



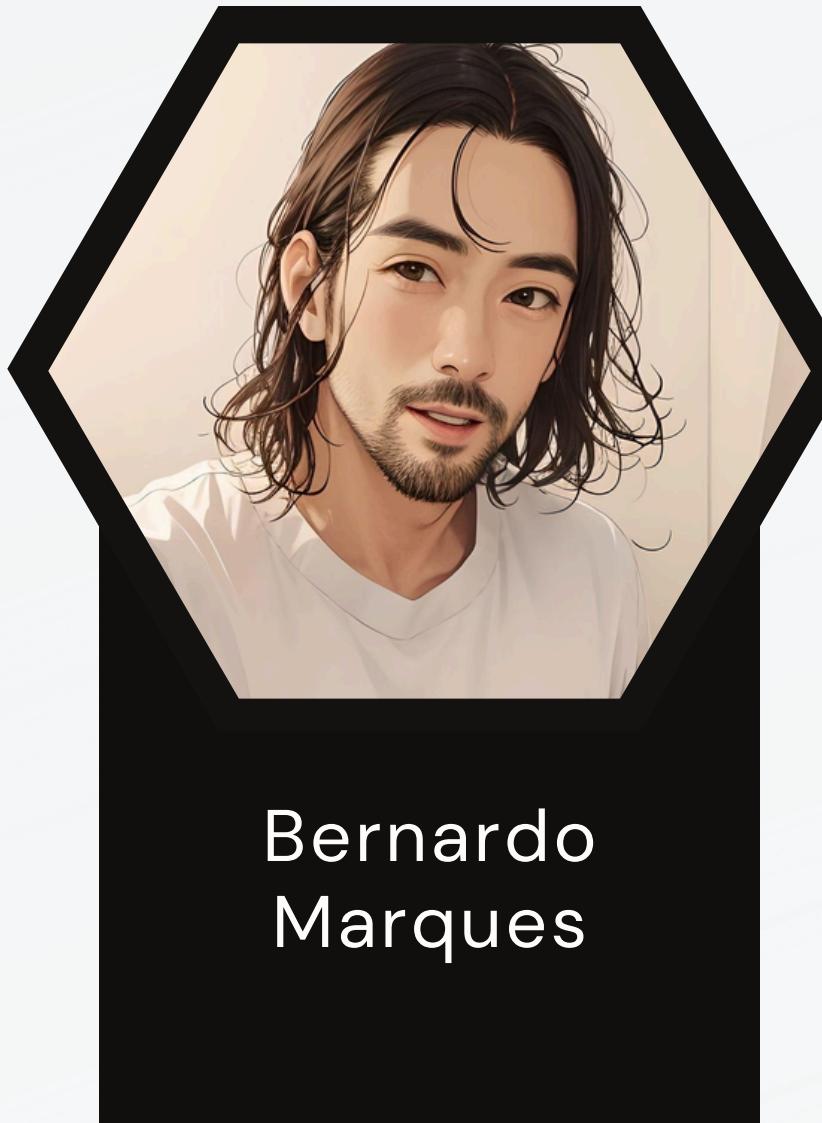
PROJETO 2

PARALELIZANDO

UMA IA



OUR TEAM



CONTENT

- 01** RANDOM FOREST
- 02** MÁQUINA USADA
- 03** OPENMP CPU
- 04** OPENMP CPU – SPEEDUPS
- 05** OPENMP GPU
- 06** OPENMP GPU – SPEEDUPS
- 07** CUDA
- 08** CUDA – SPEEDUPS
- 09** CONSIDERAÇÕES

RANDOM FOREST

mais que um Random Forest, uma Floresta aleatória

RANDOM FOREST

Cart tree

Árvore de decisão que usa o coeficiente Gini para determinar a melhor característica para a divisão da árvore

Bootstrap Sample

Para cada árvore, gera-se uma subpartição das instâncias de treino para que nenhuma árvore tenha 100% a mesma estrutura

Classificação

O resultado final da predição de cada instância de teste é tirado a partir da votação das previsões locais de cada árvore da floresta



