IFES CAMPUS SERRA
SISTEMAS DE INFORMAÇÃO
SISTEMAS OPERACIONAIS

EMANUEL HOFFMANN
VICTOR EMERSON

Introdução à Programação Multithread com PThreads
Relatório

Serra - ES
2023

SUMÁRIO

1. INTRODUÇÃO

2. QUESTÃO PROPOSTA

3. CONFIGURAÇÕES DOS COMPUTADORES

4. DESENVOLVIMENTO DO CÓDIGO

5. TESTES

- **5.1 TESTES MACROBLOCOS**
- **5.2 TESTES SPEEDUP**
- **5.3 TESTE DE PONTO CRÍTICO**
- **5.4 TESTES SEM MUTEX**
- **5.5 DESEMPENHO x64 X x86**

6. CONCLUSÃO

Serra - ES
2023

1. INTRODUÇÃO

Emanuel Hoffmann e Victor Emerson, estudantes da disciplina de Sistemas Operacionais ministrada pelo professor Flávio Ghiraldelli. Neste documento, exploraremos as análises e soluções referentes ao desafio proposto pelo professor Ghiraldelli. Nosso objetivo principal ao abordar essa tarefa é aplicar os conhecimentos adquiridos em sala de aula e os recursos fornecidos para lidar com programação concorrente, tratamento de seções críticas e sincronização de processos e threads por meio de mutex. Além disso, realizaremos uma análise de desempenho, considerando diferentes cenários com variações no número de threads.

Serra - ES
2023

2. QUESTÃO PROPOSTA

O problema deseja que seja feito uma algoritmo o qual contabilize o número de números primos existentes em uma matriz com números gerados aleatoriamente (no intervalo de 0 a 31999). Para este teste devemos analisar duas formas: De modo serial, o qual a contagem dos primos é feita um a um, e do modo paralelo, o aquela possui uma subdivisão na matriz em “macroblocos” que serão as unidades de trabalho de cada thread. A matriz não é necessariamente quadrada e os macroblocos terão tamanhos que podem variar de desde um único elemento até a matriz toda (equivalente ao caso serial).

3. CONFIGURAÇÕES DOS COMPUTADORES

	PROCESSADOR	FREQUÊNCIA	MEMÓRIA
COMPUTADOR 2	AMD Ryzen 5 7535HS with Radeon Graphics	3.30 GHz (4.55GHz)	16,0 GB DDR5 4200
COMPUTADOR 1	Intel(R) Core(TM) i5-1035G1	1.00 GHz (3.60 GHz)	8,00 GB DDR4 2666

Usamos dois notebooks com um poder de processamento e com kits diferentes para poder ter uma melhor análise sobre o trabalho, ambos testes foram feitos com o notebook na tomada para garantir o desempenho.

4. DESENVOLVIMENTO DO CÓDIGO

Na elaboração do código, decidimos documentar no próprio com comentários sobre o que está acontecendo ou para que serve às partes para aprimorar a compreensão da solução. Isso facilita tanto para o professor compreender nossa lógica quanto para nós mesmos organizarmos melhor o raciocínio.

