



POR: HELOISA ALVES

CRÉDITOS: PROFº GUILHERME GALANTE

SUMÁRIO



Breve descrição teórica

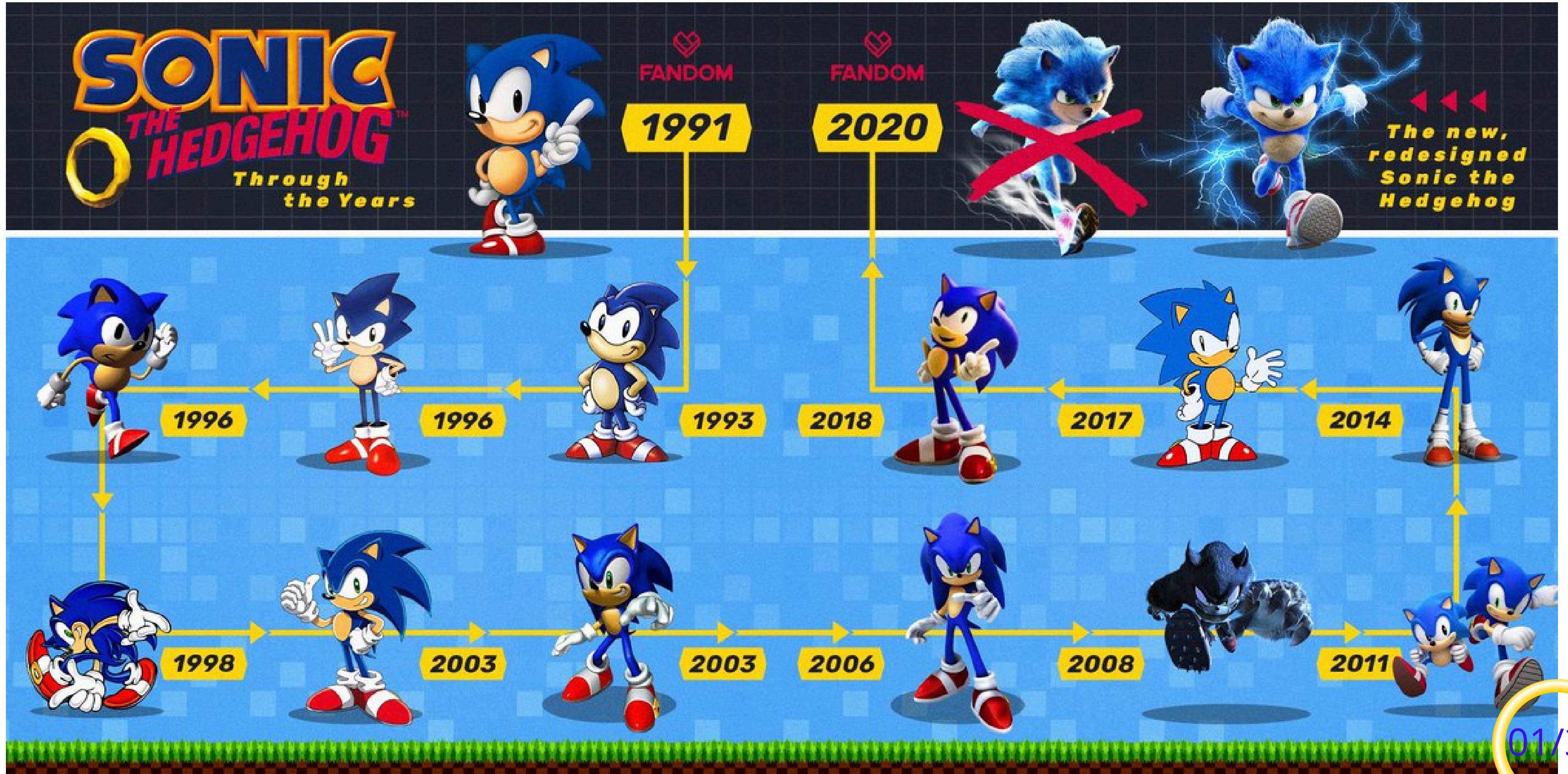


Exemplos práticos



Exercícios e Conclusões

MOTIVAÇÃO – REQUISITOS SEMPRE MUDANDO

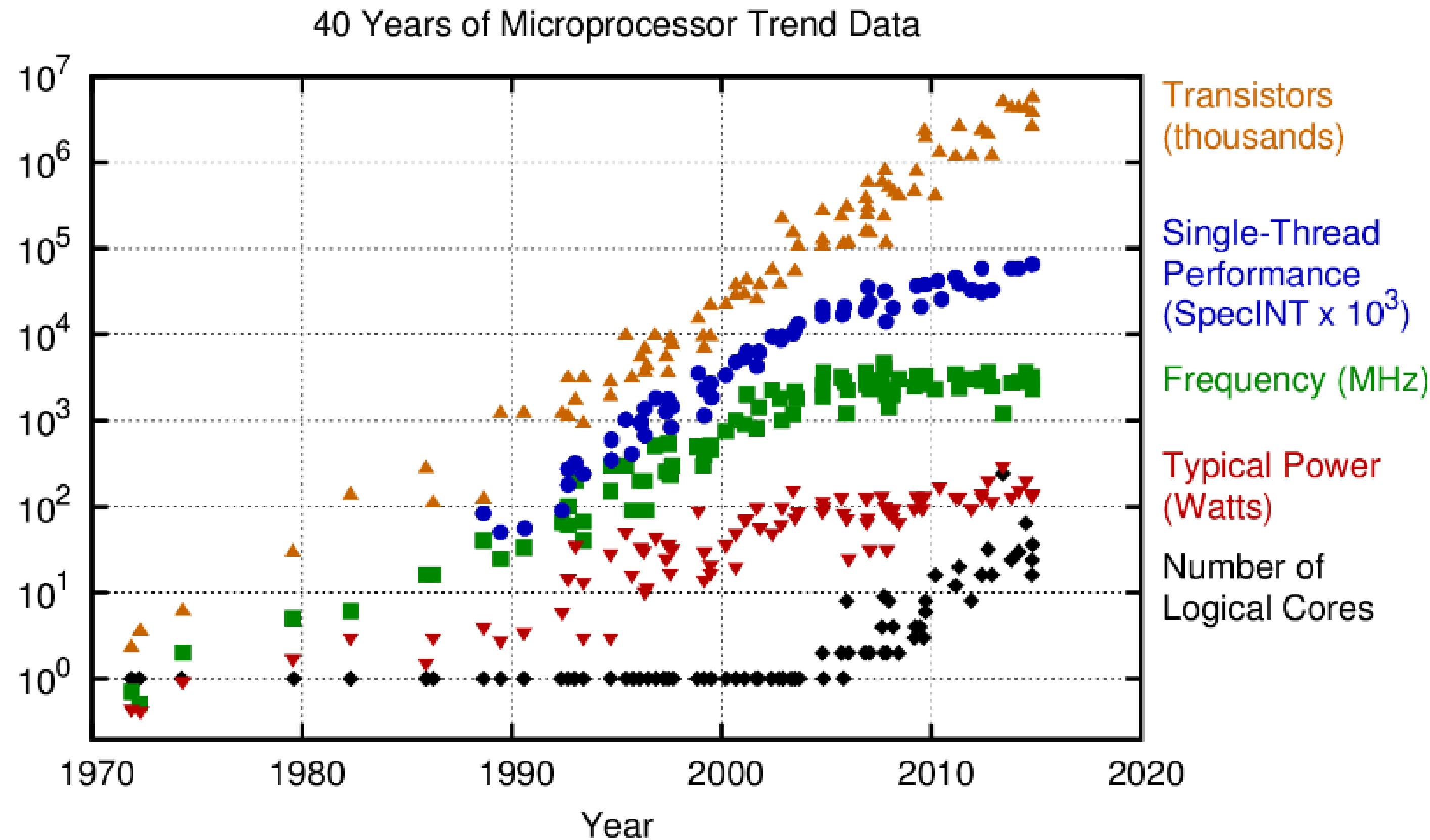


SOLUÇÕES

Em geral, temos 3 formas de resolver o problema de demanda computacional:

