Protótipo e Monitoramento

Daniel Netto, Erick Leão, Felipe Bruno, João Gabriel, João Matheus, Levy Souto
{danielcieplinski,erick.santos,fbsilva,joao.souza,joao.matheus,levys}@unisantos.br
Universidade Católica de Santos (Unisantos)

1. INTRODUÇÃO

Analisar e processar grandes conjuntos de dados envolve lidar com uma série de desafios [1], incluindo problemas como: esgotamento de memória, custos significativos de hardware, longos tempos de execução, risco de perda de dados e a complexidade associada ao gerenciamento de memória. Neste projeto, nosso foco será manipulação de um conjunto de dados volumoso através de técnicas de programação, visando a categorização dos dados.

Dando sequência ao progresso feito na Análise de Performance e Proposta de Solução, esta fase engloba o desenvolvimento de um protótipo através da implementação de programação paralela, fundamentado no levantamento da etapa anterior. Além disso, iremos abordar o monitoramento do desempenho e a comparação entre o desempenho com e sem a utilização do protótipo.

2. CONJUNTO DE DADOS

Para a realização do protótipo, foi disponibilizado um conjunto de dados de operações portuárias do Brasil, tais como eventos de atracações de embarcações e movimentação de cargas, informações de localização, identificações de carga, datas e horários, tipos de operação, entre outros.

O objetivo a ser atingido é a categorização de valores nominais, substituindo-os por valores numéricos, após isso serão criados arquivos csv individuais de cada coluna que foi fatorizada, e por fim a jução de todas as colunas fatorizadas em um dataset unificado.

É arquivo de tipo csv, com um conjunto volumoso de dados, contando com 26 colunas e mais de 14 milhões de linhas, ocupando um total de 2,86 GB de armazenamento. Um arquivo de mil linhas com a mesma estrutura foi disponibilizado para testes. O objetivo é categorizar valores nominais, substituindo por números inteiros.

Para facilitar a captura de dados no arquivo, foi feito upload deste no google drive, assim pudemos ter acesso aos dados de forma menos trabalhosa.

```
[2] dfmil.shape

(14665111, 26)

dfmil.head()
```

	idatracacao	cdtup	berco	portoatracacao	ano	mes	tipooperacao	tiponavegacaoatracacao	terminal	nacionalidadearmador	...	idcarga	origem	destino	cdmercadoria
0	788883	BRCDO	101	Cabedelo	2016	ago	Marinha	Apoio Portuário	Cais Público	0	...	17880751	BRIQI	BRCDO	2710
1	751889	BRBEL	BELM201	Belém	2016	mar	Movimentação da Carga	Cabotagem	Terminal de Miramar	2	...	16157250	BRSSA	BRBEL	2710
2	751889	BRBEL	BELM201	Belém	2016	mar	Movimentação da Carga	Cabotagem	Terminal de Miramar	2	...	16157251	BRBEL	BRBEL	CA01
3	797313	BRSE002	Quadro de Bóias	Terminal Aquaviário de Aracaju	2016	out	Movimentação da Carga	Cabotagem	Terminal Aquaviário de Aracaju	1	...	17416470	BRSE002	BRAM011	2709
4	794159	BRAM011	POF 3	Terminal Aquaviário de Manaus	2016	set	Movimentação da Carga	Cabotagem	Terminal Aquaviário de Manaus	1	...	17344434	BRFOR	BRAM011	2709

Figura 1 - Estrutura do arquivo

3. FERRAMENTAS E AMBIENTE DE EXECUÇÃO

Utilizamos a plataforma Google Collaboratory[2] para desenvolvimento dos protótipos, pois oferece serviço de ambiente Jupyter notebook hospedado em nuvem, assim podemos utilizar recursos computacionais da Google de forma colaborativa.

Sendo um serviço em nuvem, os recursos são limitados, e podemos apenas monitorar o desempenho da memória e do disco durante a execução, não podemos monitorar o uso da cpu.