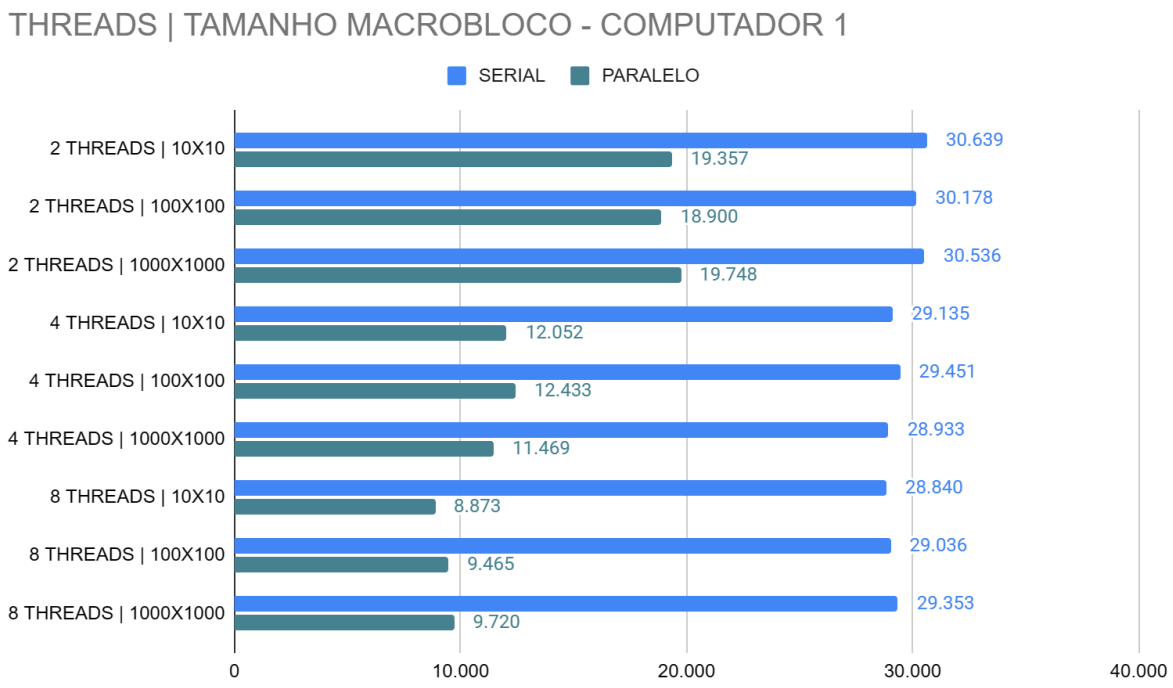
Em relação ao código conseguimos chegar a um certo nível de maestria no que diz respeito ao tempo de execução, mas optamos por fazer uma pequena alteração que já foi suficiente para deixar o código menos performático para que as análises saíssem de forma mais eficiente tendo em vista que o tempo foi maior.

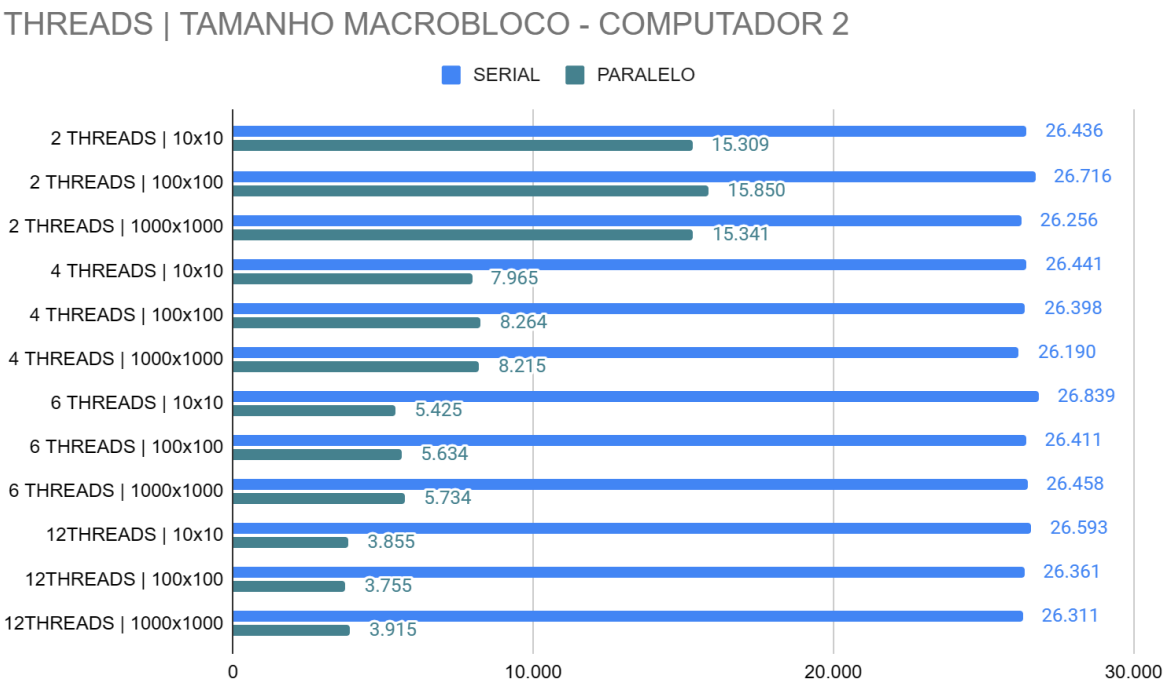
```c
int ehPrimo(int n) {
    /*Variavel que informe se n é primo*/
    int flag = 1;
    if (n > 1) {
        /*Obtendo a raiz quadrada de n*/
        int raiz_quad = sqrt(n);
        for (int i = 2; i <= raiz_quad; i++) {
            /*Caso a divisão por i seja o resto != 0 informa que n não é primo e p/ busca*/
            if (n % i == 0) {
                flag--;
                break;
            }
        }
    }
    else flag--;

    return flag;
}
```