MÁQUINA USADA

Sistema operacional: linux mint 21.1

CPU: QUAD CORE MODEL-intel i5 - 11300H

GPU: RTX 3050 4GB

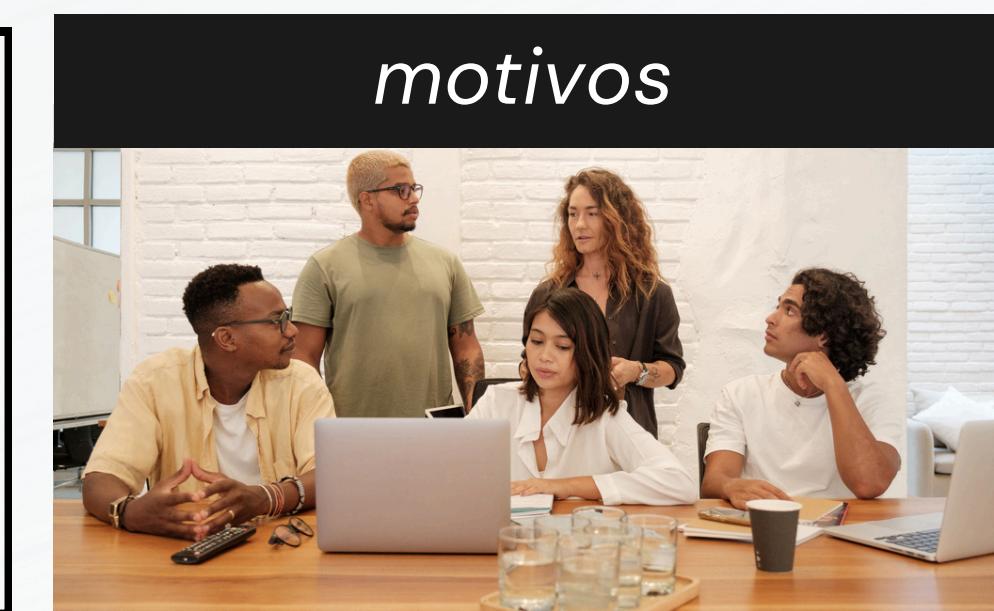
Memória RAM: 16GB

OPENMP CPU



- Para a versão do código em paralelo com OPENMP GPU utilizamos
- `#pragma omp parallel for schedule(dynamic)`
- `#pragma omp critical`

- Ao parallelizar a função `best_threshold` da classe CART, temos que a criação de cada árvore terá seu tempo de execução reduzido, pois a busca pela melhor regra demanda bastante tempo de execução
- A seção crítica garante que as variáveis `lowest_impurity`, `best_feature` e `best_threshold` não sejam acessadas por mais de uma thread ao mesmo tempo