Recursos do colab versão gratuita:
CPUs: Por padrão, cada notebook é executado em uma máquina virtual com CPU, para tarefas básicas e processamento de dados.

Processador: Intel(R) Xeon(R) CPU @ 2.20GHz com 1 core
Memória: aproximadamente 12,67 GB por ambiente de execução.
Disco: aproximadamente 107,72 GB de armazenamento.
2 Threads para execução de tarefas.

Escolhemos C++ [3]como a linguagem de programação a ser empregada devido ao seu controle de baixo nível, biblioteca para manipular arquivos, suporte à multithreading e biblioteca omp.h para OpenMP [4].

4. PROTÓTIPO E COMPARAÇÃO DE DESEMPENHO

4.1 Programação Sequencial e Processamento em Lotes

Primeiramente, foi realizada a tentativa de ler o arquivo massivo pela programação sequêncial, iterando sobre todas as linhas de uma vez. Inicialmente o cabeçalho com os nomes das colunas é salvo para posteriormente ser adicionado, em seguida é verificado se os valores das colunas não são numéricos, por fim os valores nominais são fatorizados e salvos em uma estrutura de dados conhecida

como mapas não ordenados (hash)[5], dessa maneira é feita a categorização dos dados, ligando seu valor nominal ao valor fatorizado.

As próximas técnicas de programação utilização a mesma lógica de fatorização explicada no parágrafo anterior, porém aplicando suas particularidades.

Pela programação sequencial, devido ao número massivo de infromação no arquivo, há um aumento no uso de memória até sua capacidade máxima, fazendo com que haja a desconexão do ambiente de execução, impossibilitando a visualização de qualquer resultado. Como mostrado na figura 2

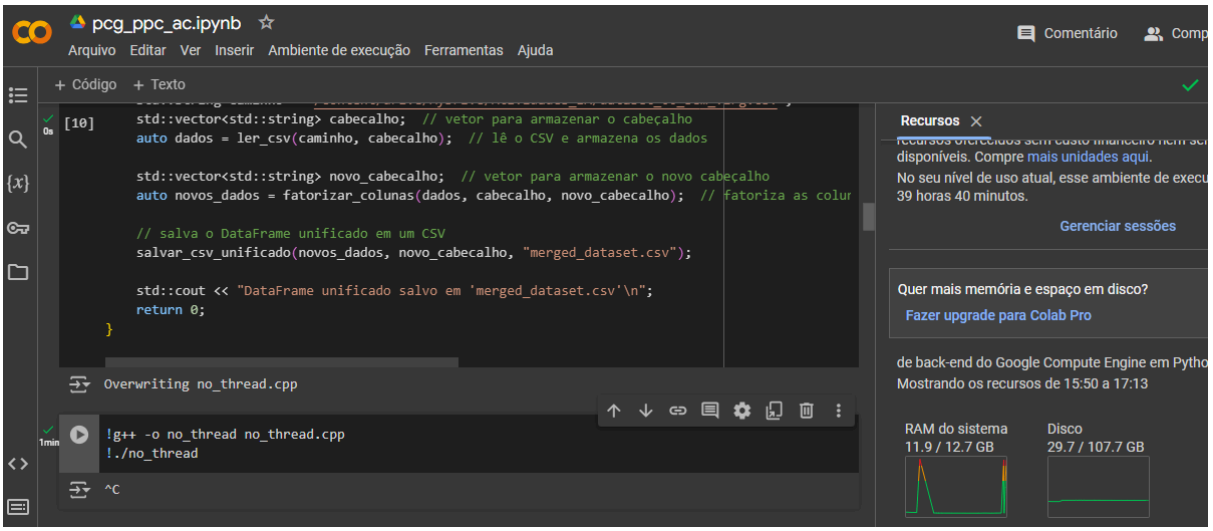

Figura 2 - Execução do código sequêncial

Como mostrado, o uso de aproximadamente 12GB de memória torna inviável a execução do código para processamento de arquivo de maneira sequencial.

