Relatório - Trabalho de OpenMP

Matheus Marques Barros

Mauro Gonçalves Bueno

Thiago Serra Dias Aguiar

Professora Lucia Maria de Assumpção Drummond

TCC00344 - Laboratório de Programação Paralela - 2024.1

1. Objetivo

Comparar o tempo de execução de dois programas que utilizam a paralelização do OpenMP para calcular a quantidade de números primos em um determinado intervalo. E, a partir disto, calcular as medidas de desempenho "*speedup*" e "eficiência".

2. Introdução

Durante as aulas de Laboratório de Programação Paralela, foram apresentados dois códigos para calcular a quantidade de números primos entre 1 e um valor informado. Ambos utilizam mpi.h, uma biblioteca da linguagem C, a qual permite a utilização de threads para executar o programa de forma paralela, reduzindo o tempo de execução. Para este relatório, foram produzidas versões destes algoritmos utilizando a biblioteca omp.h.

Apesar de ambas as bibliotecas permitirem a paralelização do código, há diferenças que influenciam na confecção e no resultado. O método MPI (mpi.h) segue um modelo de memória distribuída e é orientado a mensagens, tornando-o mais flexível. Por outro lado, o OpenMP (omp.h) utiliza um modelo de memória compartilhada e é orientado a diretivas, facilitando a programação e a detecção de erros.

No primeiro programa (referenciado como Naive), cada processo possui uma variável de início relativa ao seu ranque. Além disso, todos os processos compartilham uma variável de salto proporcional à quantidade de *threads*. Esses dois valores garantem que, ao performar um loop, cada processo receba um número diferente (sempre ímpar) para verificar se é primo. Por fim, a cláusula *reduction* soma os resultados parciais em uma variável compartilhada.

No segundo programa (referenciado como Bag-of-tasks), há uma variável global que define o tamanho de trabalho repassado para cada processo. Assim, por meio de uma cláusula *schedule* cada processo recebe a mesma quantidade inicial de trabalho. Conforme uma thread completa o que lhe foi passado, uma nova quantia de trabalho é determinada. Isto se repete até o término do loop. Por exemplo, se o tamanho é definido como cinquenta, então a thread 0 vai testar os números de 3 a 53 (exclusive), enquanto a 1 vai testar de 53 a 103, e assim por diante. Como dito anteriormente, este loop ignora os números pares. Dessa forma, processos mais rápidos recebem mais tarefas conforme terminam o que lhes foi atribuído.

3. Metodologia

Para a realização dos testes, pensando em uniformidade, todos os programas foram executados em uma mesma máquina, de processador Ryzen 5600G, de seis núcleos, e 16GB de memória RAM.

Definido o equipamento, o primeiro passo foi a execução de um código sequencial com o mesmo objetivo de calcular a quantidade de número primos. Foi usado o mesmo código sequencial criado para o Trabalho 1, pensando na futura comparação entre os testes em MPI e OPENMP. Usando o código sequencial, foi possível usar as fórmulas abaixo para calcular o *speedup* (aceleração) e a eficiência dos diferentes tipos de comunicação.

$$S(n) = \frac{T(1)}{T(n)}$$

$$E(n) = \frac{S(n)}{n}$$

Sendo n o número de processos, S(n) representa a aceleração e E(n) representa a eficiência. O tempo de execução do código sequencial elaborado é usado para T(1), com o tempo de execução de um teste com n processos sendo representado por T(n). Dessa forma, pode-se calcular os valores sem muita dificuldade.

Para uniformizar os testes, evitando possíveis falhas extraordinárias, cada teste foi realizado em três rodadas, com a média entre os três sendo o valor final aplicado nas fórmulas. Neste caso, essa metodologia foi acertada, pois a falha extraordinária ocorreu e, na última execução de Bag of Tasks para n = 10^9, o script que estava realizando as execuções foi interrompido inesperadamente. Devido a outros imprevistos, o código não pôde ser reexecutado, resultando em apenas duas rodadas de execução. Sendo assim, a ideia de reduzir impactos foi benéfica, uma vez que ainda havia execuções válidas e comparáveis para o teste em questão.

Os códigos utilizados consistem em algoritmos paralelos que calculam a quantidade de números primos entre 1 e um "N" escolhido pelo usuário, com o resultado de uma execução sendo a quantidade calculada e o tempo de execução.

Para que fosse possível garantir uma variação apropriada de tempo, os testes foram realizados com diferentes parâmetros de entrada: 10^6, 10^7, 10^8 e 10^9, sendo que, para cada caso, exceto 10^9 para Bag of Tasks, foram realizadas três rodadas de testes.

Dessa forma, havendo os códigos de Naive e Bag of Tasks, somado a um programa sequencial, totalizaram-se três códigos a serem rodados, todos com 10^6, 10^7, 10^8 e 10^9 como parâmetro e cada um desses casos possuindo três rodadas de testes. Além disso, fora o código sequencial, os códigos foram executados para 2, 4 e 8 processos. Dessa forma, totalizaram-se 81 execuções testadas ao longo de mais de 4 horas.

