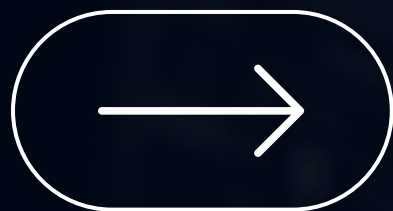


SISTEMAS OPERACIONAIS I

# Números. Primos. Com Multithread.

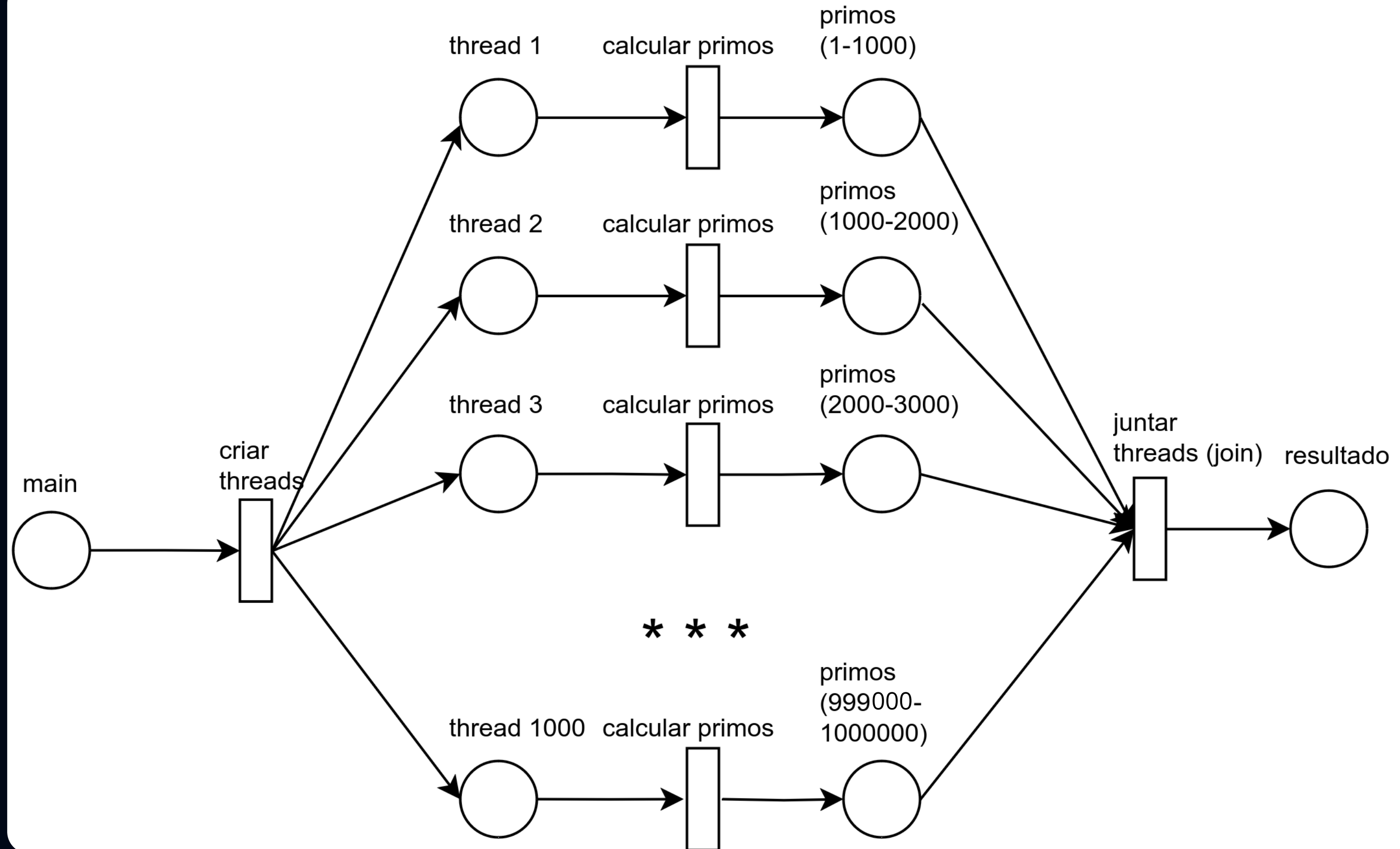


ÂNGELO GARCIA FERNANDEZ  
GUILHERME ALMEIDA DA LUZ

# Desafio

Calcular e imprimir os números primos entre 1 e 1M, utilizando 1 thread para cada faixa de 1k valores, resultando em 1k threads em processamento paralelo.





