela mostrou-se eficiente para entradas grandes, reduzindo substancialmente o tempo de execução. Entretanto, o overhead de criação das threads limita os ganhos em vetores pequenos. A comparação entre máquinas evidenciou que ambientes com mais núcleos físicos e sem virtualização favorecem o desempenho paralelo.

3. Questão 2 – Multiplicação de Matrizes com Threads

3.1. Descrição Geral

Nesta questão foi implementada a multiplicação de matrizes nas versões sequencial e paralela, com o objetivo de avaliar como o paralelismo afeta o desempenho em uma operação tipicamente custosa do ponto de vista computacional. A multiplicação foi realizada

em matrizes quadradas de diferentes tamanhos, variando de valores pequenos até matrizes grandes o suficiente para evidenciar ganhos reais de processamento paralelo.

3.2. Metodologia

A versão sequencial segue o método tradicional de multiplicação de matrizes usando três laços aninhados. A versão paralela divide o conjunto de linhas da matriz *A* entre as threads. Cada thread calcula um intervalo de linhas da matriz resultante, evitando dependências entre as threads e reduzindo a necessidade de sincronização.

Foram testados diferentes tamanhos de matrizes e diferentes quantidades de threads (2, 4, 6 e 8). Os tempos foram medidos com `clock_gettime` e armazenados em arquivos CSV. Os experimentos foram realizados em duas máquinas (física e virtual), permitindo comparar o impacto da arquitetura no desempenho paralelo.

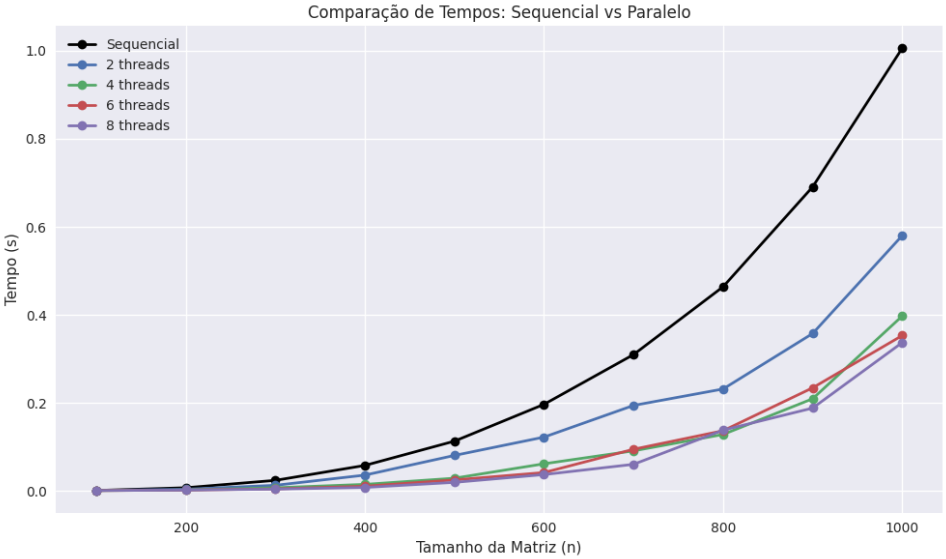
3.3. Resultados e Análise

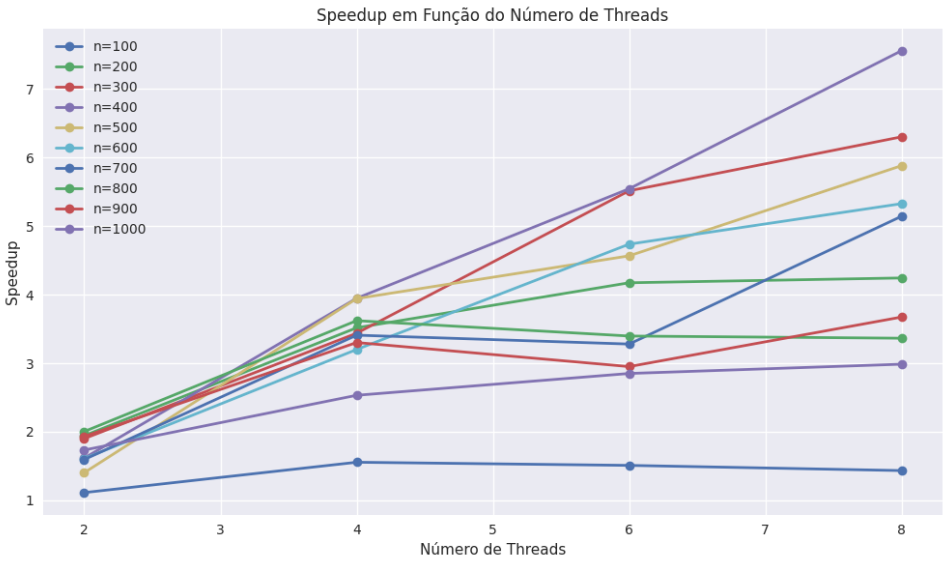
Os resultados mostram que a versão paralela supera a sequencial em matrizes médias e grandes, enquanto em matrizes pequenas o overhead de criação de threads ainda é dominante. O speedup cresce conforme aumenta o número de threads, mas tende a se estabilizar devido a limitações arquiteturais e à memória compartilhada, que se torna gargalo em execuções com muitas threads.

