



# Análise de Números: implementação com Threads

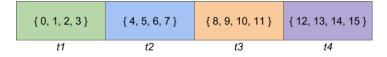
O trabalho consiste na implementação de um programa que analisa números e classifica-os como defectivos, abundantes ou perfeitos, usando abordagens sequencial e com threads. O programa recebe como parâmetros pela linha de comando (usar *argc* e *argv*): a quantidade de threads a serem criadas e o *worksize*total do programa (explicado a seguir). Deve ser feito o **teste de consistência** da entrada fornecida.

A classificação dos números deve ser realizada no intervalo de 1 até  $worksize_{total}$  (limite superior incluso). Considerando a *aliquot sum* de um número natural e positivo n como a soma de todos os seus divisores naturais próprios (i.e., excluindo ele mesmo), n pode ser classificado em:

- Defectivo: se aliquot\_sum(n) for menor que o próprio número;
- **Abundante:** se *aliquot\_sum(n)* for **maior** que o próprio número; ou
- **Perfeito:** se *aliquot\_sum(n)* for **igual** ao próprio número.

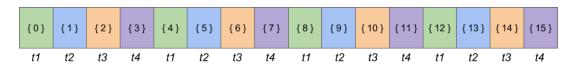
O programa deverá realizar esta classificação de três modos:

- a) **Sequencial:** implementada por uma função f onde o cálculo se dará pela simples classificação de cada número de 1 até *worksize*<sub>total</sub>, sem o uso de threads. A função f irá exibir a quantidade de números de cada tipo encontrados, retornando para a função *main* o tempo gasto para sua execução (este tempo deverá ser exibido pela função *main*!).
- b) Utilizando threads com distribuição por chunks: implementada por uma função f que irá dividir igualmente o worksizetotal entre cada thread (número passado por parâmetro). A divisão\* ocorre por intervalos, onde cada thread atua sobre uma faixa contínua de valores. Tomando como exemplo worksizetotal de 16 e 4 threads, a divisão de carga deve ser realizada da seguinte forma:



Após cada *thread* atuar sobre *worksize*<sub>thread</sub>, esta *thread* retorna a quantidade de números de cada tipo encontrado (receber o retorno no *join!*), após exibição dos resultados obtidos por cada *thread*, retorna-se para a função *main* o tempo gasto para a execução (conforme mencionado acima, este tempo deverá ser exibido pelo main!).

- \* Lembre-se de tratar casos onde a divisão do *worksize*<sub>total</sub> pelo número de *threads* não seja exata.
- c) **Utilizando threads com distribuição esparsa:** implementada por uma função *f* que irá dividir igualmente o *worksize*<sub>total</sub> entre cada *thread* (número passado por parâmetro). A divisão é feita de uma maneira esparsa (descontínua). Tomando como exemplo *worksize*<sub>total</sub> de 16 e 4 *threads*, a divisão de carga deve ser realizada da seguinte forma:







Após cada *thread* atuar sobre *worksize*<sub>thread</sub>, esta *thread* retorna a quantidade de números de cada tipo encontrado (receber o retorno no *join!*), após exibição dos resultados obtidos por cada *thread*, retorna-se para o main o tempo gasto para a execução (este tempo deverá ser exibido pela função *main!*).

#### Exemplo de output do programa:

\$ ./t2SO-aluno 4 100000

*	Sequencial:	[D] 75201	[A] 24795	[P] 4	[WTot] 100000
* * *	Thread 1: Thread 2: Thread 3: Thread 4: [TOTAL]	[D] 18805 18797 18789 18810 75201	[A] 6191 6203 6211 6190 24795	[P] 4 0 0 0 0	[WTh] 25000 25000 25000 25000 100000
* * *	Thread 1: Thread 2: Thread 3: Thread 4: [TOTAL]	[D] 24894 15951 24896 9460 75201	[A] 106 9048 104 15537 24795	[P] 0 1 0 3	[WTh] 25000 25000 25000 25000 100000

Tempo da classificação sequencial: xxx seq

Tempo da classificação com threads: xxx seg (distribuição por chunk)

Tempo da classificação com threads: xxx seg (distribuição esparsa)

-----

#### Análise de desempenho:

O programa deve realizar a análise de desempenho da implementação sequencial e com *threads* (chunk e esparsa), usando chamadas de sistema para medição de tempo *diretamente no código*.