# O que é um número primo?

Um número primo pode ser definido como um Natural maior que 1 que possui apenas dois divisores: 1 e ele mesmo.

# Algoritmo usado

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  int main() {
5      for (size_t i = 2; i < 1000000; i++) {
6          bool isPrime = true;
7
8          for (size_t j = 2; j < i; j++) {
9              if (i % j == 0) {
10                 isPrime = false;
11
12                 break;
13             }
14         }
15
16         if (isPrime) {
17             printf("%zu\n", i);
18         }
19     }
20
21     return 0;
22 }
```

# Problema

Complexidade assintótica:  $O(n^2)$ , ou seja, leva muito tempo para calcular valores grandes

# Solução MultiThread

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <pthread.h>
5
6  #define TARGET 1000000
7  #define THREAD_RANGE 1000
8  #define THREAD_NUM (TARGET / THREAD_RANGE)
9
10 // Natural numbers greater than 1 that can only be divided by 1 and itself
11 void* getPrimeInRange(void* startPtr) {
12     // Get range
13     size_t start = *((int*) startPtr);
14     const size_t END = start + THREAD_RANGE;
15
16     // Assure the first thread (from 0 to 10000) will start from 2
17     if (start < 2) {
18         start = 2;
19     }
20
21     // Check every number in range
22     for (; start < END; start++) {
23         bool isPrime = true;
24
25         for (size_t i = 2; i < start; i++) {
26             if (start % i == 0) {
27                 isPrime = false;
28
29                 break;
30             }
31         }
32     }
```

```
32
33     if (isPrime) {
34         printf("%zu\n", start);
35     }
36 }
37
38 // Returns NULL because function returns pointer to void
39 return NULL;
40 }
41
42 typedef struct {
43     pthread_t thread;
44     size_t startValue;
45 } PrimeThread;
46
47 int main() {
48     PrimeThread threads[THREAD_NUM];
49
50     // Create threads
51     for (size_t i = 0, start = 0; i < THREAD_NUM; i++, start += THREAD_RANGE) {
52         threads[i].startValue = start;
53
54         int threadCreationStatus = pthread_create(
55             &threads[i].thread,
56             NULL,
57             getPrimeInRange,
58             &threads[i].startValue
59         );
60
```



```
46
47 int main() {
48     PrimeThread threads[THREAD_NUM];
49
50     // Create threads
51     for (size_t i = 0, start = 0; i < THREAD_NUM; i++, start += THREAD_RANGE) {
52         threads[i].startValue = start;
53
54         int threadCreationStatus = pthread_create(
55             &threads[i].thread,
56             NULL,
57             getPrimeInRange,
58             &threads[i].startValue
59         );
60
61         // Check creation error
62         if (threadCreationStatus != 0) {
63             printf("Error while creating thread %zu\n", i);
64
65             exit(-1);
66         }
67     }
68
69     // Join threads
70     for (size_t i = 0; i < THREAD_NUM; i++) {
71         pthread_join(threads[i].thread, NULL);
72     }
73
74     return 0;
75 }
```

# Testes

Para testar o resultado, fizemos nosso programa imprimir os primos um em cada linha e mandamos o resultado para o comando `wc` com a flag `-l` do Unix para contar as linhas. Sabendo que existem 78498 primos entre 1 e 1M, comparamos o resultado.

Para os testes de performance,  
utilizamos o comando time, também do  
Unix

Configuração da máquina:

