Nome: Luigi Soares Sorrentino – matricula: 540540

Tabela

Contador	Sequencial	Paralelo
Task-clock (utilização)	0,945 CPUs	1,958 CPUs
Stalled-cycles-frontend (parado na ULA)	--	--
Stalled-cycles-backend (parado na busca)	--	--
Instructions (instruções por ciclo)	0,41	0,25
LLC-load-misses (falta na cache L3)	10,06 %	10,80 %
Time elapsed (tempo de execução)	3,915423919	3,051654312

Gargalos

Gargalo n1:

```
1   // Update all multiples of p
2   for (int i=p*2; i<=n; i += p)
3   prime[i] = false;
```

Remover os múltiplos de P pode ser uma tarefa em paralelo, enquanto uma thread remove um múltiplo, outra thread pode remover outro múltiplo melhorando o desempenho.

Gargalo n2:

```
1   // count prime numbers
2   for (int p=2; p<=n; p++)
3   if (prime[p])
4   primes++;
```

Verificar e contar cada posição do vetor sequencialmente demanda muito tempo, logo ao paralelizar esse for, teremos mais threads contando e somando no ‘primes’, melhorando o desempenho.

Otimização

Para obtermos uma melhoria, teremos que alterar o código.

```
1   for (int p=2; p <= sqrt_n; p++)
2   {
3       if (prime[p] == true)
4       {
5           #pragma omp parallel for num_threads(2)
6           for (int i=p*2; i<=n; i += p){
7               prime[i] = false;
8           }
9           primes++; // incremento dos primos
10      }
11  }
```

Ao excluir os múltiplos do número que é primo, já é feita a soma no contador de primos. Logo não teremos que percorrer o vetor verificando quem foi marcado como primo, reduzindo N vezes o custo do algoritmo.