Para se executar os códigos sem que houvesse grande demanda por atenção, foi feito um script que pudesse realizar as execuções necessárias automaticamente, gerando um arquivo '.txt' com os resultados ao final. Utilizando esses resultados, foram montadas planilhas que pudessem realizar de forma automatizada os cálculos de Speedup e Eficiência.

4. Resultados

Para a execução do código sequencial, o qual não aplicava nenhuma das funções mpi, utilizou-se a compilação do mpicc. Mesmo assim, essa execução foi consideravelmente mais rápida do que o código compilado pelo gcc. Dessa forma, ao utilizar "mpirun" para rodar o código, indicou-se o número de threads 1 para que o código sequencial ocorresse apenas uma vez. Os resultados (em segundos) estão descritos nas Tabelas 01, 02, 03 e 04.

Tabela 01 - Resultados para programa sequencial com N = 10^6				
Repetições	**1**	**2**	**3**	**Média**
mpi_primos_sequencial	0,089	0,087	0,087	0,088

Tabela 02 - Resultados para programa sequencial com N = 10^7				
Repetições	**1**	**2**	**3**	**Média**
mpi_primos_sequencial	2,211	2,222	2,212	2,215

Tabela 03 - Resultados para programa sequencial com N = 10^8				
Repetições	**1**	**2**	**3**	**Média**
mpi_primos_sequencial	58,616	58,640	58,589	58,615

Tabela 04 - Resultados para programa sequencial com N = 10^9				
Repetições	**1**	**2**	**3**	**Média**
mpi_primos_sequencial	1571,873	1571,820	1571,473	1571,722

Como esperado, quando N = 10^9, o tempo de execução aumenta consideravelmente, o que será discutido posteriormente. Possuindo os resultados das três execuções para cada valor de N, foi realizado o cálculo da média. Após isso, foram executados os programas paralelos para os mesmos valores de N. Nas Tabelas 05, 06, 07 e 08, podem-se ver os resultados para as combinações do método Naive.

Tabela 05 - Resultados abordagem Naive com N = 10^6												
Repetições	**1**			**2**			**3**			**Média**		
Processos	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**
omp_primos_naive	0,048	0,024	0,019	0,048	0,025	0,018	0,048	0,025	0,018	0,048	0,025	0,018

Tabela 06 - Resultados abordagem Naive com N = 10^7												
Repetições	**1**			**2**			**3**			**Média**		
Processos	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**
omp_primos_naive	1,206	0,607	0,433	1,204	0,875	0,457	1,207	0,813	0,457	1,206	0,765	0,449

Tabela 07 - Resultados abordagem Naive com N = 10^8												
Repetições	**1**			**2**			**3**			**Média**		
Processos	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**
omp_primos_naive	32,032	19,858	11,336	31,783	18,063	11,912	31,773	16,339	11,418	31,863	18,086	11,555

Tabela 08 - Resultados abordagem Naive com N = 10^9												
Repetições	**1**			**2**			**3**			**Média**		
Processos	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**
omp_primos_naive	868,789	490,323	300,488	866,748	490,133	309,707	870,690	511,027	308,795	868,743	497,161	306,330

Após a execução dos códigos no método Naive, foram executados os códigos utilizando Bag of Tasks. Os resultados podem ser vistos nas Tabelas 09, 10, 11 e 12.

Tabela 09 - Resultados abordagem Bag of Tasks com N = 10^6												
Repetições	**1**			**2**			**3**			**Média**		
Processos	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**
omp_primos_bag	0,048	0,027	0,018	0,048	0,025	0,018	0,049	0,025	0,018	0,048	0,026	0,018

Tabela 10 - Resultados abordagem Bag of Tasks com N = 10^7												
Repetições	**1**			**2**			**3**			**Média**		
Processos	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**	**2**	**4**	**8**
omp_primos_bag	1,207	0,690	0,457	1,208	0,607	0,438	1,207	0,606	0,459	1,207	0,634	0,451

Tabela 11 - Resultados abordagem Bag of Tasks com N = 10^8												
Repetições	1			2			3			Média		
Processos	2	4	8	2	4	8	2	4	8	2	4	8
omp_primos_bag	31,943	22,232	11,884	31,777	16,160	11,240	31,806	22,840	11,585	31,842	20,411	11,570

Tabela 12 - Resultados abordagem Bag of Tasks com N = 10^9									
Repetições	1			2			Média		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_bag	884,415	491,433	300,005	867,668	486,911	304,814	876,042	489,172	302,409

Então, para analisar os speedups e eficiências, foi feita uma tabela utilizando as fórmulas apresentadas anteriormente. Para T(1), foi utilizada a média das três rodadas do código sequencial. Da mesma forma, para os tempos de execução de cada programa executado, foi-se utilizada a média das três rodadas (exceto para os valores da Tabela 12, que só possuem duas rodadas).

