Exercício 1 - Programação Concorrente

Guilherme Cappelli Bouzon de Amorim Cruz
DRE: 121170269

06/04/2025

1 Programas

1.1 gen.c : Gerando vetores aleatórios

Para criarmos objetos de teste e termos como comparar a performance e precisão das threads no exercício de calcular o produto interno entre dois vetores, primeiro precisamos gerar dois vetores aleatórios.

Nesse arquivo, os argumentos passados serão (i) a dimensão dos vetores a serem gerados e (ii) o nome do arquivo binário de saída, onde as informações serão escritas. Após gerarmos os vetores, podemos prosseguir e computar o produto interno deles de forma sequencial, e assim, no arquivo binário de saída serão escritos na seguinte ordem:

- long int dimensão
- double* vetor1
- double* vetor2
- double produto interno sequencial

Modo de Uso: gen.c

```
gcc -o gen gen.c

./gen <dimensao> <arquivosaida.bin>
```

1.2 prod_thread.c : Calculando o produto interno com threads

Já no prod_thread.c, precisamos obter os dados escritos no binário com o gen.c para assim prosseguirmos com o cálculo do produto interno com múltiplas threads e comparar os resultados.

Nesse programa, cada thread recebe uma região de índices dos vetores, determinada pelo seu id numérico, onde $0 \leq \text{id} < n\_threads$, passado como argumento no pthread_create(). Depois de balancearmos as cargas de trabalho das threads, elas calculam o produto interno parcial relacionada a essa região.

Com pthread_join() recolhemos os resultados parciais retornados enquanto aguardamos as threads finalizarem e, assim, somamos os resultados parciais em uma variável double representando o produto interno final.

Além disso, adicionei a flag -p no programa, para que os resultados parciais de cada thread sejam exibidos na tela. Por fim, o produto interno via threads e sequencial são comparados pela Variação Relativa e exibidos na tela.

Modo de Uso: prod_thread.c

```
gcc -o prod_thread prod_thread.c

./prod_thread -p <num_threads> <arquivo_entrada.bin>
```

1.3 gen_v.sh : Automatizando um pouco os testes

Para testarmos múltiplas execuções do cálculo do produto interno com diferentes tamanhos de vetores, utilizamos um script bash, onde são feitas 10 iterações executando ambos os programas gen.c e prod_thread.c passando a saída do primeiro como entrada do segundo. Além disso, o número de threads é definido proporcionalmente à dimensão dos vetores.

Vale ressaltar que todos os arquivos rand_i.bin, onde $1 \leq i \leq 10$ são salvos no diretório em que os programas se encontram (script incluso, que deve permanecer no mesmo diretório que ambos os programas em C).

Modo de Uso: prod_thread.c

```
chmod +x ./gen_v.sh
./gen_v.sh -p
```

OBS: Se não quiser o produto parcial das threads exibidos na tela, basta não enviar a flag.

2 Análise dos resultados

Se o script for executado sem a flag -p, obtemos a tabela de resultados abaixo. Perceba que a variação entre os resultados das versões sequencial e concorrente foi extremamente pequena em todos os casos sem exceção. Podemos verificar que as diferenças são quase irrelevantes do ponto de vista numérico, pois a Variação Relativa está na ordem de 10^{-15}.

Dimensão	#Threads	Prod. Interno (SEQ)	Prod. Interno (THREADS)	Variação Relativa
4252	212	1597.383692712526	1597.383692712534	5.12×10^{-15}
1266	63	-998.509230583293	-998.509230583293	1.14×10^{-16}
2278	113	1737.535487935386	1737.535487935389	2.09×10^{-15}
1168	58	-1127.528756367049	-1127.528756367049	0.00×10^{0}
2128	106	1108.296232558158	1108.296232558160	2.26×10^{-15}
3728	186	1444.599392197514	1444.599392197521	4.88×10^{-15}
1902	95	148.626223547118	148.626223547120	1.89×10^{-14}
2051	102	833.622334606110	833.622334606113	3.27×10^{-15}
4080	204	2309.261104313489	2309.261104313495	2.76×10^{-15}
564	28	-843.477012687720	-843.477012687720	0.00×10^{0}

Table 1: Comparação entre produto interno sequencial e paralelo, com variação relativa

Os resultados do script revelam que a versão concorrente do programa performa, em termos de precisão, tão bem quanto o sequencial. Contudo, ainda precisamos observar o comportamento de um desses arquivos .bin quando aumentamos a quantidade de threads, vamos tomar como exemplo a primeira linha da tabela.

# Threads	Produto Interno (THREADS)	Variação Relativa
2	1597.383692712532820223714224994182586 67	4.128×10^{-15}
8	1597.383692712533729718415997922420501 71	4.697×10^{-15}
16	1597.383692712535093960468657314777374 27	5.551×10^{-15}
32	1597.383692712533957092091441154479980 47	4.840×10^{-15}
64	1597.383692712535548707819543778896331 79	5.836×10^{-15}
128	1597.383692712533274971065111458301544 19	4.413×10^{-15}
256	1597.383692712533502344740554690361022 95	4.555×10^{-15}
512	1597.383692712535776081494987010955810 55	5.978×10^{-15}
1024	1597.383692712535321334144100546836853 03	5.694×10^{-15}
2048	1597.383692712534184465766884386539459 23	4.982×10^{-15}
4096	1597.383692712527135881828144192695617 68	5.694×10^{-16}

Table 2: Produto interno com múltiplas threads e variação relativa em relação ao sequencial

Quando aumentamos o número de threads para computar o produto interno, apesar da variação relativa ser extremamente pequena, existem momentos em que ela aumenta ligeramente e outros que ela diminui. Esse comportamento pode ser relacionado ao fato de que, ao distribuir os cálculos entre muitas threads, aparecerão diferenças na ordem de acumulação das somas parciais e como a soma de doubles em C não é associativa, o resultado final poderá ser afetado.

Apesar disso, um outro comportamento foi observado, quando colocamos 4096 threads em um produto interno de vetores de dimensão 4252, houve uma redução de variação relativa que provavelmente, que vale investigar um pouco mais. Para isso, vamos aumentar um pouco mais o número de threads.

Dimensão dos vetores: 4252		
# Threads	**Produto Interno (THREADS)**	**Variação Relativa**
4110	1597.383692712527135881828144192695617 68	5.694×10^{-16}
4220	1597.383692712526681134477257728576660 16	2.847×10^{-16}
4250	1597.383692712526453760801814496517181 40	1.423×10^{-16}

Table 3: Produto interno com número elevado de threads e variação relativa em relação ao sequencial

Agora podemos observar que com esse aumento elevado de número de threads, que quase alcança a dimensão dos vetores, a variação relativa diminuiu consideravelmente.