1 Tabela

Código sequencial		
Contador	Sequencial	Paralelo
Task-clock (msec)	0,997 CPUs	1,962 CPUs
Cycles	1,660 GHz	2,345 GHz
L1-dcache-loads	605,586 M/sec	487,171 M/sec
Instructions (per cycle)	0,48	0,27
LLC-load-misses	4,67%	5,22%
Time Elapsed	4,63 s	2,93 s

Tabela 1: Comparação do sieve.c paralelo e sequencial no servidor parcode

Professor, assim como discutido pelo teams: tanto o servidor parcode, quanto o meu computador não conseguiram extrair todas as informações usando o perf stats -d(sendo essas informações o frontend cycles idle (ciclos ociosos na ULA) e o backend cycles idle (ciclos ociosos na busca de instrução)), logo essas informações foram substituidas pelo "Cycles"e o "L1-dcache-loads"respectivamente.

2 Código

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

int sieveOfEratosthenes(int n)
{

   omp_set_num_threads(2)
   // 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));

   int i, p;

   #pragma omp parallel for
   for (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
         #pragma omp parallel for
       for(i=p*2; i<=n; i += p)
             prime[i] = false;
       }
   }

   // count prime numbers
  #pragma omp parallel for reduction(+:primes)
  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;
}
```

3 Gargalos

Usando o código na seção anterior, podemos identificar os gargalos nas regiões da linha 23 e 30, onde tem a maior parte da execução do programa, e é onde tem um laço de repetição 'for' dentro de outro 'for'

Para melhorar o tempo de execução do programa, o ideal seria a utilização do scheduler, pois irá garantir que haverá um maior balançeamento de cargas entre as threads durante a execução do programa, permitindo com que um número menor de threads fiquem ociosas.