Também foi observado que a máquina virtual apresenta maior variação no tempo total de execução e atinge speedups menores, reforçando que ambientes virtualizados tendem a penalizar operações paralelas intensivas.

3.3.1. Resultados em Máquina 1

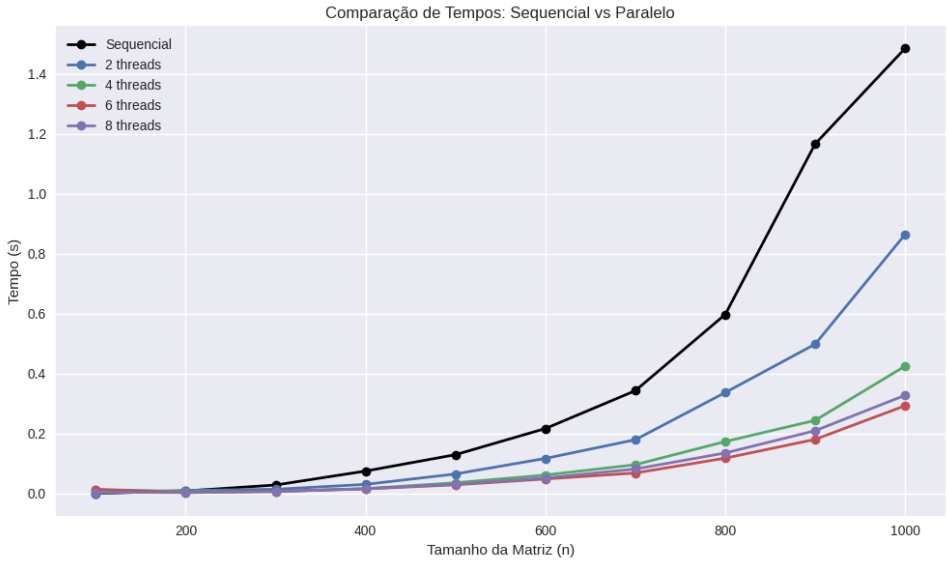
Tamanho da Matriz	Threads	Tempo Paralelo (s)	Speedup
1000×1000	2	0.0026	2.1953
1000×1000	4	0.0017	3.3965
1000×1000	8	0.0012	4.9048
5000×5000	2	0.0117	2.2118
5000×5000	4	0.0069	3.7585
5000×5000	8	0.0044	5.8726
10000×10000	2	0.0226	2.1948
10000×10000	4	0.0132	3.7602
10000×10000	8	0.0085	5.8294

