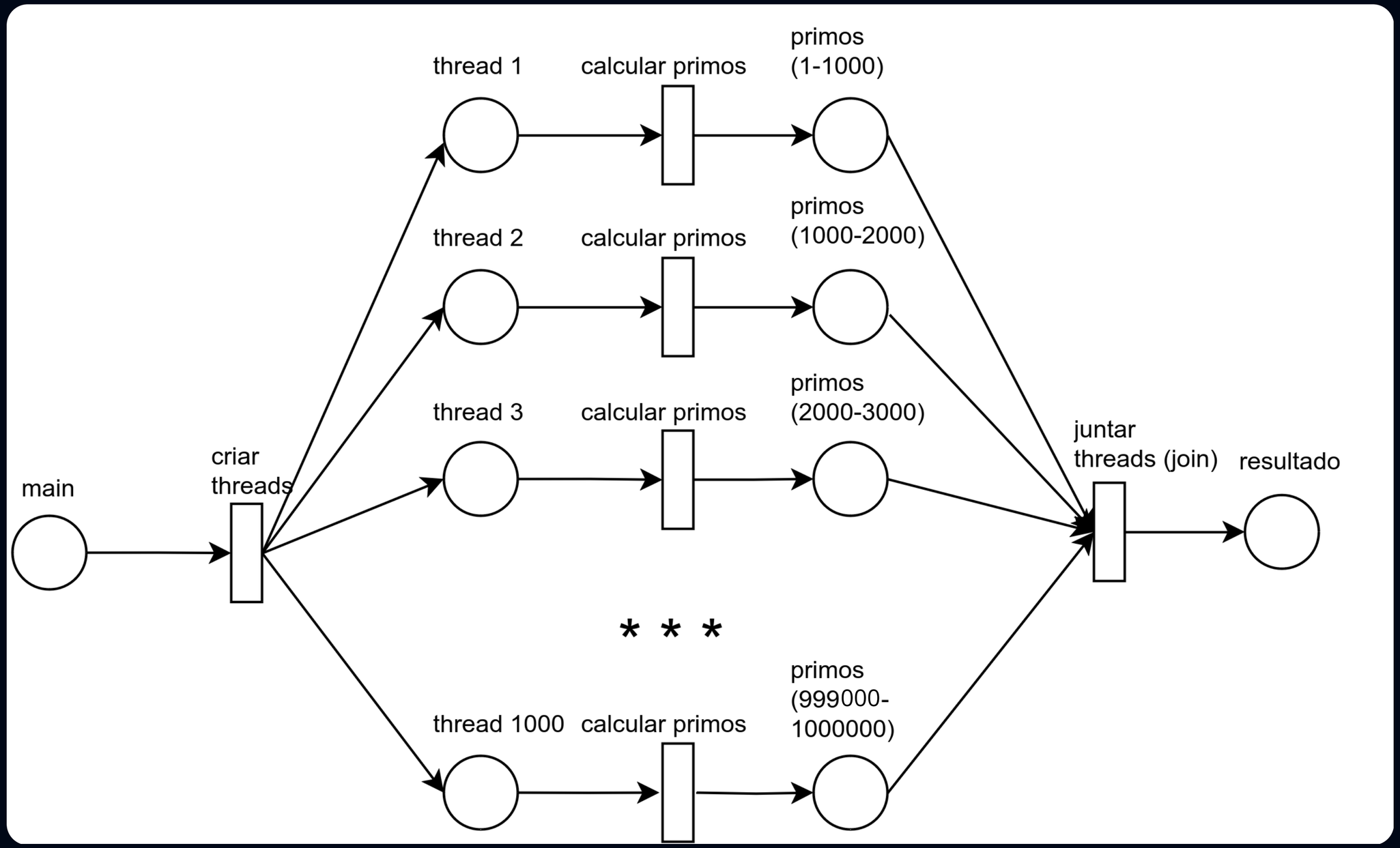
SISTEMAS OPERACIONAIS I

Números. Primos.

Com Multithread.

ÂNGELO GARCIA FERNANDEZ
GUILHERME ALMEIDA DA LUZ

Programa consiste em verificar e imprimir números primos entre 0 e 1M, utilizando 1 thread para cada faixa de 1k valores, resultando em 1k threads em processamento paralelo.


main
criar
threads
thread 1
calcular primos
primos
(1-1000)
thread 2
calcular primos
primos
(1000-2000)
thread 3
calcular primos
primos
(2000-3000)
* * *
thread 1000
calcular primos
primos
(999000-
1000000)
juntar
threads (join)
resultado