- Melhorar o algoritmo/implementação
- Comprar um computador mais rápido
- Computação paralela

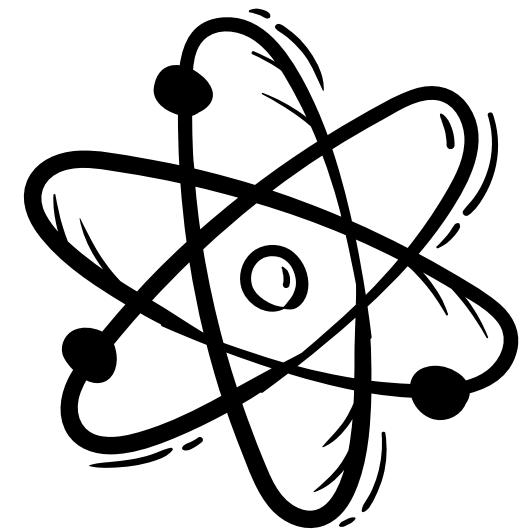
MULTICORE



COMPUTADOR MELHOR / ATUALIZADO

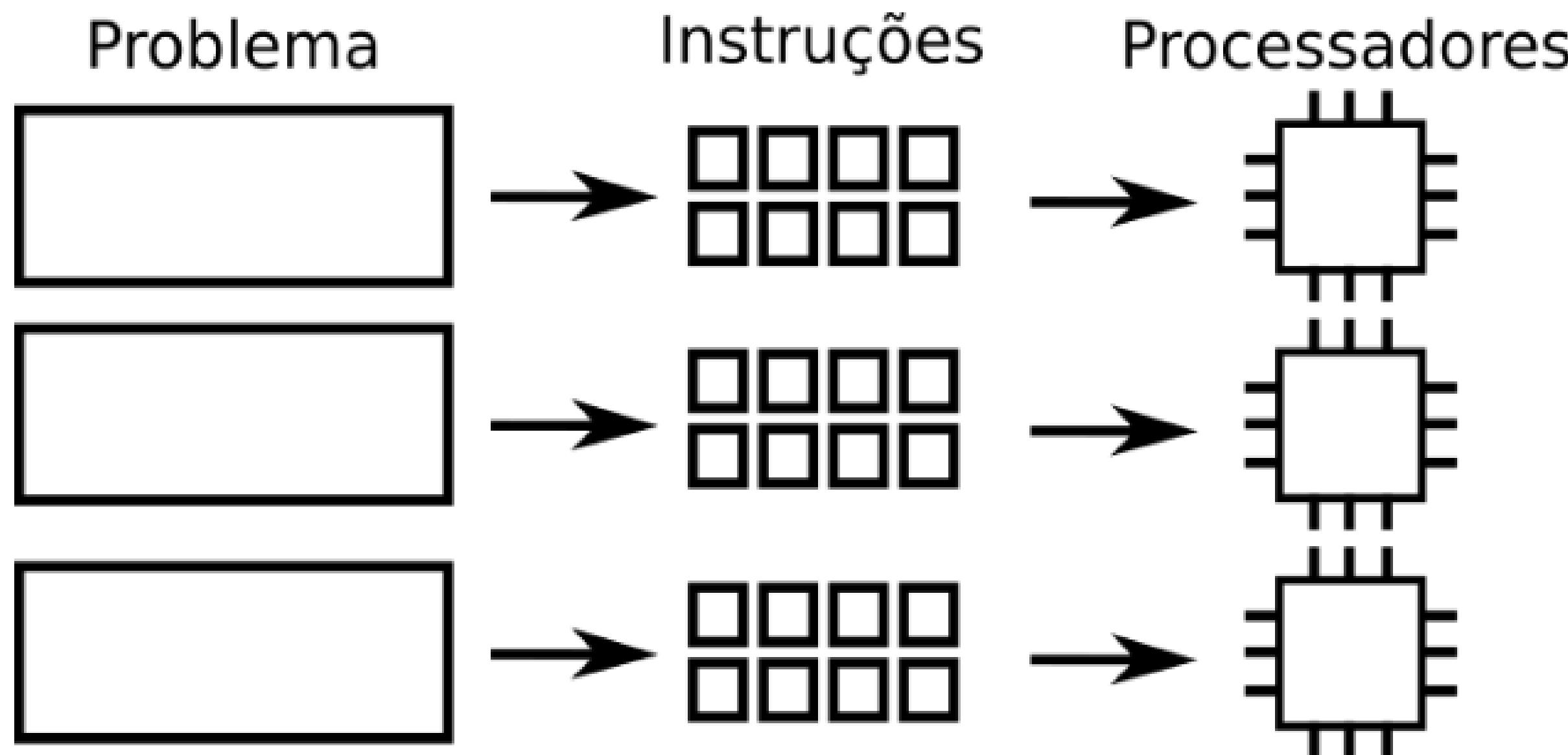
Uma pequena lição de física

- Transistores pequenos = processadores rápidos
- Processadores rápidos = aumento do consumo de energia
- Aumento do consumo de energia = aumento do calor
- Aumento do calor = processadores irreais