Intel(R) Core(TM) i7-1065G7 (8) CPU @ 1.30GHz - 3.90GHz

openSUSE Tumbleweed 20240410 x86\_64

Kernel: 6.8.4-rc1-1-default

```
cloudywitt@nihil ~/Programming/github/AngeloGuilherme_ws_OS_RR_2024/testes main ± bash teste.sh
Testando paralelo 1
78498 ← Quantidade de números gerados

real    0m27.108s ← Tempo real de execução do script
user    3m34.833s
sys     0m0.013s
Testando paralelo 2
78498

real    0m27.087s
user    3m34.744s
sys     0m0.036s
Testando paralelo 3
78498

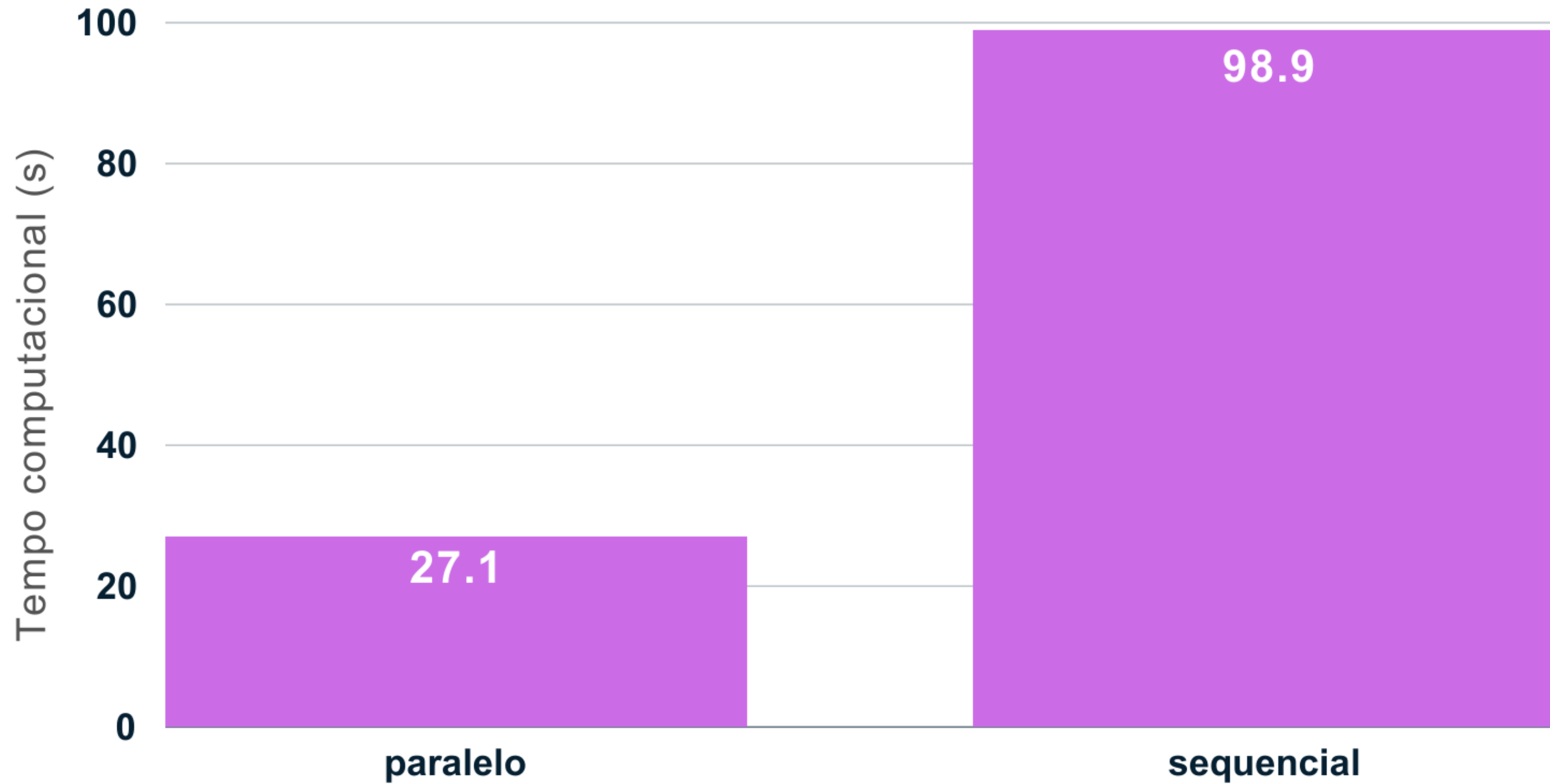
real    0m27.125s
user    3m34.773s
sys     0m0.033s
Testando sequencial 1
78498

real    1m38.907s
user    1m38.908s
sys     0m0.000s
Testando sequencial 2
78498

real    1m38.885s
user    1m38.796s
sys     0m0.003s
Testando sequencial 3
78498

real    1m38.931s
user    1m38.905s
sys     0m0.005s
```

# Algoritmo paralelo x sequencial



# Referências

<https://brasilescola.uol.com.br/matematica/numeros-primos.htm>

[https://stackoverflow.com/questions/5248915/execution-time-of-c-program#comment60164437\\_5249150](https://stackoverflow.com/questions/5248915/execution-time-of-c-program#comment60164437_5249150)