Para teste da funcionalidade do código, utilizamos a programação em lotes[6], processando e fatorizando apenas um lote de 10 mil linhas do arquivo, o qual resultou em um processamento rápido e sem gasto significativo de memória e armazenamento.

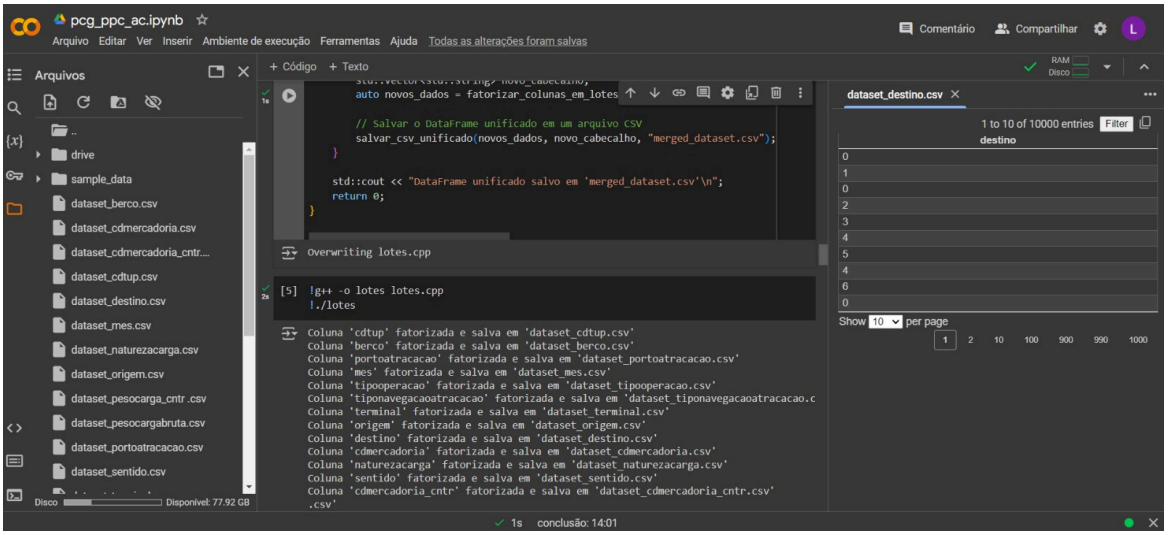

Figura 3 - Execução de um lote de 10 mil linhas

4.2 Programação paralela: Threads e OpenMP

Em seguida, utilizamos técnicas de programação paralela para maior impacto no desempenho da leitura e processamento do conjunto de dados. Para uma leitura mais eficiente, utilizamos lotes de 10 mil linhas, como já explicado anteriormente, para que o processo de paralelização e gerenciamento de memória seja facilitado.

São criadas o máximo de threads(2 no colab) para que a *função fatoriza_thread()*, responsável por fatorizar as colunas seja processada de maneira paralela, de modo que 2 lotes sejam lidos ao mesmo tempo.

Para proteção de recursos compartilhados, no caso a escrita no arquivo csv para mapeamento de valores, std:: mutex[7] é usado para garantir a integridade desses recursos, std::lock_guard<std::mutex> bloqueia o mutex no início da escrita e o libera no final, protegendo a seção crítica.

E ao fim o método *fatoriza_thread.join()* garante que a thread que fatora as colunas seja concluída antes de prosseguir com a próxima iteração, sincronizando a operação com o fluxo do programa.

