

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
SISTEMAS DE INFORMAÇÃO**

**INE5645 - Programação Paralela e Distribuída**

Profº Dr. Odorico Machado Mendizabal

Alan Vinicius Cezar Ensina (16206354)


**A.6 OpenMP: Adaptação do "Primo paralelo"**

A análise a seguir mostra a comparação de três programas que foram desenvolvidos para verificar todos os números primos dentro de um limite. Sendo um código para o processamento totalmente sequencial e os outros dois para processamento paralelo, onde um deles utiliza POSIX threads e outro OpenMP.

**Processamento sequencial**

O primeiro código analisado é o de processamento sequencial, onde não foi utilizado POSIX threads e nem OpenMP.

Foram analisados e exibidos todos os números primos no intervalo de 1 a 10.000. E o tempo de processamento foi de 0,039 segundos.



```
-----  
Tempo total = 0.039295  
-----
```

Por se tratar de um código sequencial, sua implementação é muito fácil de ser implementada.

## Processamento paralelo utilizando POSIX threads

No segundo caso, foi utilizado POSIX threads. Sua implementação foi a mais complicada, pois foi definida a granularidade do processamento, ou seja, o laço principal foi dividido conforme a quantidade de threads.

```
81  pthread_create( &threads[i], NULL, processar, (void *) dados);
82  for (int i = 0; i < cores; i++) {
83
84      Data * dados = (Data *) malloc(sizeof(Data));
85
86      if (i == 0) {
87          dados -> min = array[intervalo * i];
88          dados -> max = array[intervalo * (i + 1)];
89      }
90
91      if (i != 0 && i < cores && i != cores - 1) {
92          dados -> min = array[(i * intervalo) + 1];
93          dados -> max = array[((i * intervalo) + 1) + intervalo - 1];
94      }
95
96      if (i == cores - 1) {
97          dados -> min = array[(i * intervalo) + 1];
98          dados -> max = array[tam - 1] + 1;
99      }
100     pthread_create( &threads[i], NULL, processar, (void *) dados);
101 }
```

Foram analisados e exibidos todos os números primos no intervalo de 1 a 10.000 utilizando 4 threads para processamento, o tempo total de processamento foi de 0,00047 segundos.

```
-----
Tempo total = 0.000471
-----
```

## Processamento paralelo utilizando OpenMP

A implementação feita utilizando OpenMP não definiu um número específico de núcleos de processamento, sendo assim, utilizou a quantidade máxima de núcleos da máquina utilizada para implementação, no caso 4 núcleos.

Foi utilizado apenas a diretiva `#pragma omp parallel for`, o tempo total de processamento foi de 0,074 segundos para analisar e exibir todos os números primos no intervalo de 1 a 10.000.

```
23  
24     gettimeofday(&t1, NULL);  
25     #pragma omp parallel for  
26     for(int i = 0 ; i <= atoi(argv[1]); i++){  
27         validarNumero(i);  
28     }  
29     gettimeofday(&t2, NULL);  
30
```

```
-----  
Tempo total de execução = 0.074621  
-----
```

## Considerações finais

Podemos concluir que não existe melhor nem pior forma de implementar paralelismo em absoluto, mas sim qual o melhor ou pior conforme o contexto do problema que se deseja solucionar.

Analisando as três formas citadas acima, se fossemos optar pela opção de implementação mais fácil, logo escolheríamos a sequencial pois não teríamos que nos preocupar com gerenciamento de threads e nem de uma nova diretiva (no caso a OpenMP).

Caso o cenário exigisse um processamento paralelo e de fácil implementação, o OpenMP tem uma implementação muito mais fácil do que a implementação de POSIX threads.

Por fim, se o cenário fosse mais minucioso, onde o foco é obter uma performance mais específica e de uma granularidade menor, o uso de POSIX threads é o recomendado.

### **Link dos repositórios**

[Código fonte do processamento sequencial e paralelo utilizando POSIX Threads](#)

[Código fonte do processamento paralelo utilizando OpenMP](#)