Aluno: Geovane Fonseca de Sousa Santos
Matrícula: 553237
Matéria: Coputação Paralela
Tarefa 10: Avaliação de Desempenho do Crivo Paralelo

Tabelas com as métricas:

MÁQUINA LOCAL	Sequencial	Paralelo
Utilização de CPU (CPUs utilized)	1,000	3,830
Ciclos ociosos na ULA (frontend cycles idle)	<not supported>	<not supported>
Ciclos ociosos na busca de instrução (backend cycles idle)	<not supported>	<not supported>
IPC ou instruções por ciclo (instructions per cycle)	0,52	0,32
Taxa de falta na cache L3 (LL-cache hits)	50,02%	50,16%
Tempo total de execução (time elapsed)	2,367386948 seconds	1,354655200 seconds

MÁQUINA PARCODE	Sequencial	Paralelo
Utilização de CPU (CPUs utilized)	0,096	0,320
Ciclos ociosos na ULA (frontend cycles idle)	<not supported>	<not supported>
Ciclos ociosos na busca de instrução (backend cycles idle)	<not supported>	<not supported>
IPC ou instruções por ciclo (instructions per cycle)	0,43	0,47
Taxa de falta na cache L3 (LL-cache hits)	25,20%	24,71%
Tempo total de execução (time elapsed)	37,628896164 seconds	12,272887967 seconds

Observações:

- A utilização devereria ser próxima a 4 devido a quantidade de núcleos utilizados pelo programa. No caso da máquina PARCODE esse valor está longe devido ao compartilhamento dela entre os usuários.
- Não foi possível observar o número de ciclos ociosos em ambas as máquinas e em nenhum dos casos.

- Na máquina Local a quantidade de instruções por ciclo sequencial diminuiu bastante em relação ao paralelo enquanto na máquina PARCODE teve um pequeno aumento

- A taxa de falta na cache L3 não teve alteração considerável nas duas máquinas em ambos os casos.

Gargalos:

- No seguinte trecho de código, podemos identificar a necessidade de tornar o inteiro i como privado para que as diferentes threads do for:

```
1.  #pragma omp parallel for schedule(dynamic)
2.  for (int p=2; p <= sqrt_n; p++)
3.  {
4.      // If prime[p] is not changed, then it is a prime
5.      if (prime[p] == true)
6.      {
7.          // Update all multiples of p
8.          for (int i=p*2; i<=n; i += p)
9.              prime[i] = false;
10.         }
11.     }
```

- O balanceamento de carga encontra-se da maneira correta nele, podendo aumentar o número de chunks para 100 e assim obter um desenpenho melhor.

- No seguinte trecho é necessário definir o escalonamento das threads, para balanceá-las, além de que a variável primes é necessariamente uma redução:

```
12.     #pragma omp parallel for reduction(+:primes)
13.     for (int p=2; p<=n; p++)
14.         if (prime[p])
15.             primes++;
```