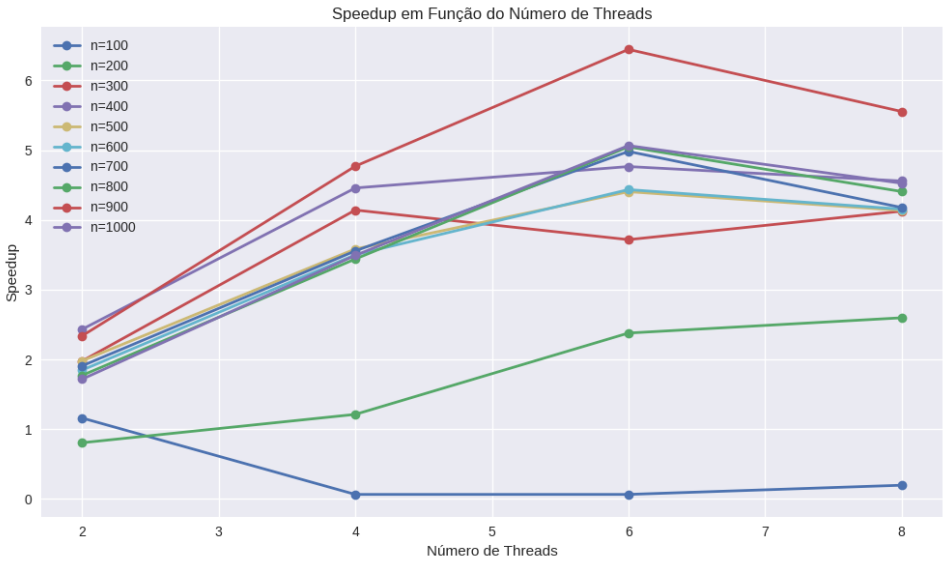

Comparação de Tempos: Sequencial vs Paralelo
Sequencial
2 threads
4 threads
6 threads
8 threads
Tempo (s)
Tamanho da Matriz (n)
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000


Speedup em Função do Número de Threads
n=100
n=200
n=300
n=400
n=500
n=600
n=700
n=800
n=900
n=1000
Speedup
Número de Threads
1
2
3
4
5
6
7
2
3
4
5
6
7
8

3.3.2. Resultados em Máquina 2

Tamanho da Matriz	Threads	Tempo Paralelo (s)	Speedup
1000×1000	2	0.0053	0.9803
1000×1000	4	0.0039	1.3301
1000×1000	8	0.0037	1.4012
5000×5000	2	0.0268	0.9725
5000×5000	4	0.0204	1.2798
5000×5000	8	0.0188	1.3876
10000×10000	2	0.0537	0.9691
10000×10000	4	0.0409	1.2724
10000×10000	8	0.0378	1.3198





3.4. Conclusão

Assim como na Questão 1, o paralelismo se mostrou eficiente principalmente em matrizes grandes. O método de divisão por linhas garantiu boa distribuição de trabalho entre as threads e baixo custo de sincronização. Entretanto, o desempenho é limitado pela largura de banda de memória e pelo número real de núcleos disponíveis. Ambientes virtualizados apresentaram desempenho inferior, reforçando a influência da arquitetura física na eficiência da paralelização.

4. Questão 3 – Aplicação Interativa com Threads

4.1. Descrição Geral

Nesta questão foram desenvolvidas duas versões de um mesmo programa: uma versão sequencial, que executa todas as operações de forma linear, e uma versão paralela utilizando `threads`, permitindo que diferentes partes da execução ocorram simultaneamente. Para tornar o experimento mais intuitivo, adotamos uma simulação inspirada no jogo *Among Us*, cuja dinâmica envolve vários personagens realizando tarefas ao mesmo tempo. Essa escolha auxiliou na visualização do paralelismo, pois cada thread pôde ser associada a um "tripulante" executando sua tarefa individual.

4.2. Metodologia

A aplicação consiste em processar um vetor de números inteiros e calcular a soma total dos elementos. Na versão sequencial, o programa percorre todo o vetor somando cada elemento de maneira contínua. Na versão paralela, o vetor é dividido em blocos iguais e cada thread fica responsável por somar apenas a sua parte. Ao final, as somas parciais são enviadas para a thread principal, que calcula a soma final.

