

Implementação Paralela de uma Árvore de Decisão (MPI/OpenMP)

Ana Fernanda Souza Cancado
Arthur de Sá Braz de Matos
Gabriel Praes Bernardes Nunes
Guilherme Otávio de Oliveira
Júlia Pinheiro Roque

ALGORITMO E BASE

Decision Tree Algorithm

<https://github.com/bowbowbow/DecisionTree>

→ Alterações para receber base CSV e dividir entre treino e teste automaticamente

Base

[US 2023 Civil Flights, delays, meteo and aircrafts](#)

→ Cancelled_Diverted_2023.csv

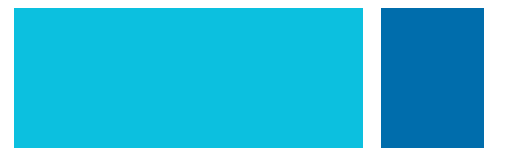
→ Prever se um voo vai ser cancelado ou desviado





LINK REPOSITÓRIO DO GRUPO

<https://github.com/arthursbmatos9/Computacao-Paralela>





COMO RODAR


Sequencial

- `g++ -std=c++11 -o dt decision_tree.cpp`
- `./dt CANCELLED_DIVERTED_2023.csv Cancelled`

OpenMP

- `g++ -std=c++11 -fopenmp -o dt decision_tree_omp.cpp`
- `./dt CANCELLED_DIVERTED_2023.csv Cancelled`

MPI/OMP

- `mpic++ -fopenmp decision_tree_MPI.cpp -o dt_parallel -O3`
 - `export OMP_NUM_THREADS=X`
`mpirun -np Y /dt_parallel CANCELLED_DIVERTED_2023.csv Cancelled`
- 

OPEN MP

- No código como um todo, 4 “for” puderam ser paralelizados

```
// PARALELIZAÇÃO: Cálculo de gain ratio para cada atributo
#pragma omp parallel for schedule(dynamic)
for(int i=0; i < numAttrs; i++) {
    gainRatios[i] = getGainRatio(table, i);
}
```

schedule(dynamic)
Teve melhor desempenho

OPEN MP

- Extração de valores de atributos

```
// PARALELIZAÇÃO: Cada atributo escreve em seu próprio attrValueList[j]
#pragma omp parallel for schedule(dynamic)
for(int j=0; j<attrName.size(); j++) {
    map<string, int> value;
    for(int i=0; i<data.size(); i++) {
        if(j < data[i].size()) {
            value[data[i][j]]=1;
        }
    }

    for(auto iter=value.begin(); iter != value.end(); iter++) {
        attrValueList[j].push_back(iter->first);
    }
}
```

- Como os atributos são independentes, paralelizar essa etapa é uma vantagem
- Paralelizar o loop interno não seria vantajoso, visto que cada thread escreveria no mesmo map value

OPEN MP

```
// PARALELIZAÇÃO: Predições em paralelo com redução
```

```
#pragma omp parallel for reduction(+:hitCount)
for(int i = 0; i < testTable.data.size(); i++) {
    vector<string> testRow = testTable.data[i];
    string actualLabel = testRow.back();
```

```
    testRow.pop_back();
    string predictedLabel = dt.guess(testRow);
```

```
// PARALELIZAÇÃO: Área crítica para atualizar contadores compartilhados
```

```
#pragma omp critical
{
    classCount[actualLabel]++;
    if(actualLabel == predictedLabel) {
        correctCount[actualLabel]++;
    }
}
```

```
if(actualLabel == predictedLabel) {
    hitCount++;
}
```