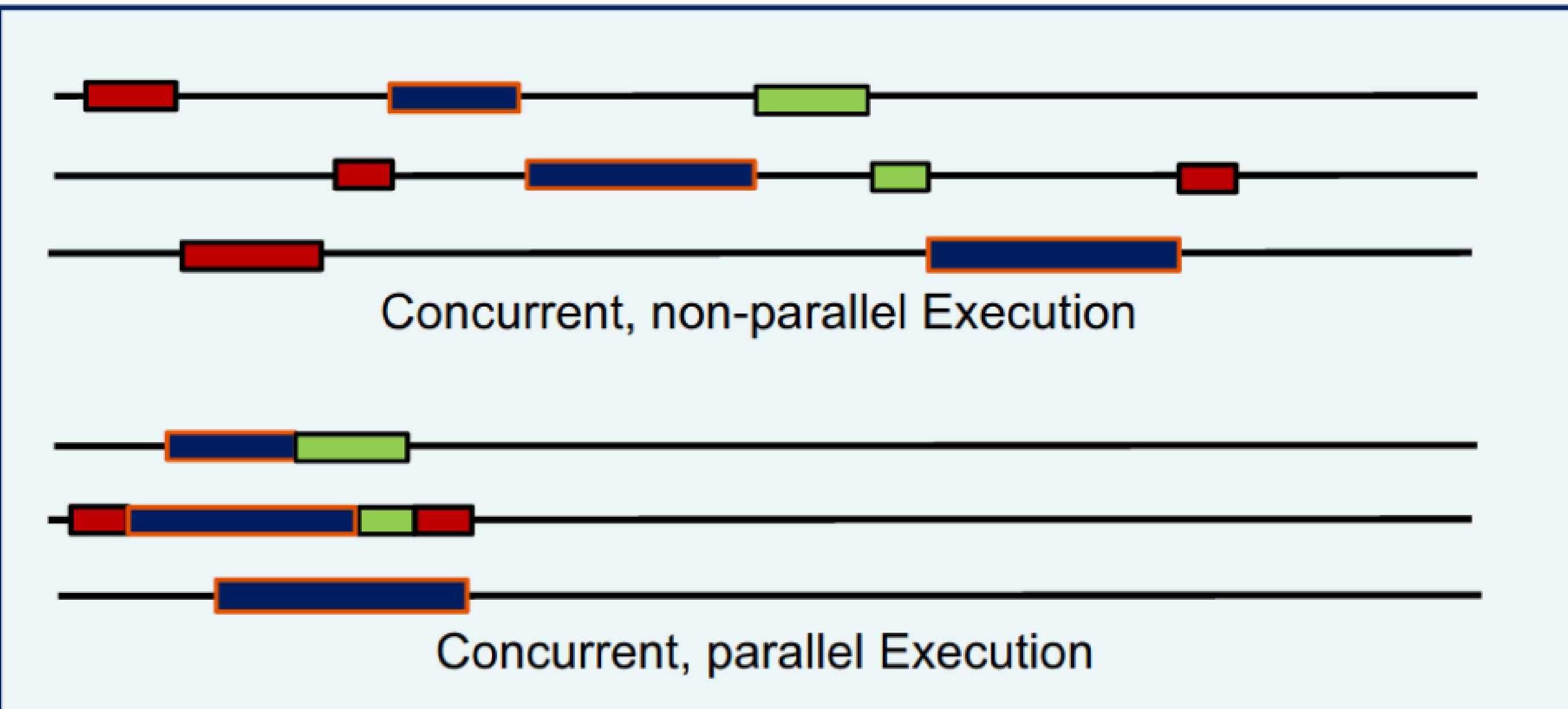
COMPUTAÇÃO PARALELA

Uso de diversas unidades de processamento ou computadores para a resolução de um problema em comum



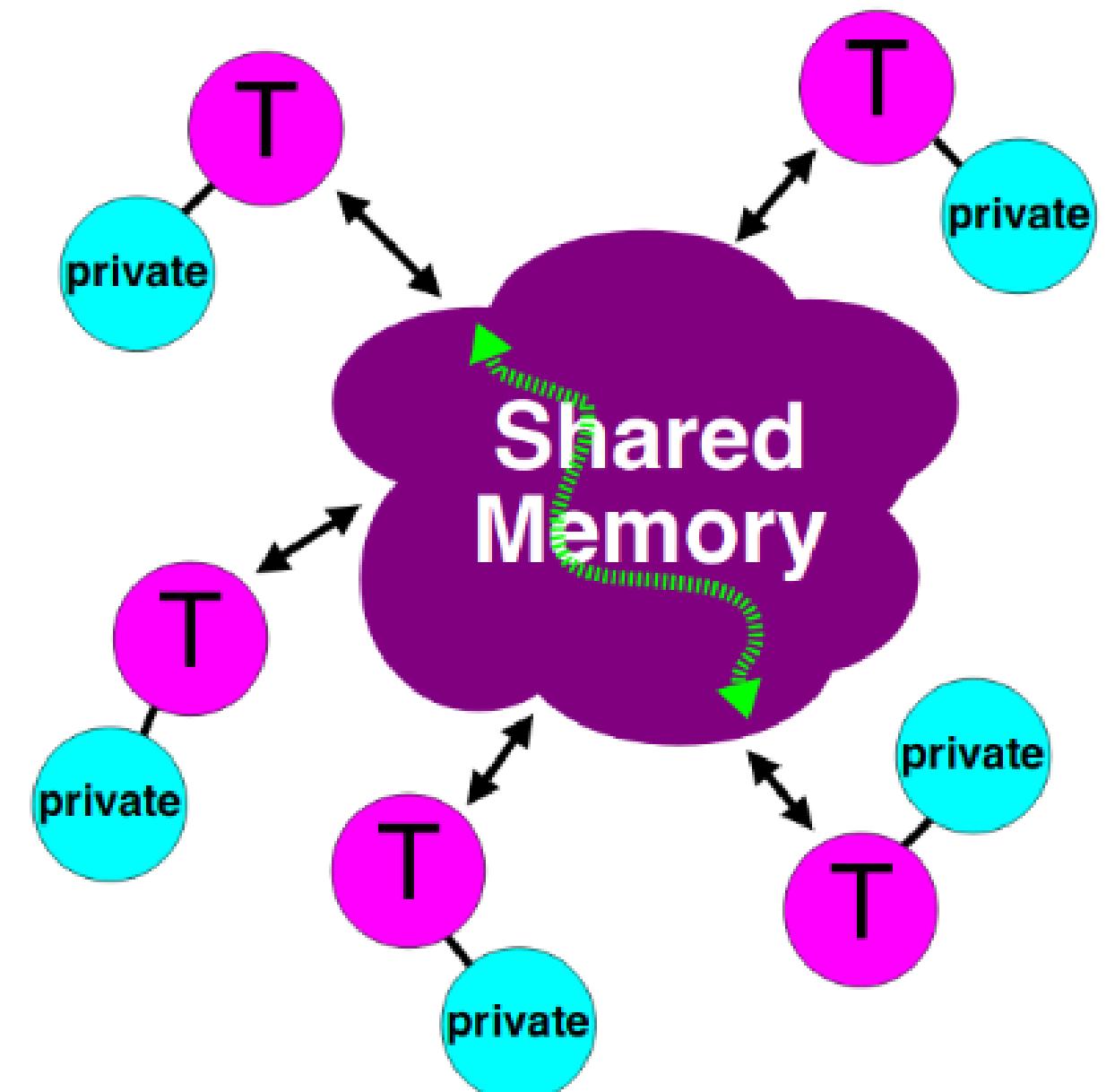
“Quando se tem uma grande carga para puxar é mais fácil colocar mais um boi do que criar um boi gigante”

