# Avaliação de Desempenho do Crivo Paralelo

Computação Paralela

Luiz Junio Veloso Dos Santos – Matricula: 624037

1 de setembro de 2020

Tabela 1: Comparativo das métricas de performance do programa Perf.

| Contador | Sequencial | Paralelo (Dinâmico) | Paralelo (Estático) |
|---|---|---|---|
| **CPUs utilized** | 1 | 1,97 | 1,97 |
| **frontend cycles idle** | n/a | n/a | n/a |
| **backend cycles idle** | n/a | n/a | n/a |
| **instructions per cycle** | 0,65 | 0,33 | 0,33 |
| **LL-cache hits** | 12,36% | 3,87% | 6,42% |
| **time elapsed** | 1,43 seg | 1,218 seg | 1,206 seg |

O maior gargalo desse programa está sendo no seu *for* externo, que não foi paralelizado na tarefa 07, o que resultou na diminuição do número de instruções por ciclo, por fazer menos processamento.

Havia paralelizado apenas o *for* interno por acreditar, que dado o funcionamento do Crivo de Aristóteles, eu não poderia "remover" os múltiplos, por exemplo 2,4,8 e uma outra thread acessar uma posição como a 4, que foi "removida".

Mas como essa implementação usa de valores booleanos e não remoção uma remoção física do valor em uma posição do array, não irá ocorrer problema. Contudo se então eu paralelizar o *for* externo, na teoria, terei um aumento nas instruções por ciclo, mas também um aumento do tempo, uma vez que estarei possivelmente acessando mais vezes o *for* interno.

Uma melhoria seria aumentar o número de threads para 4, e experimentar diferentes politicas de escalonamento.

Figura 1: Código utilizado com as primitivas OpenMP

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int sieveOfEratosthenes(int n)
{
    // Create a boolean array "prime[0..n]" and initialize
    // all entries it as true. A value in prime[i] will
    // finally be false if i is Not a prime, else true.
    int primes = 0;
    bool *prime = (bool*) malloc((n+1)*sizeof(bool));
    int sqrt_n = sqrt(n);

    memset(prime, true,(n+1)*sizeof(bool));
    for (int p=2; p <= sqrt_n; p++)
    {
        // If prime[p] is not changed, then it is a prime
        if (prime[p] == true)
        {
            // Update all multiples of p
            // Paraleliza a "remoção" de multiplos
            #pragma omp parallel for num_threads(2)
            for (int i=p*2; i<=n; i += p)
                prime[i] = false;
        }
    }

    // count prime numbers
    // Paraleliza a contagem de primos.
    // Usando o padrão reduce
    #pragma omp parallel for num_threads(2) reduction(+:primes) schedule (dynamic,100)
    for (int p=2; p<=n; p++)
        if (prime[p])
            primes++;

    return(primes);
}

int main()
{
    int n = 100000000;
    printf("%d\n",sieveOfEratosthenes(n));
    return 0;
}
```