Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD1116 - COMPUTAÇÃO DE ALTO DESEMPENHO - T01 (2025.1)
Tarefa 5: Comparação entre programação sequencial e paralela

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/blob/main/exercicios/programa_multitarefa/primes.c

Implementação

Realizei o desenvolvimento do programa conforme solicitado. Fiz com que o valor máximo de n (até onde se deve dar a contagem dos números primos) seja passado por parâmetro de compilação. Depois disso, realizei as medições com o programa em sequencial. Após isso, Paralelizei o laço principal com a instrução "#pragma omp parallel for" do openMP e refiz todas as medições. Em ambos os casos, realizei as medições indo de 10 até 100000000.

Resultados Obtidos

Valor máximo n	Tempo Sequencial (segundos)	Contagem de primos sequencial	Tempo Paralelo (segundos)	Contagem de primos paralelo
10	0.000	4	0.023	4
100	0.000	25	0.020	21
1000	0.000	168	0.021	119
10000	0.001	1229	0.020	1050
100000	0.008	9592	0.008	9096
1000000	0.100	78498	0.047	72773
10000000	2.056	664579	0.686	650097
100000000	52.137	5761455	15.964	5748995

Os resultados obtidos são muito interessantes. Em primeiro lugar, podemos perceber que a abordagem com paralelização erra a quantidade de primos contabilizados em quase todos os cenários, com exceção do primeiro. Isso ocorre pois a forma como a paralelização foi implementada não é adequada para essa situação, pois não há proteção para o incremento da variável count, que é compartilhada entre as threads. Isso causa uma condição de corrida onde essas múltiplas threads tentam modificar a variável count, resultando em uma contagem incorreta.

Ademais, podemos que ver o desempenho do programa em paralelo melhora conforme o valor máximo de n também aumenta, ao passo, que no primeiro cenário, somente com 10 registros, podemos perceber que o programa sequencial é mais performático, levando menos tempo para a contagem. Por outro lado, no cenário onde a contagem deve ir até 100000000, o programa em paralelo consegue terminar a contagem em menos da metade do tempo do programa sequencial, por mais que a contagem termine errada por conta da forma como foi implementada.

Considerações

Considerando os resultados obtidos e refletido a respeito das problemáticas propostas, podemos entender que essa abordagem de implementação é falha pois desconsidera, entre outras coisas, os princípios de correção e distribuição de carga. Em primeiro lugar, como abordado anteriormente, a condição de corrida causada pelos múltiplos acessos à variável count acabam por prejudicar a contagem, de modo que dos 8 cenários de contagem, somente o primeiro e mais simples consegue contabilizar a quantidade de primos corretos no programa em paralelo. Para resolver essa problemática no cenário proposta, poderia ter sido utilizado a diretiva "#pragma omp parallel for reduction(+:count)", que garantiria que cada thread tenha sua própria de count e combine os resultados no final.

Outro ponto que é deixado de lado nessa abordagem de paralelização é a distribuição de carga. No nosso exemplo, podemos observar que o esforço computacional para verificar se o número 99999993 é primo é muito maior do que para verificar a mesma coisa com o número 3. Isso não é considerado na implementação, pois faz com que threads que processam números grandes terão mais trabalho. Para tal situação, podemos usar a diretiva "#pragma omp parallel for reduction(+:count) schedule(dynamic)", de modo que divide o loop em blocos menores e atribui dinamicamente às threads conforme elas terminam suas tarefas. Em um cenário onde precisamos de um desempenho computacional mais performático, essa solução trás uma melhoria para a gestão dos recursos.

Diante desses pontos, podemos perceber uma série de preocupações que devemos levar em conta ao desenvolvermos um programa em paralelo. No nosso cenário, a mera utilização de uma

diretiva inadequada não garantirá um resultado “melhor” ou mesmo correto para o funcionamento do nosso programa.