CONCORRÊNCIA X PARALELISMO

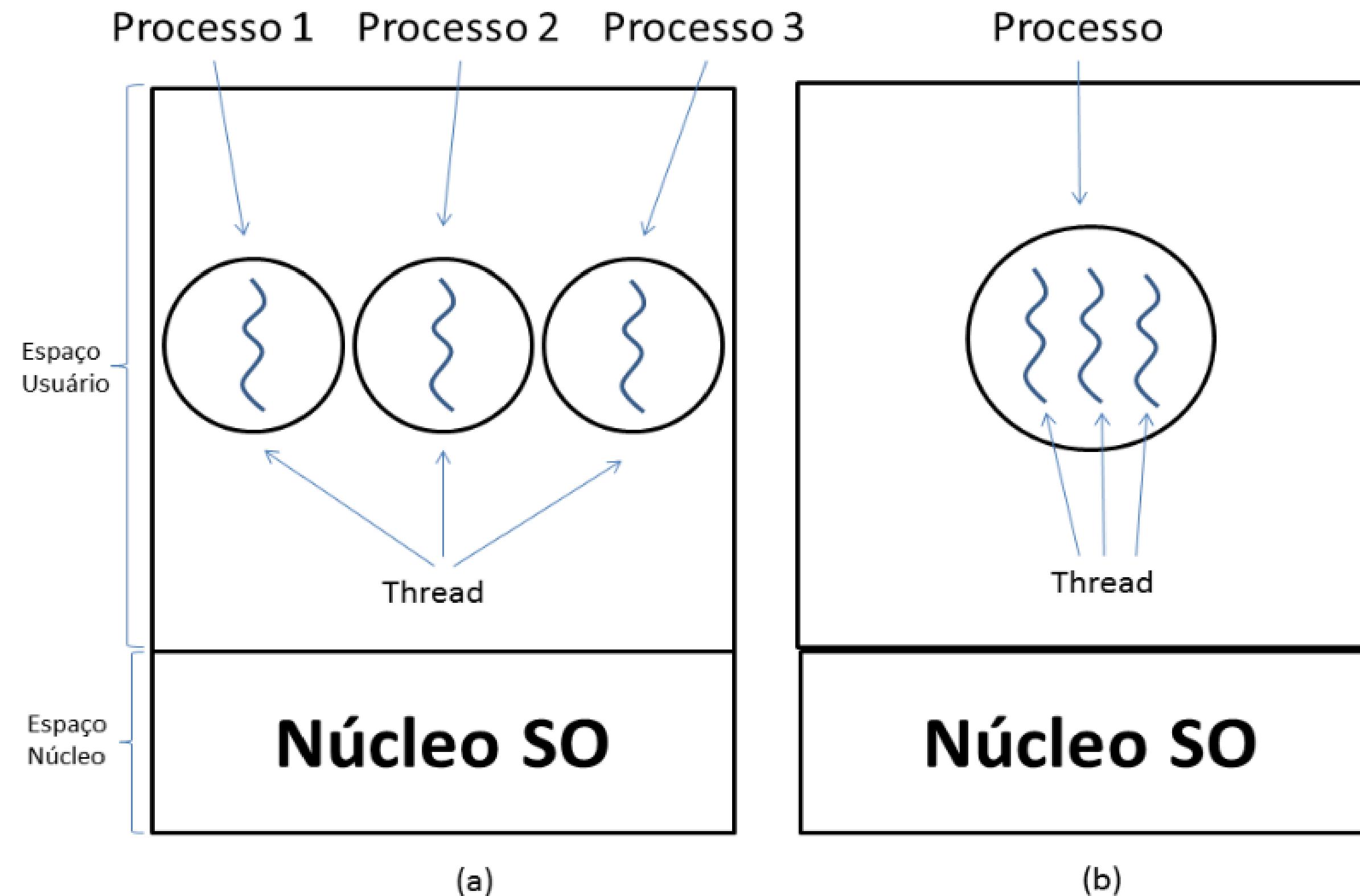


OPENMP

- API para programação em memória compartilhada
 - C/C++ e Fortran
- Baseia-se na criação de várias threads que compartilham o mesmo recurso
- Todas as threads têm acesso à mesma memória global compartilhada
- Threads tem a sua própria memória local privada
- Se comunicam implicitamente lendo e escrevendo variáveis compartilhadas

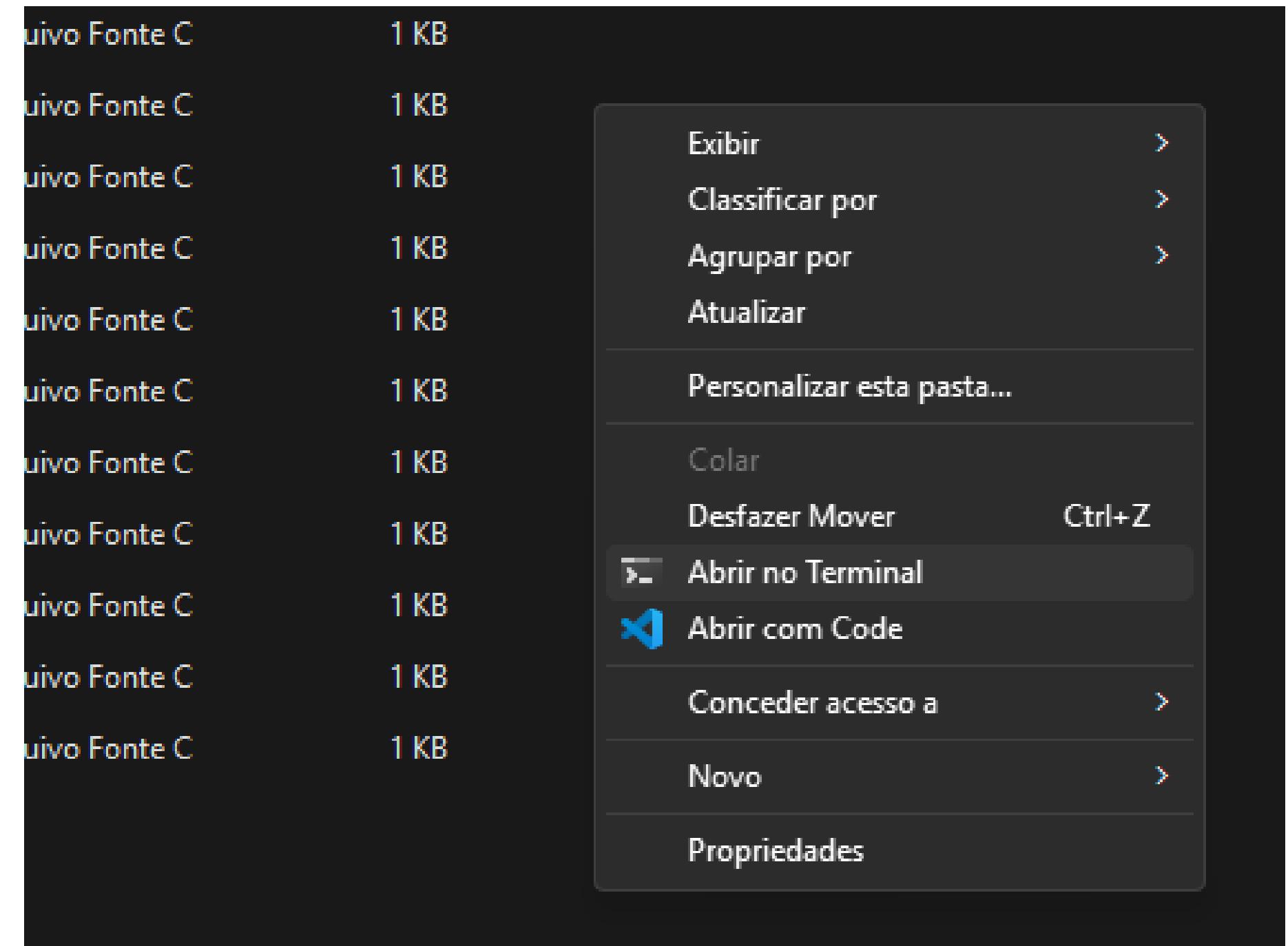
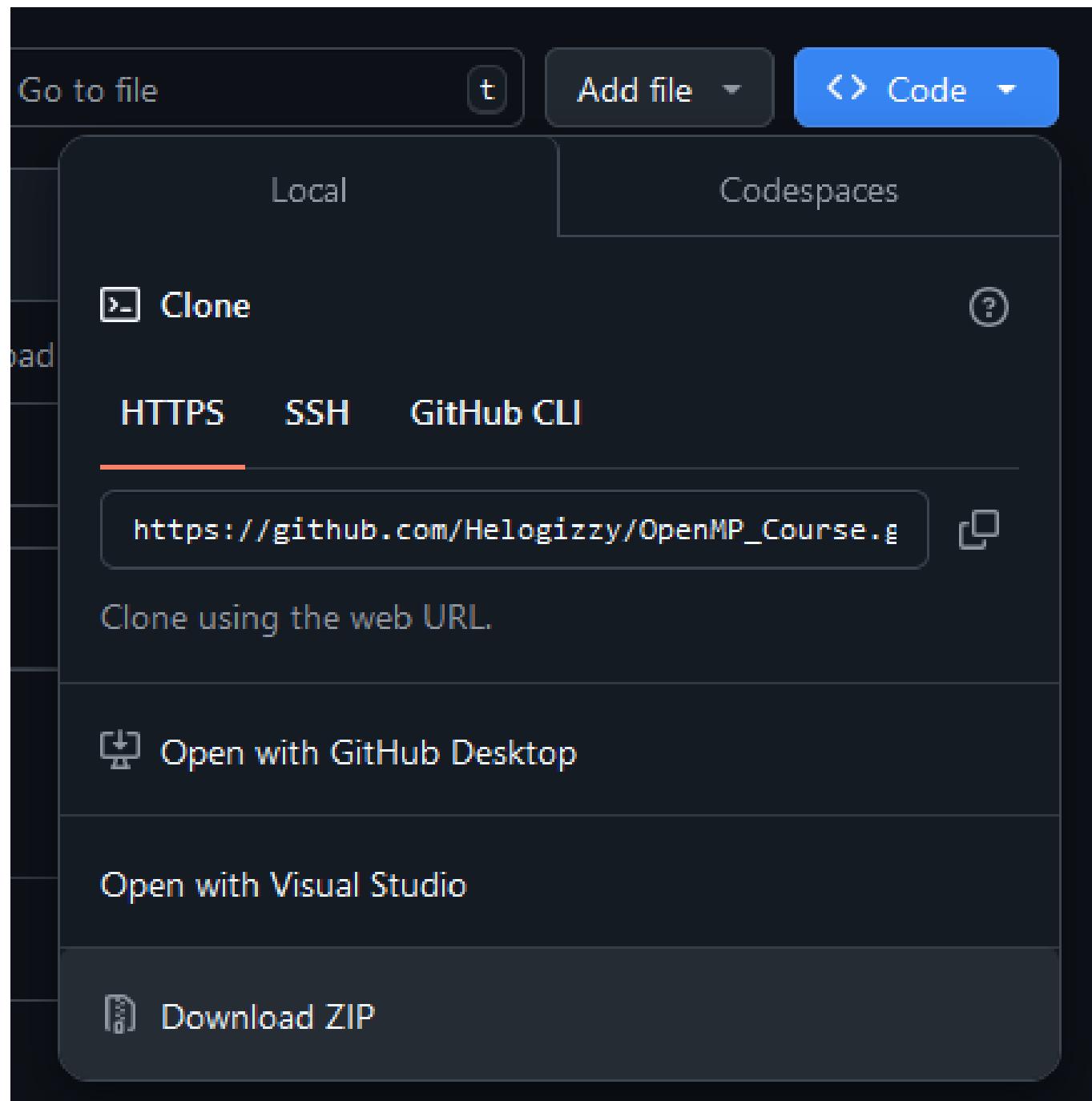