Em geral, para utilizar funções de medição de tempo, pertencentes a bibliotecas como a time.h, é preciso especificar em qual ponto do fluxo de execução do código deve ser feita a primeira tomada de tempo (ponto t0) e em qual ponto deve ser feita a segunda tomada (ponto t1). Com isso, é possível calcular a diferença entre os tempos, obtendo o tempo decorrido para executar as instruções entre os pontos t0 e t1. Veja a seguir um exemplo de como utilizar a função  $clock\_gettime$  (biblioteca time.h) para obter o tempo decorrido entre dois pontos em segundos:





```
#include <stdio.h>
#include <time.h>

double diffTimeSec(struct timespec t0, struct timespec t1) {
    return ((double) t1.tv_sec - t0.tv_sec) + ((double) (t1.tv_nsec - t0.tv_nsec) * 1e-9);
}

double xxx() {
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);
    // ...
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    return diffTimeSec(start, end);
}

int main() {
    double tempo_xxx = xxx();
    printf("Tempo decorrido: %lf segundos\n", tempo_xxx);
    return 0;
}
```

Para o trabalho, certifique-se de posicionar as tomadas de tempo em locais onde essas reflitam o tempo da computação realizada, ou seja, o tempo até classificar os números, o qual é o objeto de comparação entre os modos sequencial e com *threads*.

<u>Dica</u>: para os modos que utilizam *threads*, a tomada de tempo *t0* pode ser realizada imediatamente antes da criação das *threads* e a tomada *t1* após o término das *threads* (i.e., imediatamente após o *join*), dessa forma o *overhead* para criação e finalização das *threads* estará incluso na análise.

Varie o número de *threads* e o *worksize*<sub>total</sub> para realizar os testes, abrangendo os seguintes cenários:

0 6! -	worksize <sub>total</sub>	Modos			
Cenário		Sequencial	Distribuição por chunks	Distribuição esparsa	
Α	100	padrão	4 threads	4 threads	
В	1.000		4 threads	4 threads	
С	10.000		4 threads	4 threads	
D	100.000		2, 4, 8 e 16 <i>threads</i>	2, 4, 8 e 16 <i>threads</i>	
E	500.000		2, 4, 8 e 16 threads	2, 4, 8 e 16 threads	

<u>Importante</u>: embora o número máximo de threads a serem executadas nos cenários seja 16, seu programa deve funcionar para qualquer número de threads!





Deverão ser realizadas, pelo menos, **5 execuções por modo com cada cenário** para que se obtenha uma amostra aceitável e fiel.

<u>Dicas</u>: a) você pode utilizar um *script* para otimizar os múltiplos testes e capturar os tempos de execução com cada modo; b) se estiver utilizando máquina virtual, certifique-se de que a VM não esteja limitada a 1 core/CPU.

Com base nos dados obtidos, realize uma avaliação comparativa sobre os resultados alcançados nos experimentos. Utilize de elementos que facilitem a discussão (média dos valores obtidos nos cenários, tabelas, gráficos, etc.). A avaliação de desempenho, em formato PDF, deve ser enviada juntamente com o programa desenvolvido.

### Observações:

- A interpretação da especificação faz parte da avaliação;
- O código deve estar organizado, indentado e obedecer criteriosamente à especificação;
- Certifique-se de compilar e testar exaustivamente o programa antes do envio;
- O arquivo NomeDoAluno-threads.zip, contendo os arquivo NomeDoAluno-threads.c
  e NomeDoAluno-threads.pdf (com a avaliação de desempenho), deve ser enviado
  pelo Moodle até às 20h do dia 23/07/2021. Não serão aceitos trabalhos entregues fora
  da plataforma Moodle;
- Trabalho individual;
- Trabalhos total ou parcialmente copiados receberão nota zero.