Nas Tabelas 13, 14, 15 e 16, os cálculos para o método Naive.

Tabela 13 - Abordagem Naive, desempenho com N = 10^6									
	Tempo de Execução			Speed Up			Eficiência		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_naive	0,048	0,025	0,018	1,833	3,520	4,889	0,917	0,880	0,611

Tabela 14 - Abordagem Naive, desempenho com N = 10^7									
	Tempo de Execução			Speed Up			Eficiência		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_naive	1,206	0,765	0,449	1,837	2,895	4,933	0,918	0,724	0,617

Tabela 15 - Abordagem Naive, desempenho com N = 10^8									
	Tempo de Execução			Speed Up			Eficiência		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_naive	31,863	18,086	11,555	1,840	3,241	5,073	0,920	0,810	0,634

Tabela 16 - Abordagem Naive, desempenho com N = 10^9									
	Tempo de Execução			Speed Up			Eficiência		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_naive	868,743	497,161	306,330	1,809	3,161	5,131	0,905	0,790	0,641

E, por fim, nas Tabelas 17, 18, 19 e 20, os cálculos para o método Bag of Tasks.

Tabela 17 - Abordagem Bag of tasks, desempenho com N = 10^6									
	Tempo de Execução			Speed Up			Eficiência		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_bag	0,048	0,026	0,018	1,833	3,385	4,889	0,917	0,846	0,611

Tabela 18 - Abordagem Bag of tasks, desempenho com N = 10^7									
	Tempo de Execução			Speed Up			Eficiência		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_bag	1,207	0,634	0,451	1,835	3,494	4,911	0,918	0,873	0,614

Tabela 19 - Abordagem Bag of tasks, desempenho com N = 10^8									
	Tempo de Execução			Speed Up			Eficiência		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_bag	31,842	20,411	11,570	1,841	2,872	5,066	0,920	0,718	0,633

Tabela 20 - Abordagem Bag of tasks, desempenho com N = 10^9									
	Tempo de Execução			Speed Up			Eficiência		
Processos	2	4	8	2	4	8	2	4	8
omp_primos_bag	876,042	489,172	302,409	1,794	3,213	5,197	0,897	0,803	0,650

5. Discussão

5.1. Naive

Ao analisar o desempenho no Naive, observamos que, para todos os tamanhos de entrada executados, o método demonstra um ganho de *speedup* razoável e máxima eficiência para dois processos, porém, à medida que novos processos são adicionados, observa-se uma perda de eficiência gradual. Obtemos em média uma eficiência de 0,913 para dois processos, 0,810 para quatro processos e 0,625 para oito processos. Isto sugere que, independente do tamanho de N, o método Naive não consegue obter o aproveitamento ótimo que mais threads podem proporcionar.

5.2 Bag of Tasks

Nota-se uma grande semelhança entre os resultados obtidos pelos dois métodos. As pequenas variâncias podem ser atribuídas ao *overhead* da máquina (i.e. o estresse causado por demais programas concorrentes, como o próprio sistema operacional).

Vale ressaltar que este não era o resultado esperado, uma vez que o Bag of Task é capaz de reduzir a ociosidade durante a execução. Logo, seus valores de tempo deveriam ser inferiores aos apresentados pelo Naive. Essa discrepância serviu como motivação para um maior aprofundamento na funcionalidade das ferramentas utilizadas.

5.3 Conclusão:

Ao comparar o tempo de execução do programa sequencial com os demais, nota-se que sempre há um ganho de desempenho. Logo, pode-se concluir que tanto o Naive quanto o Bag-of-tasks se beneficiam do uso da programação paralela ao trabalhar com números maiores ou iguais a 10^6, e que a programação paralela em OpenMP é eficaz para reduzir o tempo de execução em comparação com o código sequencial. O aumento no número de processos resulta em bons ganhos de speedup, embora a eficiência diminua à medida que mais processos são adicionados.

5.4 Comparação com o trabalho anterior

Este tópico serve como um anexo para poder comparar a tecnologia MPI analisada no último relatório com a OpenMP vista neste. Por não ser o foco deste trabalho, as tabelas não foram repetidas, sendo necessário visitar a última publicação para consulta.

Ao comparar somente as medidas de desempenho dos dois métodos, observa-se que, de forma geral, o MPI é mais eficiente ao utilizar uma maior quantidade de processos. Enquanto que o OpenMP apresenta uma eficiência maior conforme haja menos threads.

Deste modo, conclui-se que, apesar de facilitar a programação, o OpenMP não deve ser usado em qualquer cenário. Igualmente, a flexibilidade do MPI não garante o melhor resultado em todas as condições.

6. Referências bibliográficas

SILVA, Gabriel P.; BIANCHINI, Calebe P.; COSTA, Evaldo B.. **Programação Paralela e Distribuída** com MPI, OpenMP e OpenACC para computação de alto desempenho. São Paulo: Casa do Código, 2022. ISBN 978-85-5519-303-3.