PROCESSOS X THREADS



< PRÁTICA >

ACESSO AOS CÓDIGOS FONTE



Repositório: https://github.com/Helogizzy/OpenMP_Course

COMPILAÇÃO E EXECUÇÃO

WSL

Compilação:
gcc exemplo1.c -fopenmp

Execução:
.a.out

VS CODE

Compilação:
gcc exemplo1.c -fopenmp

Execução:
.a.exe

```
in/codes$ gcc exemplo1.c -fopenmp
in/codes$ ./a.out
```

COMMENTS

```
\codes> gcc exemplo1.c -fopenmp
\codes> ./a.exe
```

EXEMPLO 1 – HELLO WORLD

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

int main()
{
    #pragma omp parallel
    {
        printf("Hello, world!\n");
    }
    return 0;
}
```

EXEMPLO 2 – INFORMAÇÃO DA IDENTIFICAÇÃO DE THREAD

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

int main()
{
    printf("--Fora da regiao paralela--\n");

    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        printf("Regiao Paralela - Hello, world da thread %d!\n",id);
    }

    printf("--Fora da regiao paralela--\n");
    return 0;
}
```

EXEMPLO 3 – INFORMAÇÃO DO Nº DE THREADS DISPONÍVEIS

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

int main()
{
    printf("--Fora da regiao paralela--\n");

    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int nt = omp_get_num_threads();
        printf("Regiao Paralela - Hello, world da thread %d - %d threads disponiveis\n",id, nt);
    }

    printf("--Fora da regiao paralela--\n");
    return 0;
}
```

EXEMPLO 4 – DEFININDO A QUANTIDADE DE THREADS

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

int main()
{
    printf("--Fora da regiao paralela--\n");

    omp_set_num_threads(3);
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int nt = omp_get_num_threads();
        printf("Regiao Paralela - Hello, world da thread %d - %d threads disponiveis\n",id, nt);
    }

    printf("--Fora da regiao paralela--\n");
    return 0;
}
```

EXEMPLO 4 B – DEFININDO A QUANTIDADE DE THREADS

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

int main()
{
    printf("--Fora da regiao paralela--\n");

    #pragma omp parallel num_threads(5)
    {
        int id = omp_get_thread_num();
        int nt = omp_get_num_threads();
        printf("Regiao Paralela - Hello, world da thread %d - %d threads disponiveis\n",id, nt);
    }

    printf("--Fora da regiao paralela--\n");
    return 0;
}
```

EXEMPLO 4 C – DEFININDO A QUANTIDADE DE THREADS

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

int main()
{
    #pragma omp parallel num_threads(5)
    {
        int id = omp_get_thread_num();
        int nt = omp_get_num_threads();
        printf("Regiao Paralela 1 - Thread %d de %d threads disponiveis\n",id, nt);
    }

    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int nt = omp_get_num_threads();
        printf("Regiao Paralela 1 - Thread %d de %d threads disponiveis\n",id, nt);
    }

    return 0;
}
```

Com quantas threads
cada seção paralela vai
executar se
OMP_NUM_THREADS=2?

EXEMPLO 5 – DIRETIVA BARRIER

```
int main()
{
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        printf("Hello World - Thread %d\n",id);

        #pragma omp barrier ←
        if (id == 0)
        {
            int nt = omp_get_num_threads();
            printf("Total de threads= %d\n",nt);

        }
    }

    return 0;
}
```

A diretiva *barrier* é utilizada para sincronizar todas as threads em um determinado ponto do código.

O que acontece se retirarmos a linha contendo a barreira?