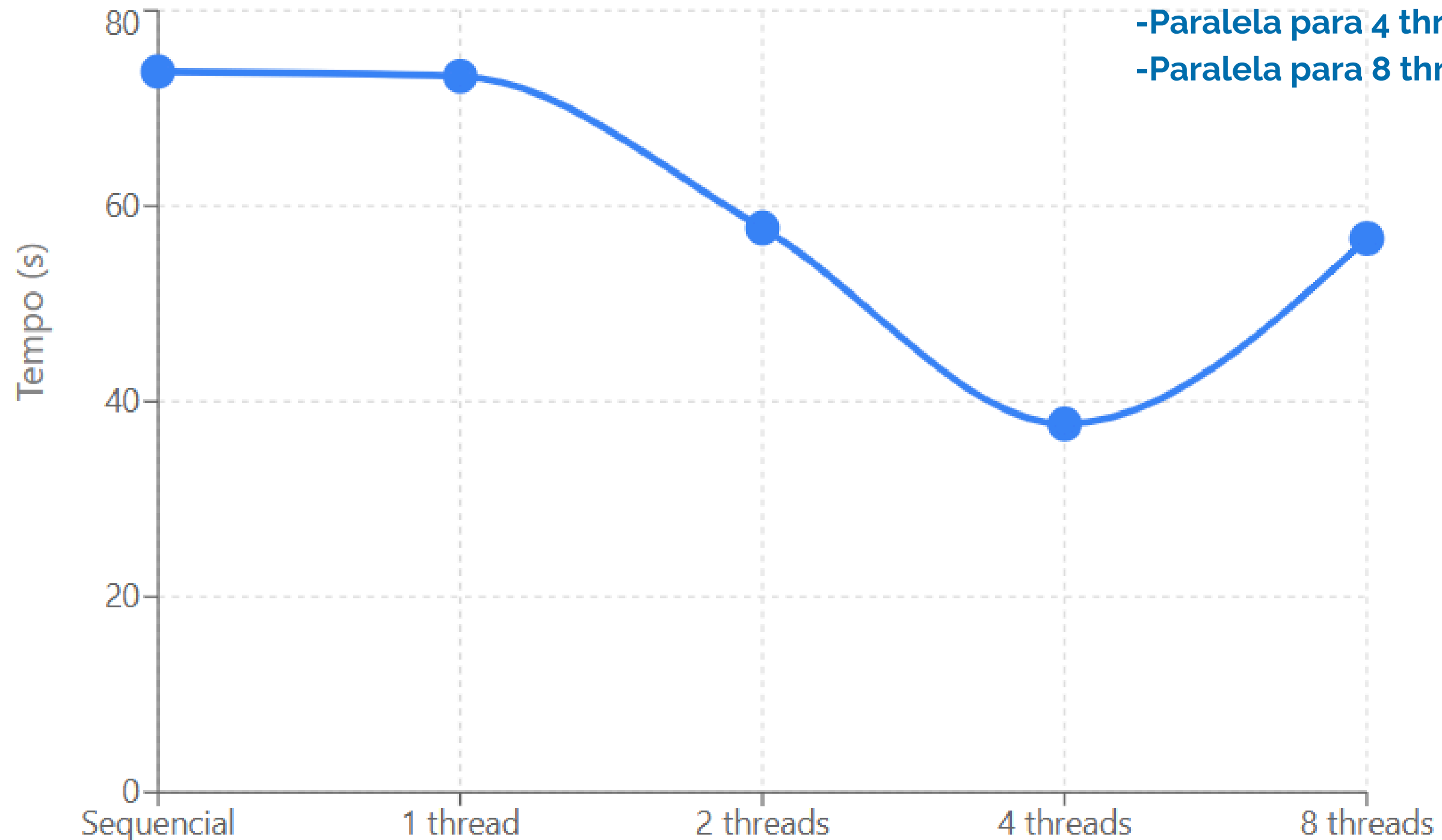
```
}
```

→ Usa **reduction** para acumulação thread-safe e **#pragma omp critical** para atualizar contadores compartilhados



OPEN MP


- Sequencial: 73.79 s
- Paralela para 1 threads: 73.31 s
- Paralela para 2 threads: 57.76 s
- Paralela para 4 threads: 37.70 s
- Paralela para 8 threads: 56.70 s





MPI

Passo	Descrição	Código Principal
1	Processo 0 lê o arquivo CSV completo	<code>if(rank == 0) { InputReader... }</code>
2	Broadcast distribui dados para todos os processos	<code>MPI_Bcast(...)</code>
3	Divisão do dataset de treino em chunks	<code>start = rank * chunkSize</code>
4	Cada processo treina sua própria árvore	<code>DecisionTree decisionTree(trainTable)</code>
5	Todos avaliam independentemente	<code>AccuracyCalculator::calculate(...)</code>