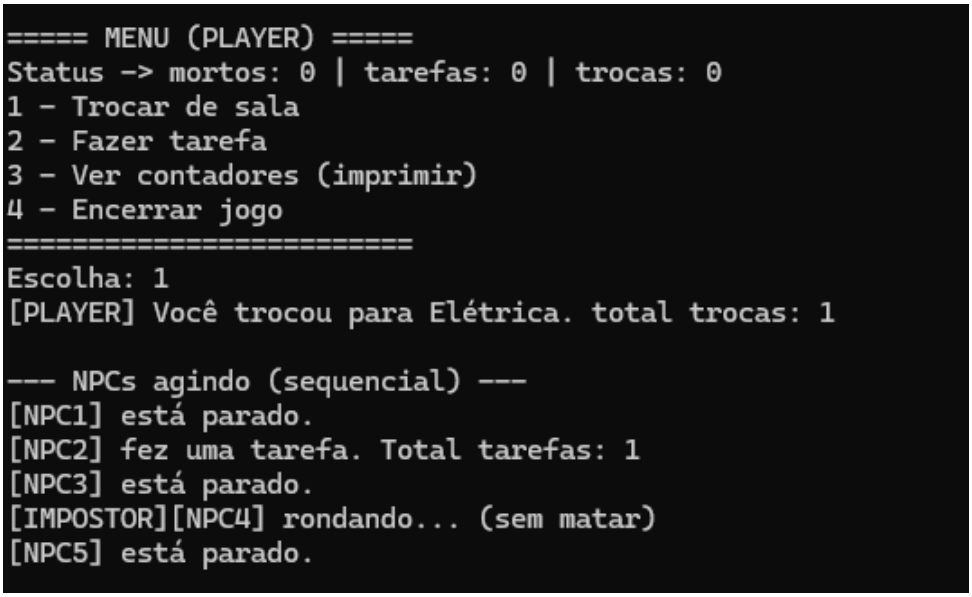
Para padronizar os testes, utilizamos vetores de tamanho fixo e definimos previamente o número de threads. As partições foram distribuídas de forma equilibrada, garantindo que cada thread realizasse aproximadamente a mesma quantidade de trabalho. Além disso, registramos o tempo de execução das duas versões para fins de comparação.

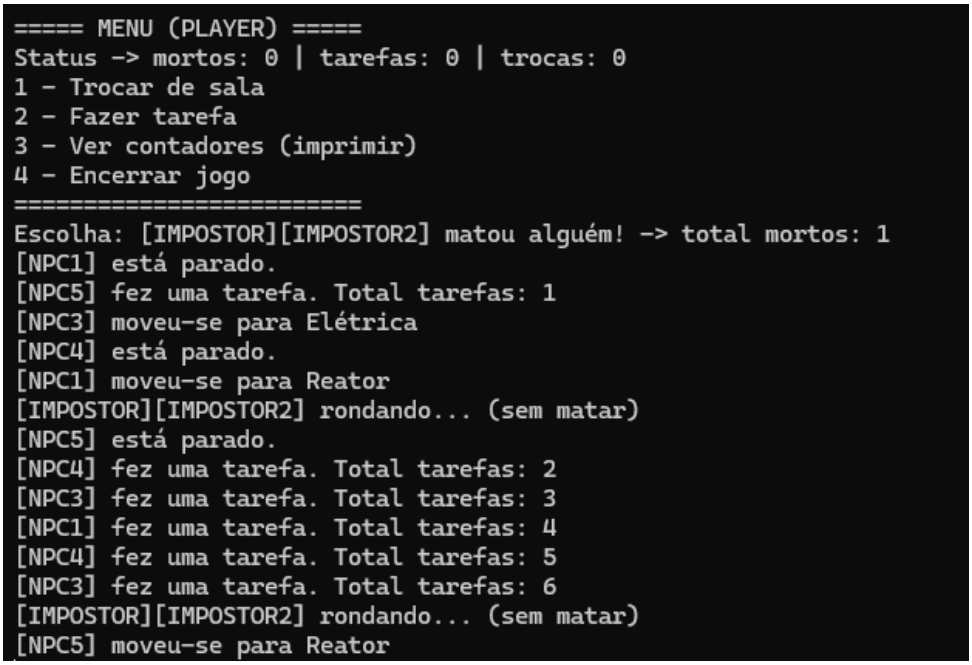
4.3. Resultados e Análise

Os testes mostraram que, para vetores pequenos, o uso de threads não gera ganho de desempenho relevante, pois o custo de criação e sincronização das threads se torna significativo. Entretanto, para vetores maiores, a versão paralela apresentou tempos menores de execução, especialmente em máquinas com vários núcleos de processamento, evidenciando o benefício do paralelismo em tarefas computacionalmente intensivas.

A escolha da temática inspirada em *Among Us* contribuiu positivamente para o entendimento do problema, permitindo relacionar cada thread a um agente executando uma tarefa específica. Isso facilitou a compreensão da diferença entre execução sequencial e paralela, além de tornar o experimento mais intuitivo e didático.

4.4. Imagens da Execução





4.5. Conclusão

A aplicação demonstrou claramente os benefícios do uso de threads em tarefas divididas em partes independentes. Embora o paralelismo não traga vantagens para entradas pequenas, ele se mostra eficaz conforme o volume de dados aumenta. A simulação com o tema *Among Us* facilitou a compreensão do comportamento paralelo, tornando o processo mais acessível e ilustrativo.