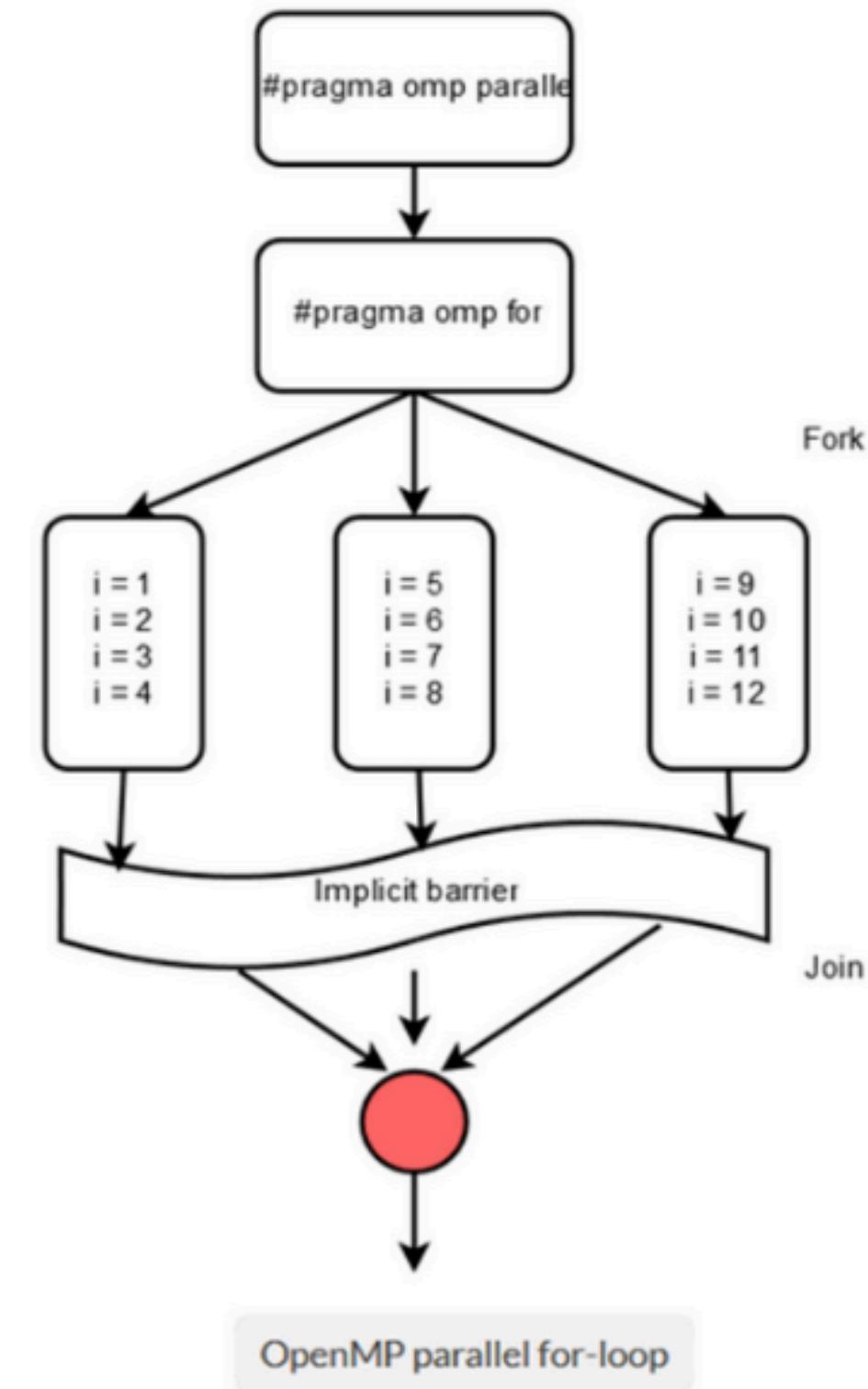
EXEMPLO 6 – DIRETIVA FOR

As iterações são distribuídas entre as threads do grupo criado na região paralela onde o laço se encontra.

```
int main()
{
    int vetor[12];

    inicializa(vetor); //isso será feito sequencial

    #pragma omp parallel num_threads(3)
    {
        #pragma omp for
        for(int i = 0; i < size; i++)
        {
            int id = omp_get_thread_num();
            vetor[i] = sqrt(vetor[i]);
            printf("Thread %d - posicao %i\n", id, i);
        }
    }
    return 0;
}
```



EXEMPLO 6 B – DIRETIVA FOR, OUTRA SINTAXE VÁLIDA

```
int main()
{
    int vetor[12];

    inicializa(vetor); //isso será feito sequencial

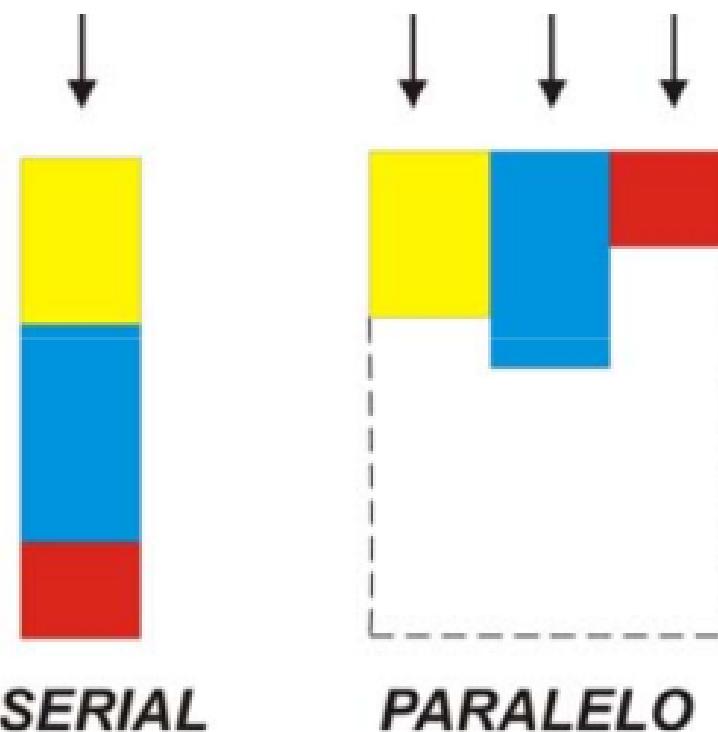
    #pragma omp parallel for num_threads(2)
    for(int i = 0; i < size; i++)
    {
        int id = omp_get_thread_num();
        vetor[i] = sqrt(vetor[i]);
        printf("Thread %d - posicao %i\n", id, i);
    }
    return 0;
}
```

Pode ser escrito dessa forma, se a região paralela possui apenas um construtor for

EXEMPLO 7 – DIRETIVA SECTIONS

```
int main ()  
{  
    #pragma omp parallel  
    {  
        int id = omp_get_thread_num();  
        #pragma omp sections nowait  
        {  
            #pragma omp section  
            printf("Section 1 - Thread %d\n", id);  
  
            #pragma omp section  
            printf("Section 2 - Thread %d\n", id);  
  
            #pragma omp section  
            printf("Section 3 - Thread %d\n", id);  
        } /*end of sections*/  
    }  
    return 0;  
}
```

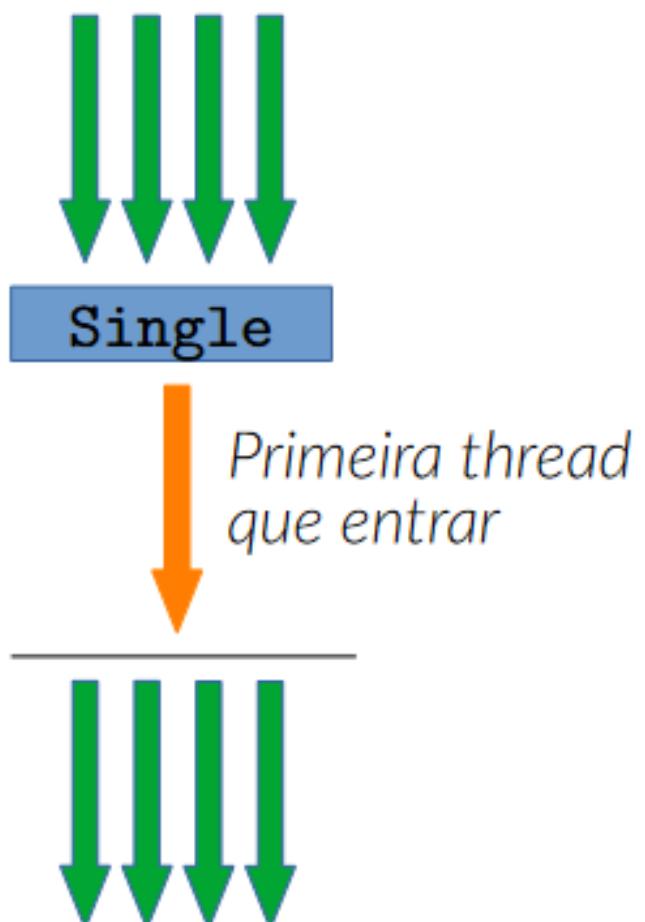
O construtor `sections` é utilizado para dividir tarefas para códigos que não possuem iterações. Dessa forma, cada thread irá executar um bloco de código diferente.
Paralelismo funcional