SPEEDUP

OPENMP - 1 Thread

Sequential - 224,181s

OpenMP - 223,253s

1,0042

SPEEDUP

OPENMP - 2 threads

Sequential - 224,181s

OpenMP - 125,274s

1,7895

SPEEDUP

OPENMP - 4 threads

Sequencial - 224,181s

OpenMP - 76,150s

2,9439

SPEEDUP

OPENMP - 8 threads

Sequential - 224,181s

OpenMP - 71,444s

3,1378

SPEEDUP

OPENMP - 16 threads

Sequential - 224,181s

OpenMP - 63,981s

3,5039

SPEEDUP

OPENMP - 32 threads

Sequential - 224,181s

OpenMP - 64,503s

3,4755

SPEEDUP

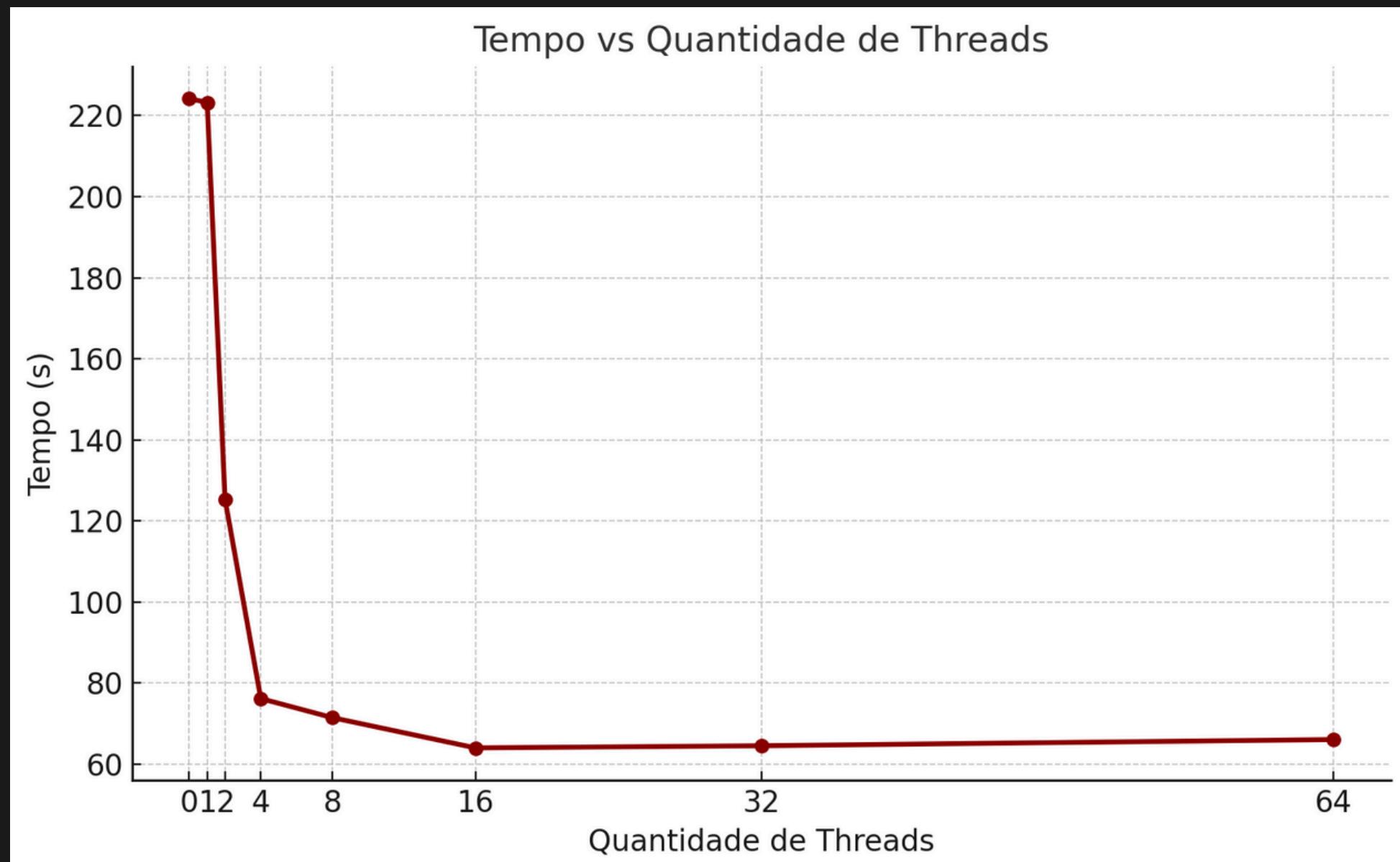
OPENMP - 64 threads

Sequential - 224,181s

OpenMP - 66,039s

3,3957

OPENMP CPU - GRÁFICO



OPENMP GPU



Random Forest - Fit

- Para a versão do código em paralelo com OPENMP GPU utilizamos
- `#pragma omp target teams distribute parallel for num_teams(x) schedule(dynamic)`

- Ao paralelizar a função fit, nós fazemos que cada árvore seja treinada em paralelo do inicio ao fim, garantindo assim, uma eficiência maior ainda.

