Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD1116 - COMPUTAÇÃO DE ALTO DESEMPENHO - T01 (2025.1)
Tarefa 8: Coerência de Cache e Falso Compartilhamento

Docente: SAMUEL XAVIER DE SOUZA
Discente: Iago Gabriel Nobre de Macedo (20220037927)
Link do repositório da atividade no Github: https://github.com/IagoGMacedo/IMD1116-Computacao-de-Alto-Desempenho/tree/main/exercicios/coerencia_cache_falso_compartilhamento

Implementação

Em primeiro lugar, desenvolvi o Main1.c utilizando com somente uma variável privada para contar os acertos e acumular o total em uma variável global. Em segundo lugar, implementei o Main2.c, nessa implementação fiz com que cada thread escreva seus acertos em uma posição distinta de um vetor compartilhado. Em ambos os casos deixei os trechos para alternar entre rand e rand_r.

Para avaliar o desempenho das implementações, foram realizados testes com um total fixo de 10.000.000 pontos, variando-se o número de threads utilizadas (1, 2, 4, 8 e 16).

Tempos de execução em segundos usando rand()		
Número de threads	**implementação 1**	**Implementação 2**
1	0.260347	0.265515
2	2.224668	1.357454
4	0.790966	0.779899
8	1.407933	1.468686
16	1.577913	1.497396
Tempos de execução em segundos usando rand_r()		
Número de threads	**implementação 1**	**Implementação 2**
1	0.056690	0.058171
2	0.033148	0.047977
4	0.027480	0.021516
8	0.023266	0.020592
16	0.012286	0.018675

Considerações

Pude perceber que a versão com variável global protegida por região crítica apresentou um aumento significativo no tempo de execução à medida que o número de threads aumentava, devido à contenção gerada pela necessidade de sincronização frequente entre as threads. Por outro lado, a versão que utiliza um vetor compartilhado apresentou melhor desempenho relativo, especialmente com um número maior de threads, já que cada thread escreve em uma posição distinta, reduzindo a necessidade de sincronização direta.

Ao substituir a função rand() por rand_r(), notou-se uma melhora considerável no desempenho em ambas as implementações. Isso ocorre porque a função rand() utiliza um estado global compartilhado, o que gera contenção interna e limita a escalabilidade em ambientes paralelos. Já a função rand_r() utiliza um estado privado por thread, eliminando essa contenção e permitindo maior eficiência na execução paralela.

Além disso, percebe-se que a segunda implementação, apesar de apresentar melhor desempenho que a primeira, ainda pode sofrer com o falso compartilhamento (false sharing). Esse efeito ocorre quando múltiplas threads acessam posições distintas, porém próximas, na memória compartilhada, causando invalidações frequentes das linhas de cache e reduzindo o desempenho geral. Uma possível solução para mitigar esse problema seria introduzir padding entre os elementos do vetor compartilhado, garantindo que cada thread acesse uma linha de cache distinta.