EXEMPLO 8 – DIRETIVA SINGLE

```
int main ()  
{  
    int i;  
    int a[N];  
  
    #pragma omp parallel  
    {  
        int id = omp_get_thread_num();  
  
        #pragma omp for  
        for(i=0; i<N; i++)  
            a[i] = id;  
  
        #pragma omp single  
        printf("Thread %d executou single\n", id);  
  
        #pragma omp for  
        for(i=0; i<N; i++)  
            a[i]=id*id;  
    }  
    return 0;  
}
```

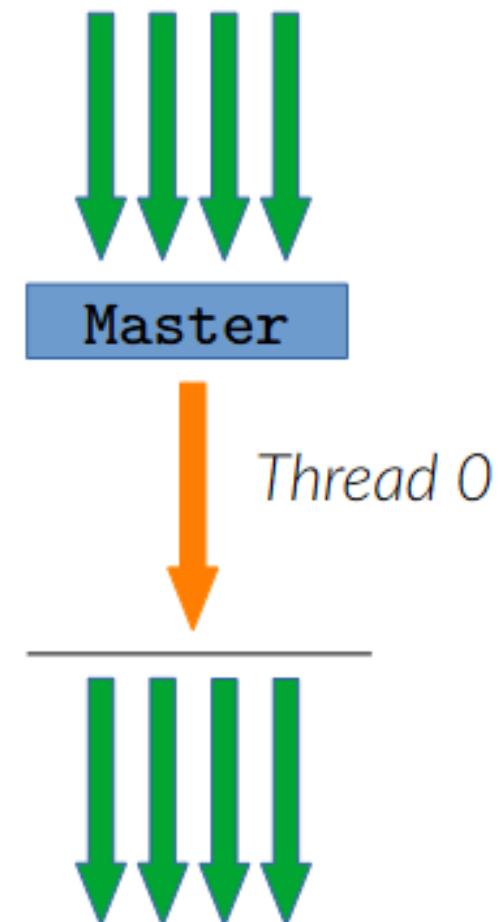
Indica que um bloco de código deve ser executado apenas por um thread (primeira thread que alcançar). Outros threads esperam até que o bloco seja executado (a menos que se use um nowait)



EXEMPLO 9 – DIRETIVA SINGLE(MASTER)

```
int main ()  
{  
    int a[N];  
  
    #pragma omp parallel  
    {  
        int id = omp_get_thread_num();  
  
        #pragma omp for  
        for(int i=0; i<N; i++ )  
            a[i] = id;  
  
        #pragma omp master  
        printf("Thread %d executou omp master\n", id);  
  
        #pragma omp for  
        for(int i=0; i<N; i++ )  
            a[i]=id*id;  
    }  
    return 0;  
}
```

Indica que um bloco deve ser executado apenas pelo thread master (thread 0). Outros threads pulam o bloco e continuam a execução



DIFERENÇAS ENTRE SINGLE E MASTER

Aspecto	Single	Master
Thread que executa	Qualquer thread pode executar	Apenas a thread mestre (thread 0)
Barreira implícita	Sim (por padrão), pode ser desativada	Não
Uso típico	Executar o código apenas uma vez	Executar algo que só a thread mestre deve realizar



< EXERCÍCIO >

CÓDIGO: SOMA.C

EXEMPLO 10 – CLÁUSULA SHARED

```
int main()
{
    int vetor[size];
    inicializa(vetor);

#pragma omp parallel for shared(vetor)
    for(int i = 0; i < size; i++)
    {
        int id = omp_get_thread_num();
        vetor[i] = vetor[i]+id;
    }

    for (int i = 0; i < size; i++)
    {
        printf("V[%d]: %d\n", i, vetor[i]);
    }

    return 0;
}
```

shared(var1,var2,...)

- Variáveis a serem compartilhadas entre todos os threads
- Threads acessam os mesmos locais de memória
- As diretivas que podem utilizar essa cláusula são as seguintes:
 - #pragma omp parallel
 - #pragma omp parallel for
 - #pragma omp parallel sections

EXEMPLO 11 – CLÁUSULA PRIVATE

```
int main()
{
    int i=0;
    int a=10;
    int id=0;

    #pragma omp parallel num_threads(2) private(id,a)
    {
        id=omp_get_thread_num();

        #pragma omp for
        for (i=0; i<n; i++)
        {
            a = a+1;
            printf("id=%d a=%d i=%d\n",id,a,i);
        }
    }
    printf("Qual o valor de a?: %d ",a);
    return 0;
}
```

private(var1,var2,...)

- Cada thread tem sua própria cópia das variáveis na execução do código paralelo
- As diretivas que podem utilizar essa cláusula são as seguintes:
 - #pragma omp parallel
 - #pragma omp parallel for
 - #pragma omp parallel section
 - #pragma omp single
 - #pragma omp for
 - #pragma omp sections

EXEMPLO 12 – CLÁUSULA FIRSTPRIVATE

```
int main()
{
    int i=0;
    int a=10;
    int id=0;

    #pragma omp parallel firstprivate(id,a)
    {
        id=omp_get_thread_num();

        #pragma omp for
        for (i=0; i<n; i++)
        {
            a = a+1;
            printf("id=%d \t a=%d \t for i=%d\n",id,a,i);
        }
        printf("Qual o valor de a?: %d ",a);
        return 0;
    }
}
```

firstprivate(var1,var2,...)

- Semelhante ao private
- As variáveis da lista entram na região paralela com os valores que possuíam antes de encontrar o construtor ao qual à cláusula está associada
- Pode ser usado com:
 - #pragma omp parallel
 - #pragma omp parallel for
 - #pragma omp parallel section
 - #pragma omp single
 - #pragma omp for
 - #pragma omp sections

EXEMPLO 13 – CLÁUSULA LASTPRIVATE

```
int main()
{
    int i=0;
    int a=10;
    int id=0;

    #pragma omp parallel num_threads(3) private(id)
    {
        id=omp_get_thread_num();

        #pragma omp for lastprivate(a)
        for (i=0; i<n; i++)
        {
            a = a+1;
            printf("id=%d a=%d i=%d\n", id,a,i);
        }
    }
    printf("Qual o valor de a?: %d ",a);
    return 0;
}
```

lastprivate(var1,var2,...)