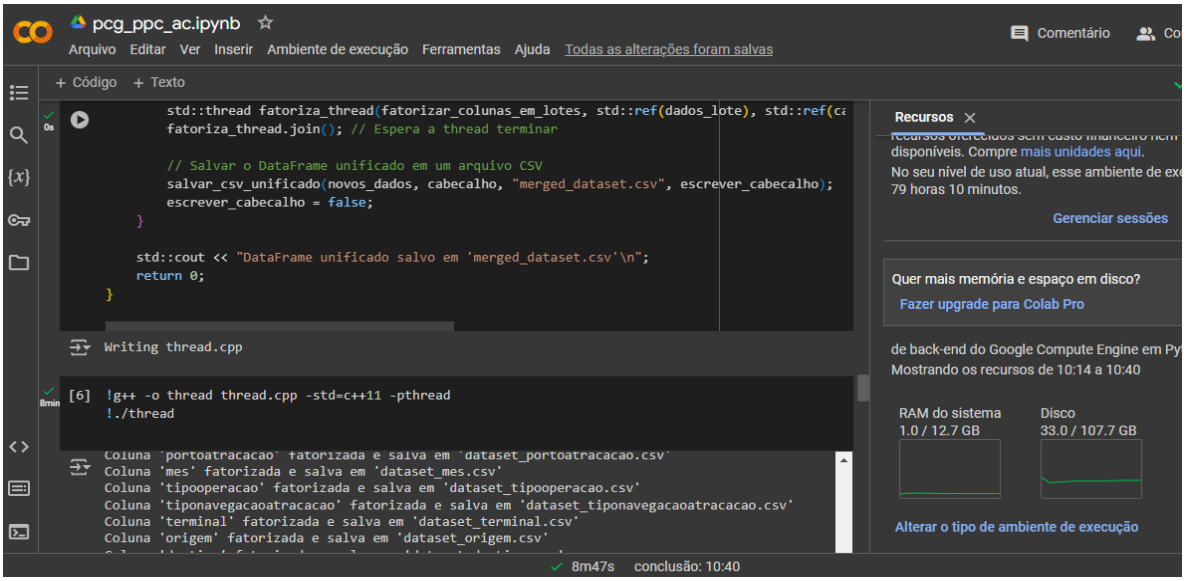

Figura 4 - Execução com threads

Agora empregando o modelo Open MP, foi utilizada a biblioteca std <omp.h> que permite incorporar diretivas para compilação[8], como *#pragma omp paralel for*, que foi implementada antes do loop criado pela função fatorizar_colunas_em lotes, distribuindo a iteração sobre as colunas entre as 2 threads disponíveis.

Como dito anteriormente, no código há uma seção crítica no código, no caso o mapeamento de valores no arquivo, tornando assim um recurso compartilhado que pode ser escrito pelas 2 threads. Para garantir a segurança da seção crítica, usamos a diretiva *# pragma omp critical,* dessa maneira garantindo que apenas uma thread escreva por vez, evitando condição de corrida.

Lembrando que em outro ambiente de execução com mais threads disponíveis, poderíamos controlar o número de threads para paralelizar o loop por váriavel de ambiente. No caso, utilizamos o máximo de threads do google colab, 2 threads, e preparamos o ambiente para execução paralela.

