Avaliando ao Desempenho do Crivo Paralelo - Ferramenta *Perf*

Lucas Santiago

Março de 2022

Código do Crivo de Erastóteles

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <string.h>
5  #include <math.h>
6  #include <omp.h>
7
8  int sieveOfEratosthenes(int n)
9  {
10    // Create a boolean array "prime[0..n]" and initialize
11    // all entries it as true. A value in prime[i] will
12    // finally be false if i is Not a prime, else true.
13    int primes = 0;
14    bool *prime = (bool*) malloc((n+1)*sizeof(bool));
15    int sqrt_n = sqrt(n);
16
17    memset(prime, true,(n+1)*sizeof(bool));
```

```
18
19   #pragma omp parallel for schedule(guided, 100)
20    for (int p=2; p <= sqrt_n; p++)
21    {
22        // If prime[p] is not changed, then it is a prime
23        if (prime[p] == true)
24        {
25            #pragma omp parallel for
26            for (int i=p*2; i<=n; i += p)
27             prime[i] = false;
28         }
29     }
30
31     // count prime numbers
32 #pragma omp parallel for schedule(static,100) reduction (+:primes)
33     for (int p=2; p<=n; p++)
34        if (prime[p])
35          primes++;
36
37     return(primes);
38 }
39
40 int main()
41 {
42     omp_set_num_threads(2);
43     int n = 100000000;
44     printf("%d\n",sieveOfEratosthenes(n));
45     return 0;
46 }
```

Valores obtidos pela ferramenta *perf*

	Crivo Sequêncial	Crivo Paralelo
CPU utilizada (%)	99,8%	126%
Ciclos ociosos na ULA	*	*
Ciclos ociosos na busca de instrução	*	*
Trocas de contexto	0,085 K/sec	0,085 K/sec
Page Faults	0,006 M/sec	0,005 M/sec
Instruções por ciclo	0,38 inst por cicle	0,34 inst por cicle
Taxa de falta na cache L3	1,419 M/sec	2,117 M/sec
Tempo total de execução	4,072374329 segundos	3,846906854 segundos

Tabela 1: Tabela de desempenho do Crivo de Erastóteles

Possíveis ideias de otimização do código

Olhando para esses valores é possível notar que a execução do código sequên- cial consegue fazer um grande uso de 99,8% da CPU utilizada, enquanto a versão paralela está tendo um ganho de aproximadamente 26% a mais na segunda CPU utilizada. Com esse número é fácil de notar uma falta de balanceamento de carga de trabalho entre as *threads*. A primeira consegue fazer um uso de quase 100% da CPU, enquanto a segunda tem apenas 1/4 de sua capacidade utilizada. Uma primeira otimização seria dividir melhor o trabalho entre as *threads*, usando uma *scheduler* melhor no grande for do crivo.

Apenas isso não será suficiente, outro ponto notado foi a gigantesca taxa de erro da cache L3. Um aumento em aproximadamente 49% da versão sequêncial para a paralela. Uma melhor distribuição das variáveis dentro da cache seria um ganho significativo. Para isso um melhor uso de variáveis do tipo *shared* e *private* teria um possível ganho.

*Elemento não presente na ferramenta *Perf*