- Semelhante ao private
- Permite que a variável saia da região definida pelo construtor com o último valor que foi atualizado para a mesma dentro da região
- Pode ser usado com:
 - #pragma omp parallel for
 - #pragma omp parallel section
 - #pragma omp for
 - #pragma omp sections

RESUMINDO

Cláusula	Descrição
Shared	Todas as threads compartilham as mesmas variáveis
Private	Cada thread tem sua própria cópia de uma variável, e as alterações feitas por uma thread não afetam as outras
Firstprivate	Cada thread tem sua própria cópia de uma variável, inicializada com o valor da variável original antes de entrar na região paralela
Lastprivate	A última thread a executar uma iteração do loop paralelo atualiza o valor da variável original com o valor da cópia da thread

EXEMPLO 14 – CLÁUSULA REDUCTION

```
...
#pragma omp parallel
{
    #pragma omp for private(i) reduction(+:dot)
    for (i = 0; i < n; i++)
    {
        dot += a[i]*b[i];
    }
}
...
...
```

reduction(op:var1,...)

- Especifica uma ou mais variáveis que são **privadas de cada thread** serão submetidas a uma operação de redução no final da região
- Pode ser usado com:
 - #pragma omp parallel for
 - #pragma omp parallel section
 - #pragma omp for
 - #pragma omp sections
 - #pragma omp parallel

EXEMPLO 14 – CLÁUSULA REDUCTION(2)

```
int main()
{
    int dot=0;
    float x[n];
    int y[n];
    int i, c;
    float d=0.0;

    for (i = 0; i < n; i++)
    {
        x[i]=(float)i;
        y[i]=i*2+1;
    }
    c=y[0];

    #pragma omp parallel for private(i) \
    reduction(min:c) reduction(max:d)
    for (i=0; i<n; i++)
    {
        if (y[i] < c)
            c = y[i];
        d = fmaxf(d,x[i]);
    }

    printf("Maior de x: %f \n", d);
    printf("Menor de y: %d \n", c);

    return 0;
}
```

Uso da cláusula **reduction** com min e max

EXEMPLO 15 – CLÁUSULA SCHEDULE

```
int main()
{
    int i;
    int a[n], b[n], c[n];

    //inicialização dos vetores

#pragma omp parallel num_threads(4)
{
    #pragma omp for schedule(guided, chunk_size) ← testar diferentes scheduler
    for (i = 0; i < n; i++)
    {
        c[i] = a[i]+b[i];
        printf("Thread %d -- iteração %d\n",omp_get_thread_num(),i);
    }
}

return 0;
}
```

QUANDO USAR CADA UM?

Escalonador	Descrição
Static	Quando o tempo de execução de cada iteração do loop é semelhante. Evita sobrecarga com o gerenciamento dinâmico.
Dynamic	Quando as iterações têm tempos de execução variados e você quer que as threads mais rápidas peguem mais trabalho enquanto as outras terminam suas tarefas.
Guided	Quando a carga de trabalho das iterações do loop diminui à medida que o loop avança.
Auto	Quando você não sabe qual estratégia de agendamento será mais eficiente e prefere que o OpenMP escolha automaticamente.

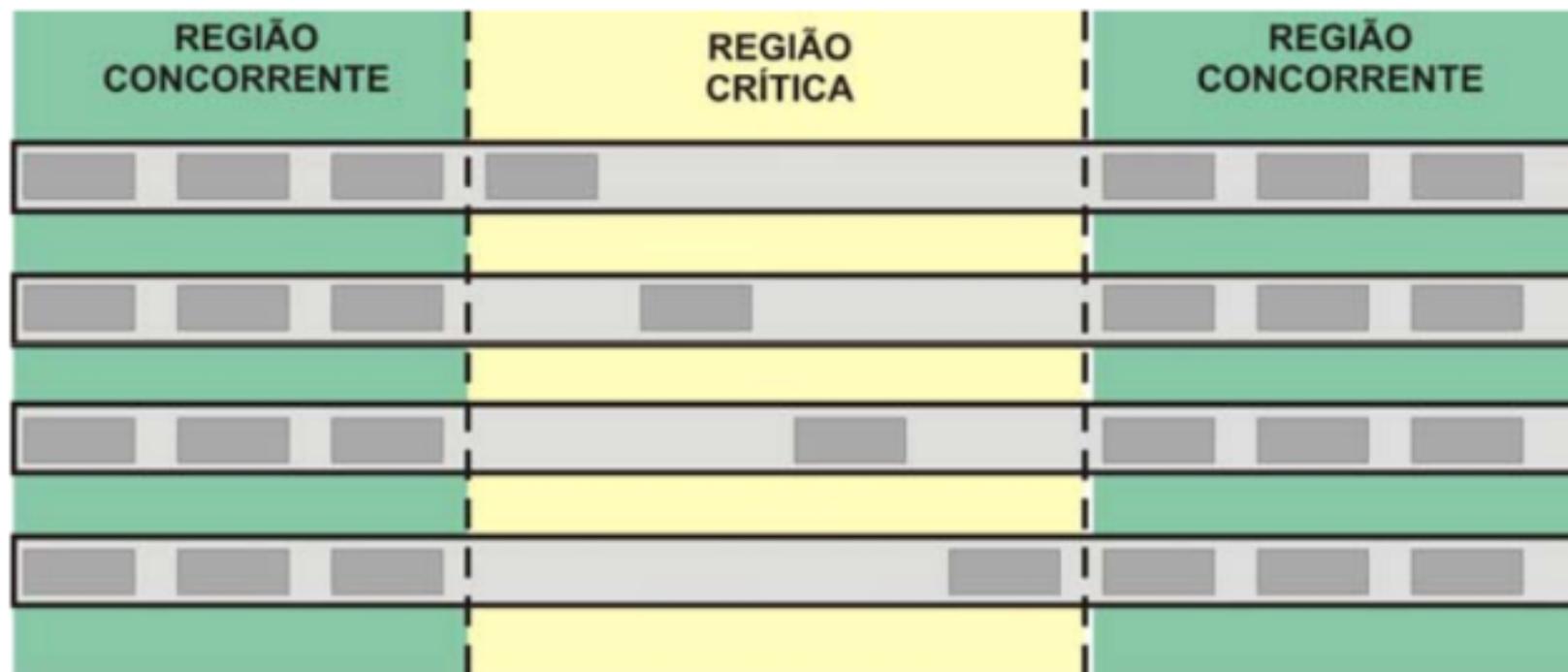


< EXERCÍCIO >

CÓDIGO: PRIMES.C

EXEMPLO 16 – DIRETIVA CRITICAL

O construtor **critical** restringe a execução de uma determinada tarefa a apenas uma thread por vez



```
int main()
{
    int i=0;
    int dot=0, aux_dot=0;
    int a[n];
    int b[n];

    for (i = 0; i < n; i++)
    {
        a[i]=1;
        b[i]=1;
    }

    #pragma omp parallel private(aux_dot)
    {
        #pragma omp for
        for (i = 0; i < n; i++)
        {
            aux_dot += a[i]*b[i];
        }

        #pragma omp critical
        dot += aux_dot;
    }

    printf("Dot: %d ", dot);
    return 0;
}
```

EXEMPLO 17 – DIRETIVA ATOMIC

```
#define n 10

int main()
{
    int ic, i;

    ic = 0;

    #pragma omp parallel shared(ic) private(i)
    for(i=0;i<n;i++)
    {
        #pragma omp atomic
        ic++;
    }

    printf("counter = %d\n", ic);

    return 0;
}
```

- Permite que certa região da memórias seja atualizada atomicamente, impedindo que várias threads accessem essa região ao mesmo tempo
- uma forma leve e especial de uma seção crítica
- aplica-se apenas a **uma única sentença**

*“Gyssg é tudo,
personal!”*

