APROP - Entrega 2

trabalho realizado por

Miguel Oliveira nº1210445

Jorge Silva nº1210443

docente

Prof. Tiago Carvalho

22 de dezembro de 2022

1 Introdução

Este trabalho foi desenvolvido numa máquina virtual usando o VirtualBox, que é um software de virtualização, onde foi criada uma máquina virtual com um sistema operativo Ubuntu server de 64-bit. A nível de especificações, a mesma tem um processador AMD Ryzen 5 5600X, com 4 cores e 4Gb de RAM.

2 Alínea A (Rayon)

2.1 Rayon - Versão com Iteradores Paralelos (Linear)

Nº de vezes	**Tempo máximo**	**Tempo minimo**	**Média dos tempos**	**Desvio Padrão**
10	3.015006212	2.952538732	2.988595708	0.021904013

Tabela 1: Registo dos valores

2.2 Rayon - Versão com Iteradores Paralelos (Bloco)

Nº de vezes	**Tempo máximo**	**Tempo minimo**	**Média dos tempos**	**Desvio Padrão**
10	3.085610685	3.00329955	3.027643504	0.025051603

Tabela 2: Registo dos valores

2.3 Rayon - Versão com Scope (Linear)

Nº de vezes	**Tempo máximo**	**Tempo minimo**	**Média dos tempos**	**Desvio Padrão**
10	5.230441483	5.015715011	5.098552508	0.056016145

Tabela 3: Registo dos valores

2.4 Rayon - Versão com Scope (Bloco)

Nº de vezes	**Tempo máximo**	**Tempo minimo**	**Média dos tempos**	**Desvio Padrão**
10	4.21757184	4.080952961	4.166577105	0.040132856

Tabela 4: Registo dos valores

Analisando os valores, concluímos que a versão linear usando o **Rayon** e **iteradores paralelos** é mais rápida a executar, em média, comparando com as outras abordagens. Contudo, a versão em bloco usando o **Rayon** e **iteradores paralelos** difere muito pouco.

A versão com **scope** de tasks acaba por ser mais lenta, porque a criação das tasks requer mais trabalho para o CPU, e portanto o ganho de performance é inferior à versão com **iteradores paralelos**. Esta versão acaba por ser bastante semelhante à versão com **Threadpool** que será analisada posteriormente.

3 Alínea B (Comparação com Threadpool e Versão Sequencial)

3.1 Threadpool (Linear)

Nº de vezes	Tempo máximo	Tempo minimo	Média dos tempos	Desvio Padrão
10	5.216023652	4.916023942	5.033714458	0.091072689

Tabela 5: Registo dos valores

3.2 Threadpool (Bloco)

Nº de vezes	Tempo máximo	Tempo minimo	Média dos tempos	Desvio Padrão
10	3.704776145	3.51821074	3.60096892	0.044238999

Tabela 6: Registo dos valores

3.3 Sequencial

Nº de vezes	Tempo máximo	Tempo minimo	Média dos tempos	Desvio Padrão
10	5.050565762	4.762401543	4.929688833	0.075007958

Tabela 7: Registo dos valores

Analisando os valores, concluímos que a versão em bloco utilizando **Threadpool** possui melhor performance do que a versão linear. A versão linear acaba por ser ligeiramente mais lenta do que a versão sequencial, isto porque o ganho de performance obtido com a criação de tasks não supera o seu overhead.

4 Alínea C (Comparação com OpenMP)

Nº de vezes	**Tempo máximo**	**Tempo minimo**	**Média dos tempos**	**Desvio Padrão**
10	0.862485	0.777554	0.8096356	0.024525506

Tabela 8: Registo dos valores

A versão com **OpenMP** utiliza threads para paralelizar o trabalho por linhas. Analisando o resultado obtido, concluímos que esta versão é a que possui melhor performance, superando todas as versões do Rust, até mesmo as que dividem o trabalho em bloco.

5 Conclusões

Concluindo, a versão com melhor performance é a versão com **OpenMP**, isto porque possui melhor optimização no processamento de threads e atribuição de tarefas, em comparação com as bibliotecas **Rayon** e **Threadpool** do **Rust**.

Em relação às versões com **Rust**, é importante referir que o mecanismo utilizado para a partilha de dados foi o **mutex**, e portanto, reconhecemos que a performance obtida poderia ser melhor optimizada com a utilização de outros mecanismos como por exemplo **message passing**, mas como apenas conseguimos implementar este mecanismo na versão com **Threadpool**, e o melhoramento de performance não era significativo, acabamos por excluir esta versão.