Tarefa10
Crivo

- Uso do Perf -

ANO

2020

Código

```
30
31  int sieveOfEratosthenes(int n)
32  {
33      // Create a boolean array "prime[0..n]" and initialize
34      // all entries it as true. A value in prime[i] will
35      // finally be false if i is Not a prime, else true.
36      int primes = 0;
37      bool *prime = (bool*) malloc((n+1)*sizeof(bool));
38      int sqrt_n = sqrt(n);
39
40      memset(prime, true,(n+1)*sizeof(bool));
41      #pragma omp parallel for schedule (dynamic)
42      for (int p=2; p <= sqrt_n; p++)
43      {
44          // If prime[p] is not changed, then it is a prime
45          if (prime[p] == true)
46          {
47              // Update all multiples of p
48              #pragma omp parallel for
49              for (int i=p*2; i<=n; i += p)
50              prime[i] = false;
51           }
52       }
53
54       // count prime numbers
55       #pragma omp parallel for reduction(+:primes) schedule (dynamic, 500)
56       for (int p=2; p<=n; p++)
57          if (prime[p])
58            primes++;
59
60       return(primes);
61  }
```

Tabela

Alcançando de eficiência : 43,93%

Gargalos

- As faltas na cache da aplicação paralela em relação a sequencial
- O IPC da aplicação paralela caiu em relação a sequencial

Sugestões

Primeiro aumentarei a entrada da aplicação de n=100000000 para n= 500000000.

A taxa de falta na cache diminuiu levemente em ambas aplicações(paralela e sequencial).

IPC de ambos diminuíram, bem levemente também.

Utilização da cpu teve uma melhora, mas poderia ser um pouco mais próximo de 4, que é numero de elementos de processamento que tivemos na sua execução.

Resolução

Proponho uma mudança política de escalonamento, de um jeito que torne estrategicamente mais balanceada a aplicação tendo assim melhor aproveitamento da cpu e aproveite também da localidade espacial dos dados evitando mais acesso a memória principal e secundária evitando assim as faltas na cache.

Modificando o Código e tentando alcançar o desejado na prática.

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

// count prime numbers
#pragma omp parallel for reduction(+:primes) schedule (dynamic, 100)
for (int p=2; p<=n; p++)
    if (prime[p])
      primes++;
```

```
 15.152.234.442      L1-dcache-loads           #  305,664 M/sec                    (25,02%)
     73.302.833      L1-dcache-load-misses     #    0,48% of all L1-dcache hits    (24,99%)
     61.060.920      LLC-loads                 #    1,232 M/sec                    (24,99%)
      4.267.553      LLC-load-misses           #    6,99% of all LL-cache hits     (24,99%)

   12,744792534 seconds time elapsed
```

A taxa de falta na cache foi de 11,37% para 6,99%(como imaginado), e tendo um speedup de 1,094 em relação a aplicação paralela anterior a esta.