motivos



SPEEDUP

OPENMP GPU - num threads 1

Sequential - 224,181s

OpenMP GPU - 76,684s

2,923

SPEEDUP

OPENMP GPU - num threads 2

Sequential - 224,181s

OpenMP GPU - 73,043s

3,069

SPEEDUP

OPENMP GPU - num threads 4

Sequential - 224,181s

OpenMP GPU - 73,461s

3,052

SPEEDUP

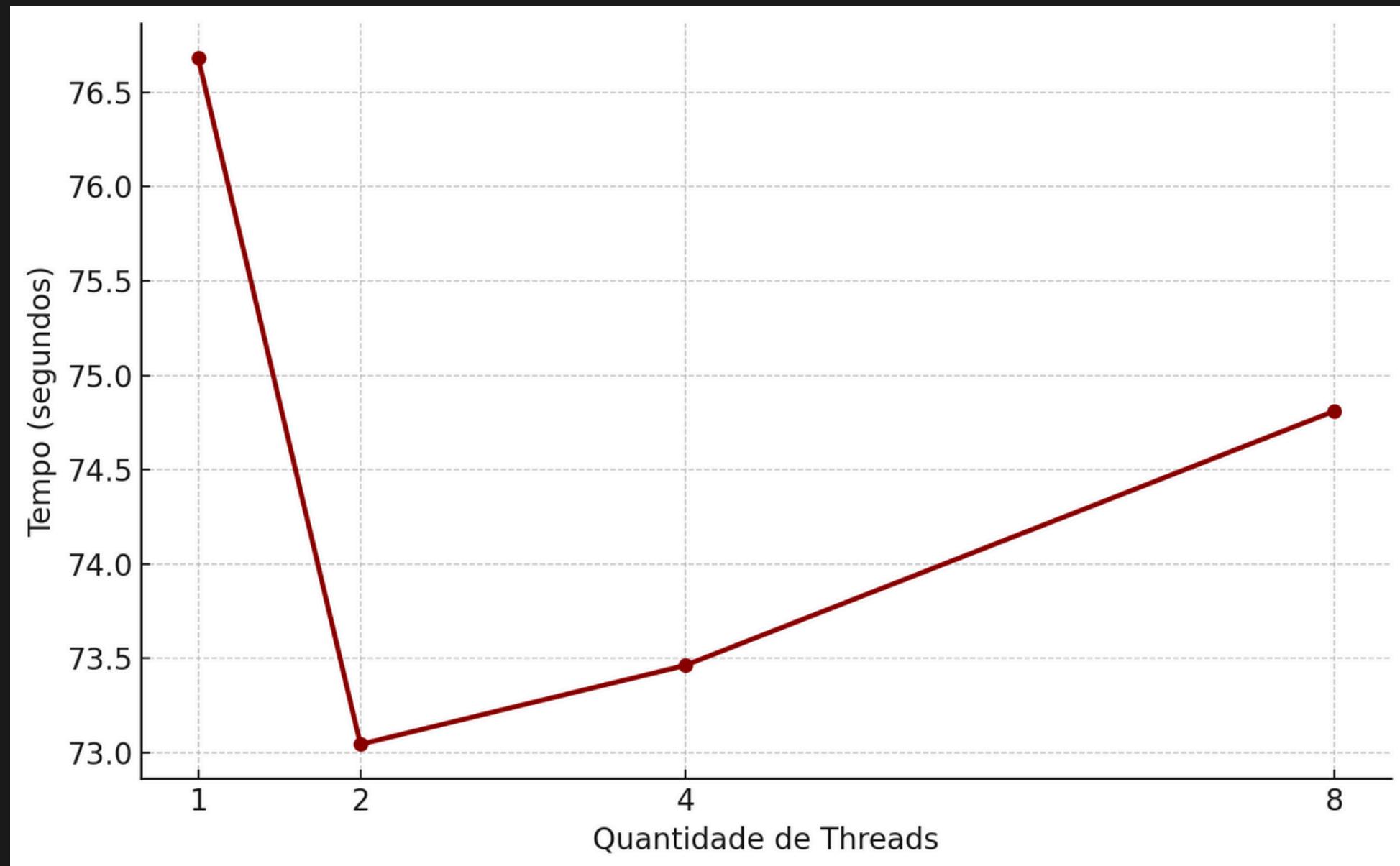
OPENMP GPU - num threads 8

Sequential - 224,181s

OpenMP GPU - 74,810s