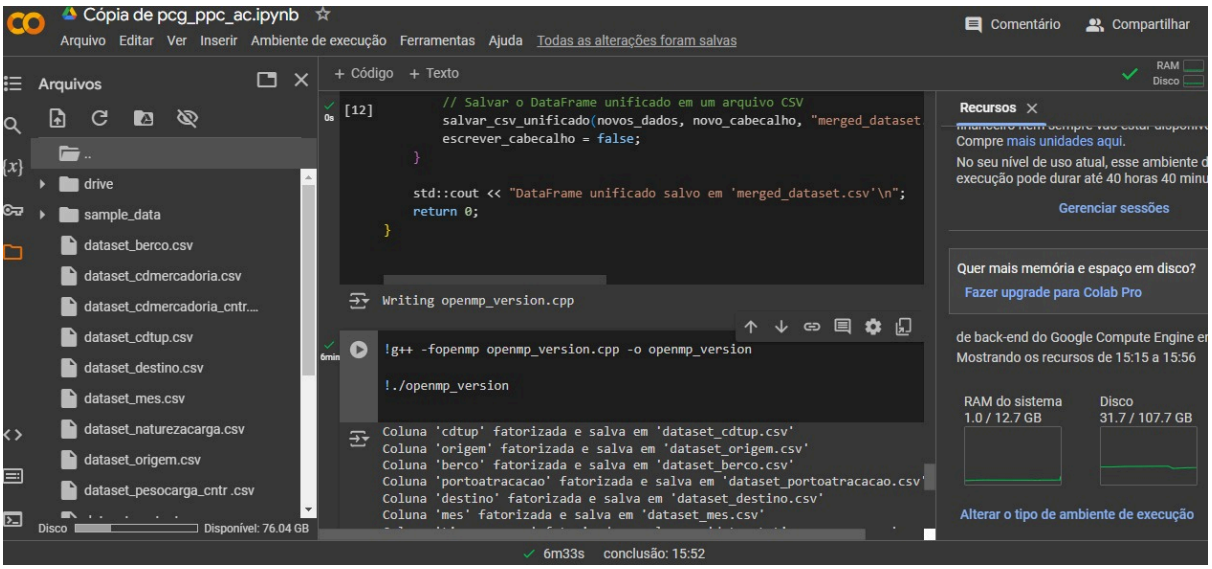

Figura 5 - Execução em OpenMP

4.3 Comparação de Desempenho e Uso do Hardware

Agora iremos realizar uma comparação de desempenho das diferentes técnicas para o processamento do conjunto de dados.

Programação sequencial: Como visto, utilizando a programação sequencial, há um aumento exarcebado no uso da memória, quase ultrapassando a totalidade de 12GB disponibilizada pelo ambiente do Google Colab devido ao alto número de informação no arquivo, dessa maneira levando ao estouro de memória antes da conclusão da execução.

Testamos o desempenho para ler e processa um lote de 10 mil linhas do arquivo, o qual levou por volta de 2 segundos para a conclusão, realizando de forma correta a categorização dos valores nominais, criação dos arquivos por coluna e a unificação dos datasets.

Já entre as técnicas de programação paralela, Open MP se mostrou um pouco mais eficiente em questão de desempenho e tempo de execução, devido à automatização da sincronização por parte de suas diretivas.

As threads convencionais com mutex, levaram entre 7 e 10 minutos para processar todo o dataset em diversos testes realizados, utilizando pouco mais de 1GB de memória e 31.7 de disco.

Já com Open MP, utilizando diretivas para paralelizar a loops e proteção da seção crítica, o processamento do dataset completo levou entre 6 a 8 minutos.

5. DIFICULDADES ENCONTRADAS

Durante a realização do trabalho, encontramos diversas dificuldades, a maioria devido ao alto número de informação do dataset, com tanto volume de dados, a manipulação do arquivo csv tornou-se dificultosa, iniciando pela visualização de seu conteúdo, não sendo possível abrir a planilha pelo Excel além da demora pelo download.

Para resolver esse problema, fizemos o upload no drive, dessa maneira utilizamos recursos de código no colab como drive.mount para ter acesso ao google drive e então ver a estrutura do arquivo com a biblioteca pandas do Python. Após verificar o arquivo, iniciamos a leitura e processamento com C++.

Outra dificuldade foi o tempo para leitura do arquivo, na tentativa de ler a planilha inteira de uma vez, além do longo tempo de execução, houve um grande uso da memória até sua capacidade máxima, como dito anteriormente. Para resolvermos esse problema utilizamos a leitura em lotes, o qual por coluna, lia um lote de 10 mil linhas por vez.

Por fim, depois do código já executado, a visualização dos arquivos gerados, tanto os csvs individuais quanto o unificado, não puderam ser visualizados pelo colab devido ao alto número de linhas, levando o google colab ao estouro de memória e fim do ambiente de execução, como mostrado abaixo na figura 6.

Para visualizar os dados categorizados gerados, fizemos downloads dos datasets gerados e fizemos upload no google drive para a visualização, durante os testes interrompemos a execução em no máximo 5 minutos para que pudéssemos ver a fatorização no colab, a qual se deu de forma correta.