MPI

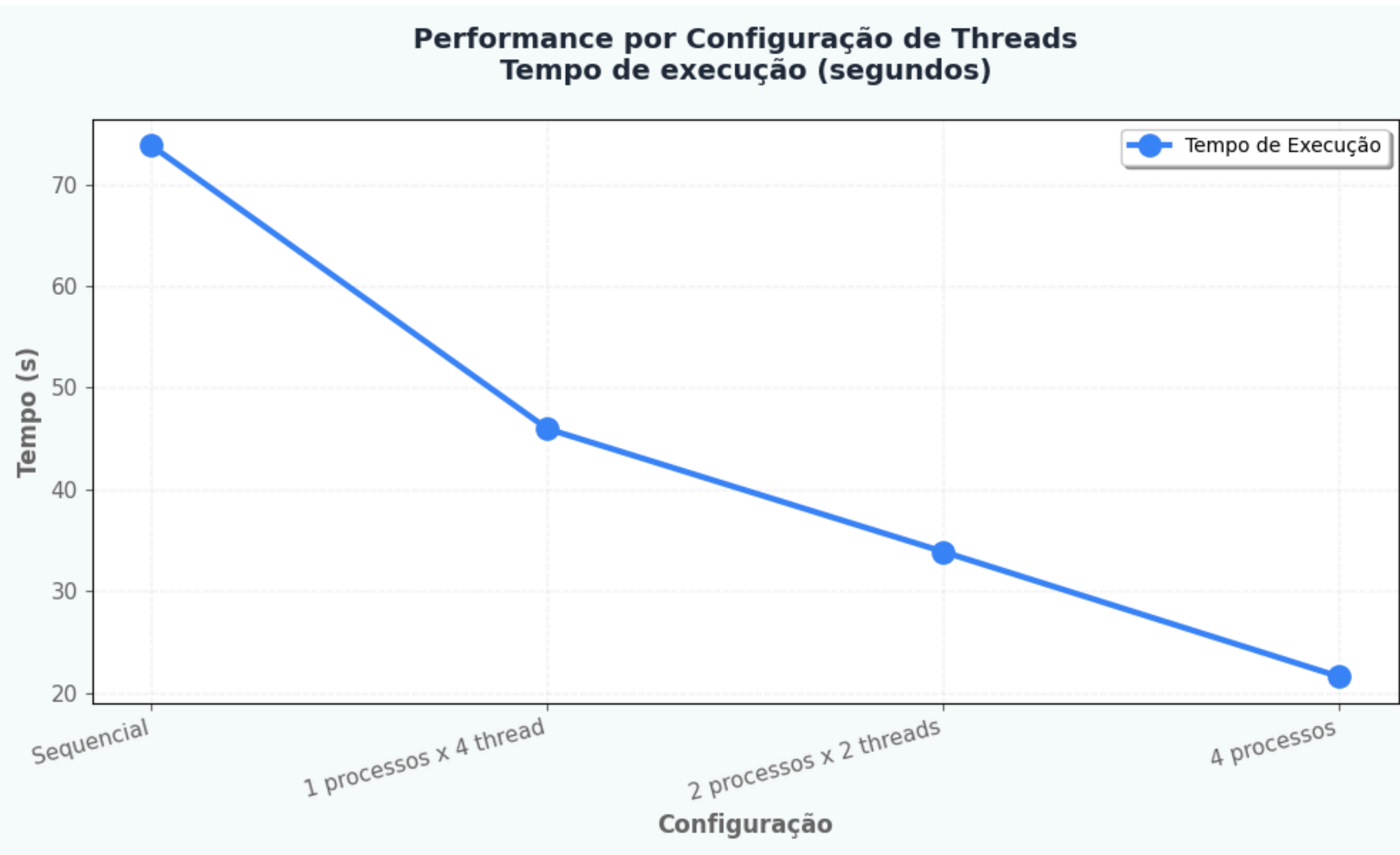
Exemplos class DataSplitter

```
// Apenas o processo 0 faz o shuffle
if(rank==0) {
    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
    shuffle(indices.begin(), indices.end(), default_random_engine(seed));
}

// COMUNICAÇÃO MPI: Broadcast dos índices embaralhados
MPI_Bcast(indices.data(), indices.size(), MPI_INT, 0, MPI_COMM_WORLD);

// Dividir dados de treino entre processos
int chunkSize = trainSize / size;
int start = rank * chunkSize;
```


MPI





MPI

Por que 4 processos entregam o menor tempo?

- **INDEPENDÊNCIA vs. COMPARTILHAMENTO:**
 - MPI: Cada processo tem memória própria, trabalha independente;
 - OpenMP: Threads compartilham memória, precisam coordenar acessos;
 - **PARALELISMO REAL:**
 - MPI: Cada processo em núcleo separado com cache próprio
 - OpenMP: Threads competem por recursos e cache (false sharing)
 - **SINCRONIZAÇÃO:**
 - MPI: Sincronização rara, processos autônomos
 - OpenMP: Muitos pontos de espera (#pragma omp critical)
- 



OBRIGADO!