2,997

OPENMP GPU - GRÁFICO

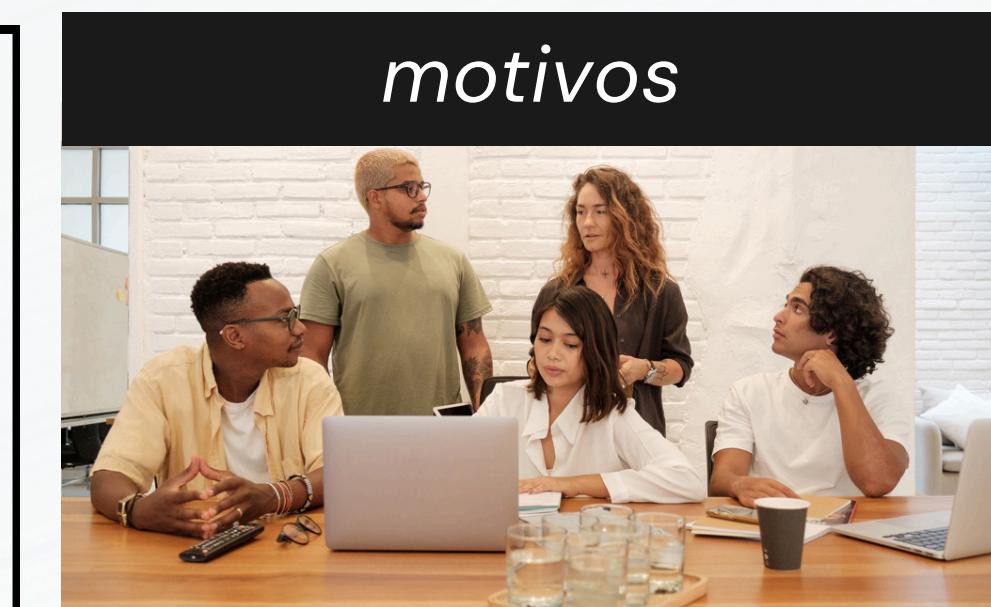


CUDA



- O código inteiro da função foi reescrito em CUDA
- As tabelas de features (X) e targets (y) são enviadas para a GPU

- Ao paralelizar a função `best_threshold` da classe CART, temos que a criação de cada árvore terá seu tempo de execução reduzido, pois a busca pela melhor regra demanda bastante tempo de execução.