Os testes foram feitos sem o break

5.1 TESTES MACROBLOCO

Para esse testes usamos como padrão matriz 10000 X 10000, rodamos em diferentes computadores cuja especificações estão no item 3. Usamos diferentes threads em diferentes tamanho de macroblocos em dois processadores substancialmente diferentes. O tempo do comparativo está em segundos o ponto está sendo usado para poder pegar mais precisamente os milissegundos. O teste também foi feito apenas em x64





Ao examinar os gráficos, notamos algo interessante, os tamanhos dos macroblocos não mostram diferenças significativas no intervalo que fizemos os testes. No entanto, fica evidente que à medida que aumentamos o número de threads, o tempo de execução diminui. Isso acontece porque as threads conseguem trabalhar ao mesmo tempo em seus próprios blocos, realizando vários cálculos simultaneamente. Ter mais threads aproveita melhor a capacidade de processamento paralelo nos computadores analisados.

5.2 TESTES SPEEDUP

Para este teste usamos os computadores descritos no item 3. Antes de tudo usamos uma matriz 10000 x 10000 e o macro bloco de 100 x 100. No teste vamos analisar um pouco sobre o speedup após as tabelas abaixo:

	X64			
	N DE THREADS	SERIAL (s)	PARALELO	SPEEDUP
PC 1	2 THREADS	30.178	18.900	1,60
	4 THREADS	29.451	12.433	2,37
	8 THREADS	29.036	9.465	3,07
	N DE THREADS	SERIAL	PARALELO	SPEEDUP
PC 2	2 THREADS	26.716	15.850	1,69
	4 THREADS	26.398	8.264	3,19
	6 THREADS	26.411	5.634	4,69
	12THREADS	26.361	3.755	7,02

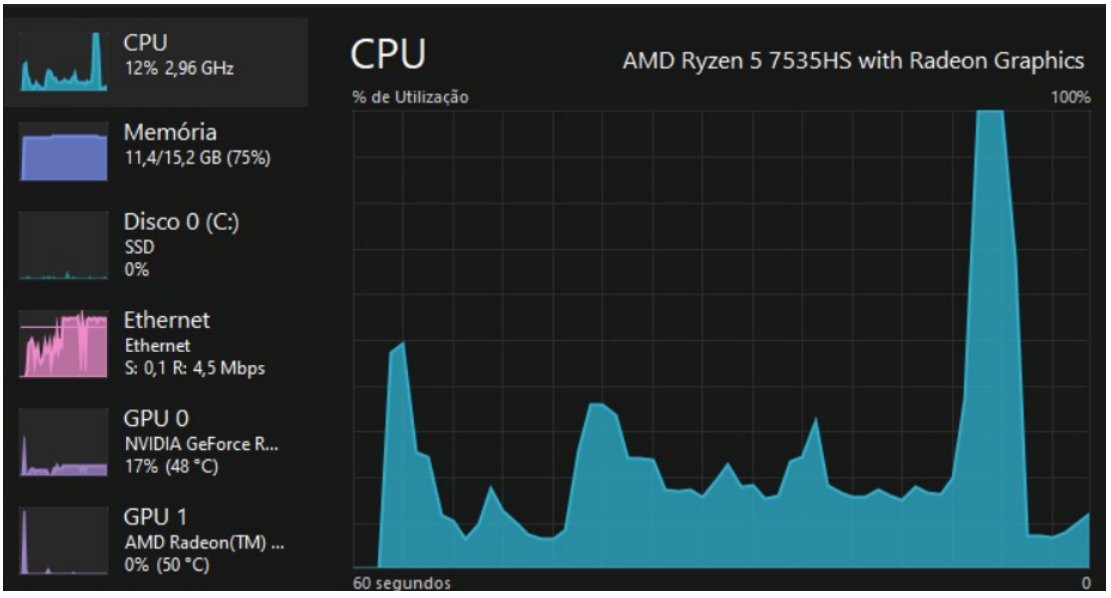
O uso de paralelismo pode significar reduzir os tempos de execução, como evidenciado pelos resultados que podem ser observados na tabela. Aqui vemos a lei Amdhal, se destaca a importância de otimizar a fração paralela do código para obter o máximo benefício do paralelismo e com isso ter um maior desempenho,observando a tabela podemos ver o ganho no speedup à medida que vamos aumentando o paralelismo.