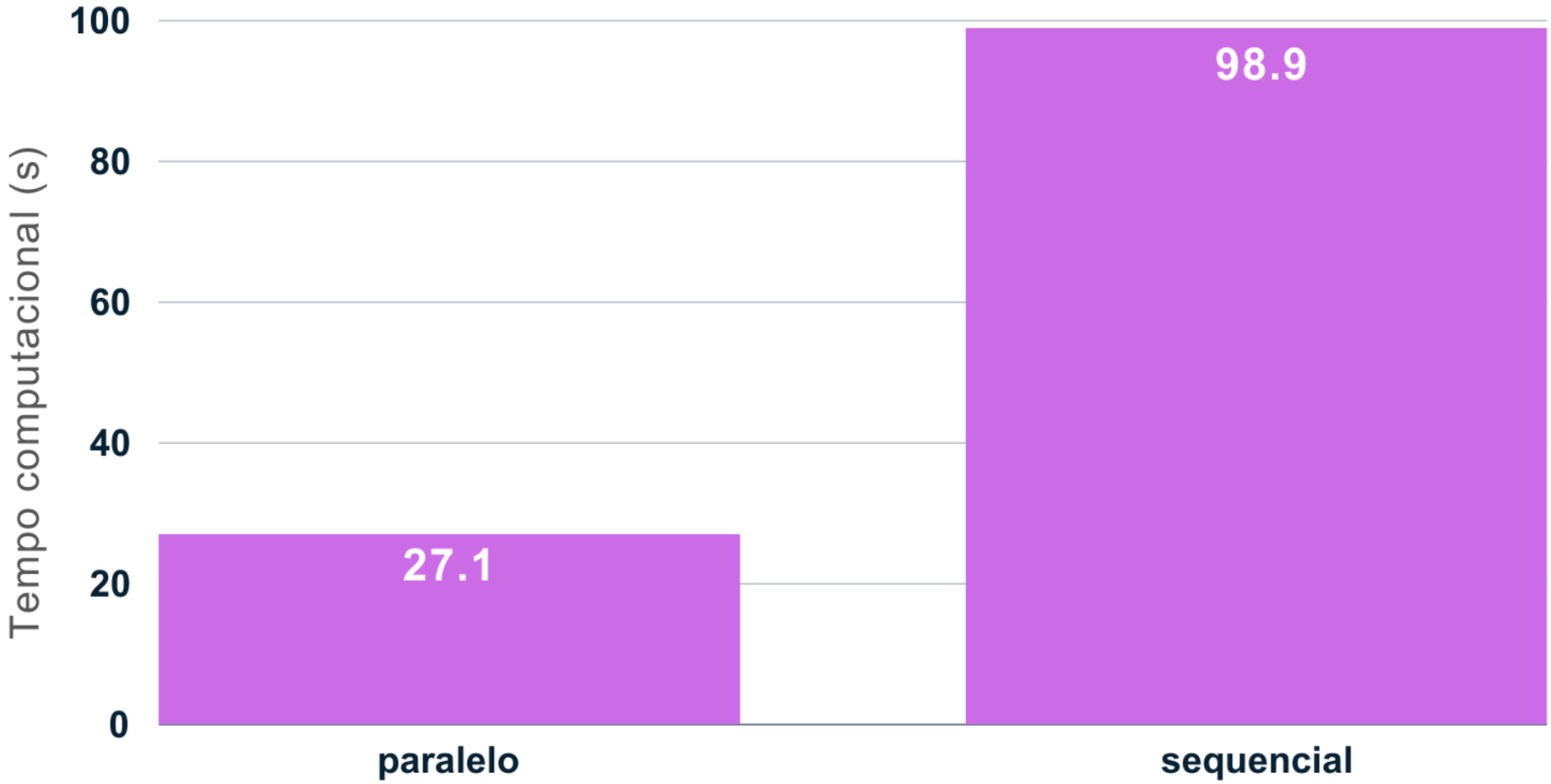
Solução MultiThread

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <pthread.h>

#define TARGET 1000000
#define THREAD_RANGE 1000
#define THREAD_NUM (TARGET / THREAD_RANGE)

// Natural numbers greater than 1 that can only be divided by 1 and itself
void* getPrimeInRange(void* startPtr) {
    // Get range
    size_t start = *((int*) startPtr);
    const size_t END = start + THREAD_RANGE;

    // Assure the first thread (from 0 to 10000) will start from 2
    if (start < 2) {
        start = 2;
    }

    // Check every number in range
    for (; start < END; start++) {
        bool isPrime = true;

        for (size_t i = 2; i < start; i++) {
            if (start % i == 0) {
                isPrime = false;

                break;
            }
        }

```

```
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  }
```

Algoritmo paralelo x sequencial



```
✘ 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
```

Solução SingleThread

```c
#include <stdio.h>
#include <stdbool.h>

int main() {
    for (size_t i = 2; i < 1000000; i++) {
        bool isPrime = true;

        for (size_t j = 2; j < i; j++) {
            if (i % j == 0) {
                isPrime = false;

                break;
            }
        }

        if (isPrime) {
            printf("%zu\n", i);
        }
    }

    return 0;
}
```