SPEEDUP

CUDA

Threads por bloco - 32

Sequencial - 224,181s

CUDA - 0,406s

552,1699

SPEEDUP

CUDA

Threads por bloco - 64

Sequencial - 224,181s

CUDA - 0,376s

596,2261

SPEEDUP

CUDA

Threads por bloco - 128

Sequencial - 224,181s

CUDA - 0,321s

698,3832

SPEEDUP

CUDA

Threads por bloco - 256

Sequencial - 224,181s

CUDA - 0,297s

754,8181

SPEEDUP

CUDA

Threads por bloco - 512

Sequencial - 224,181s

CUDA - 0,306s

732,6176

SPEEDUP

CUDA

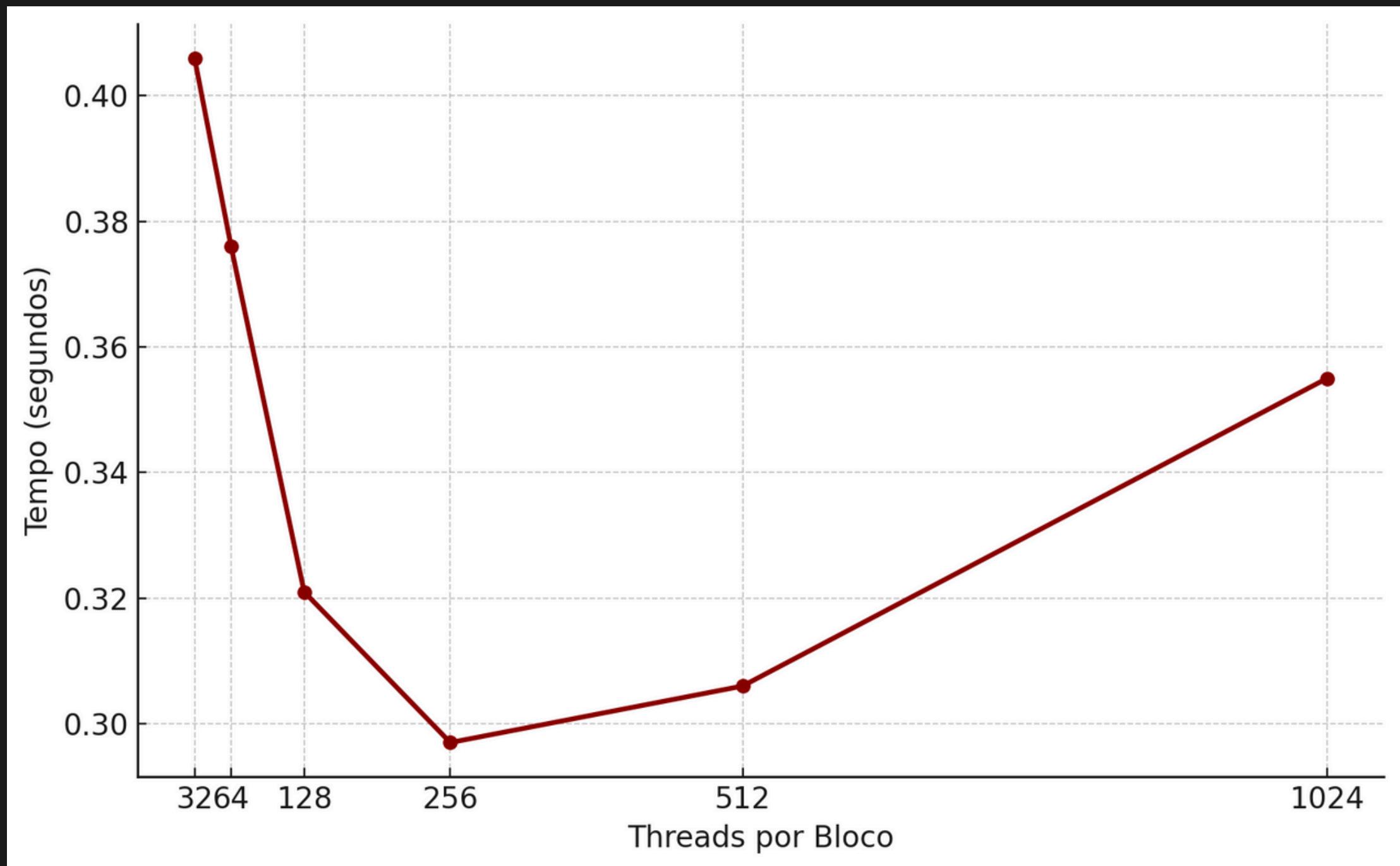
Threads por bloco - 1024

Sequencial - 224,181s

CUDA - 0,355s

631,4958

CUDA - GRÁFICO



CONSIDERAÇÕES

Melhora considerável mas a escalabilidade continua praticamente estática.

OPENMP

Melhora drástica em comparação ao OpenMP.
Provavelmente boa parte do tempo é overhead de envio de dados a GPU.

CUDA

Provavelmente se terá muito mais speedup em casos onde as tabelas sejam maiores, e perda se aumentar o número de árvores.

CONSIDERAÇÕES

**THANK'S FOR
WATCHING**