5.3 TESTE DE PONTO CRÍTICO

Neste teste usamos uma matriz 10000 x 10000 com macro bloco de 100 x 100. Decidimos usar o computador 2 para para fazer o teste pois o computador 1 a todo momento ficava próximo de 100% pois é uma máquina mais antiga, para estressar usamos 500 threads. Feito em X64. Este teste de estresse consiste em alterar o número de threads para um valor bem mais alto do que o necessário para poder observar os eventos que acontecem tanto em relação ao processador quanto aos resultados dos tempos.

COMPUTADOR 2	SERIAL (s)	PARALELO (s)
12 THREADS	26.361	3.755
500 THREADS	26.355	3.863

Essa tabela é a diferença de rodar em 12 threads e 500 threads nas mesmas condições, podemos notar que ele já começa a ficar ligeiramente mais lento e também que mesmo que aumentar a thread em um valor exponencialmente maior não vai trazer resultados melhores.



Neste ‘print’ é possível ver o estresse que foi causado ao processador no momento em que se inicia o multi thread excessivo que fizemos para o teste.

5.4 TESTES SEM MUTEX

Para realizar esse teste foi necessário comentar no código toda a parte onde se é usado o mutex, que como aprendido durante a matéria de Sistemas Operacionais é um mecanismo indispensável para ser usado em ambientes que tenham concorrência, assim como o trabalho em questão. Ao realizar os testes sem o mutex ainda durante a execução do teste é possível ver que muitas vezes houve erro durante a contagem de primos. A tabela abaixo vai representar os números que encontramos ao realizar esse teste no computador 1, com uma matriz de 10000 X 10000 e o macro bloco setado em 10 X 10.

NÚMEROS DE THREADS	NÚMERO DE PRIMOS <PARALELO>
2 THREADS	10895218
4 THREADS	10916333
8 THREADS	10936211

Ao rodar o mesmo código nas mesmas condições porém com o mutex ‘descomentado’ temos a seguinte tabela

NÚMEROS DE THREADS	NÚMERO DE PRIMOS <PARALELO>
2 THREADS	10884454
4 THREADS	10884454
8 THREADS	10884454

5.5 DESEMPENHO x64 X x86

Para realizar este teste fizemos testes nos dois computadores citados no item 3, foi feita com uma matriz de 10000 x 10000, com 4 threads e com isso tivemos as seguintes tabelas:

	ARQUITETURA	THREADS	TMP SERIAL(s)	TMP PARALELO(s)
COMPUTADOR 1	X64	4	29.451	12.433
COMPUTADOR 1	X86	4	35.869	13.346
Resultado do X64 - X86 em segundos =			-6.418s	-1s

	ARQUITETURA	THREADS	TMP SERIAL(s)	TMP PARALELO(s)
COMPUTADOR 2	X64	4	26.398	8.264
COMPUTADOR 2	X86	4	27.222	8.335
Resultado do X64 - X86 em segundos =			-1s	-1s

Os resultados indicam que no primeiro computador, a arquitetura X64 teve uma melhoria significativa no tempo serial em comparação com a X86, com uma redução de 6.5 segundos. No entanto, no segundo computador, a diferença entre as arquiteturas não foi tão marcante, com uma diferença de apenas 1 segundo. Esses resultados sugerem que a arquitetura do processador pode impactar o desempenho, mas a quantidade dessa influência pode variar entre diferentes sistemas em processadores diferentes. Além disso, a paralelização do código parece trazer benefícios significativos em ambos os casos, reduzindo consideravelmente o tempo de execução.

6. CONCLUSÃO

O trabalho foi desafiador, usar a linguagem C juntamente com o conceito aprendido na matéria de sistemas operacionais é uma experiência única. A implementação do algoritmo de contagem de números primos utilizando programação multithread com a biblioteca ‘pthreads’ mostrou resultados promissores em termos de desempenho quando comparada à versão serial. Ao dividir a tarefa de verificação de primos em macroblocos e distribuí-la entre múltiplas threads, observamos uma redução significativa no tempo total de execução. O uso adequado dos ‘mutexes’permitiu que as threads compartilhassem os resultados de forma segura, evitando condições de corrida entre si. A estratégia de distribuir os macroblocos entre as threads possibilitou uma melhor utilização dos recursos disponíveis, especialmente nos testes que usam múltiplos núcleos. A análise de desempenho revelou que o speedup, definido como a razão entre o tempo de execução ‘serial’ e o tempo de execução paralelo, indicando ganhos significativos em eficiência.