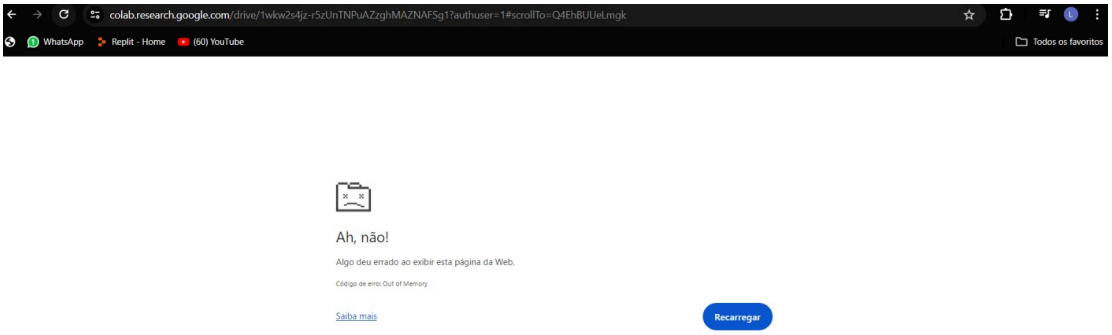

Figura 6 - Estouro de memória

6. CONSIDERAÇÕES FINAIS

Foi possível perceber a importância do desenvolvimento do protótipo e do monitoramento de dados como uma ferramenta fundamental para lidar com grandes conjuntos de dados. Os resultados obtidos demonstraram a eficácia das técnicas de programação paralela na categorização de valores nominais, proporcionando uma análise mais eficiente e rápida dos dados.

Os desafios enfrentados, como o uso excessivo de memória, mostram a necessidade de explorar abordagens alternativas à programação sequencial, como o processamento em lotes e emprego da programação paralela, para otimizar o desempenho e evitar o estouro de memória.

Vimos a relevância de dar continuídade aos estudos e aprimoramentos no protótipo, visando aperfeiçoar a manipulação de grandes conjuntos de dados e a aplicação de técnicas de programação paralela em futuros projetos de análise de dados.

7. REFERÊNCIAS BIBLIOGRÁFICAS

[1] **O que é Big Data?** Disponível em: <https://www.oracle.com/br/big-data/what-is-big-data/>. Acesso em: maio de 2024.

[2] **Google colab**. Disponível em: <https://research.google.com/colaboratory/faq.html>. Acesso em: maio de 2024.

[3] FCUP, F. S. D. **Introdução ao OpenMP**. Disponível em: <https://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/intro_openmp-Fernando-Silva.pdf>. Acesso em: maio de 2024.

[4] CS107: **C++ programming**. Disponível em: <https://learn.saylor.org/course/view.php?id=65&gad_source=1&gclid=CjwKCAjwrIixBhBbEiwACEqDJbLx_0iWpWUlKfQIvi6NETA9dJtX7OJOqdcKbn1K7GvH71iHkBq7RxoC3NEQAvD_BwE>. Acesso em: maio de 2024.

[5] TYLERMSFT. **Classe unordered_map**. Disponível em: <https://learn.microsoft.com/pt-br/cpp/standard-library/unordered-map-class?view=msvc-170>. Acesso em: maio de 2024.

[6] **Processamento em lotes**. AWS Amazon, Disponível em: <https://aws.amazon.com/pt/what-is/batch-processing/>. Acesso em: maio de 2024.

[7] LEIVAS, F. **Métodos de sincronização de threads**. Disponível em: <https://www2.ufpel.edu.br/cic/2011/anais/pdf/CE/CE_00769.pdf>. Acesso em: 31 maio. 2024.

[8] SCHEPKE, C.; LUCCA, **Diretivas Paralelas de OpenMP**.SBC, 10 maio 2023. Disponível em: <https://books-sol.sbc.org.br/index.php/sbc/catalog/download/119/526/805?inline=1> DOI: ˜ 10.5753/sbc.11938.7 Acesso em: maio de 2024.