

Programação Concorrente (IC/CCMN/UFRJ)

Construindo algoritmos concorrentes - Agosto de 2025

Profa. Silvana Rossetto

Definição de programação concorrente/paralela

- *Navarro et al.* definem a programação concorrente/paralela como a tarefa de resolver um problema de tamanho n dividindo o seu domínio em $k \geq 2$ ($k \in \mathbb{N}$) partes que deverão ser executadas em p processadores físicos simultaneamente
- Seguindo essa definição, um problema P_D com domínio D é dito paralelizável se for possível decompor P_D em k subproblemas:

$$D = d_1 \oplus d_2 \oplus \cdots \oplus d_k = \sum_{i=1}^k d_i$$

- Dependendo das características do problema, há diferentes formas de realizar essa decomposição

Paralelismo de dados e paralelismo de tarefas

- Dizemos que o problema P_D é um problema com **paralelismo de dados** se D é composto de elementos de dados e é possível aplicar uma função $f(\dots)$ para todo o domínio:

$$f(D) = f(d_1) \oplus f(d_2) \oplus \dots \oplus f(d_k) = \sum_{i=1}^k f(d_i)$$

- Por outro lado, se D é composto por funções e a solução do problema consiste em aplicar cada função sobre um fluxo de dados comum S , dizemos que o problema P_D é um problema com **paralelismo de tarefas**:

$$D(S) = d_1(S) \oplus d_2(S) \oplus \dots \oplus d_k(S) = \sum_{i=1}^k d_i(S)$$

Etapas do desenvolvimento de aplicações concorrentes

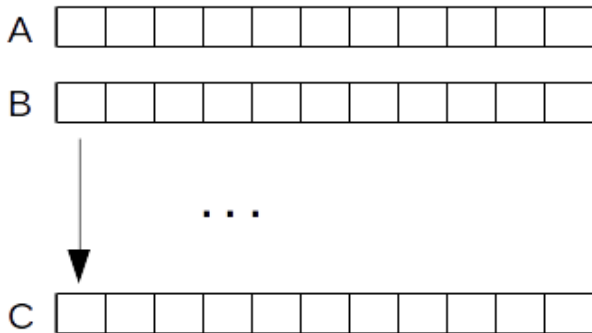
- 1 Dividir a aplicação em tarefas que podem ser submetidas à execução concorrente
- 2 Definir a estratégia de interação (comunicação e sincronização) entre os **fluxos de execução concorrentes**
- 3 Minimizar os custos associados à execução concorrente

- **1-Particionamento:** a tarefa (e o conjunto de dados associado a ela) é decomposta em subtarefas menores
- **2-Comunicação:** as demandas para coordenar a execução das subtarefas são levantadas
- **3- Aglomeração:** as subtarefas e estruturas de comunicação são avaliadas e, se necessário, agrupadas
- **4- Mapeamento:** cada subtarefa é designada para uma unidade de processamento (depende da arquitetura alvo)

*Single precision $A * X$ plus Y*

- Considere a operação $\mathbf{C} = \mathbf{A} * k + \mathbf{B}$, sendo A , B e C vetores de tamanho N e k um número
- Para encontrar o vetor de saída C , precisamos calcular cada elemento desse vetor fazendo:
$$\mathbf{C}[i] = \mathbf{A}[i] * k + \mathbf{B}[i], 0 \leq i < N$$

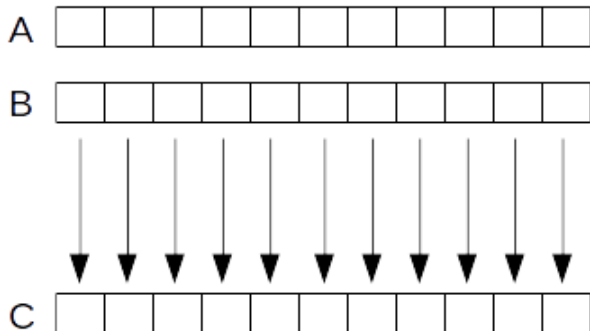
Algoritmo sequencial



Algoritmo sequencial

```
void somaVetoresSeq(const float a[], const float b[],  
                    float c[], float k, int n) {  
    for(int i=0; i<n; i++) {  
        c[i] = a[i] * k + b[i];  
    }  
}  
  
void main() {  
    float a[N], b[N], c[N];  
    //inicializa os vetores a e b  
    ...  
    somaVetoresSeq(a, b, c, 2.0, N);  
}
```


Algoritmo concorrente



Algoritmo concorrente inicial

```
float a[N], b[N], c[N];
void *calculaElementoVetor(void * args) {
    t_args *arg = (t_args*) args;
    int pos = arg->id;
    c[pos] = a[pos] * arg->k + b[pos];
}
void main() {
    //inicializa os vetores a e b
    short int nthreads = N;
    pthread_t tid[nthreads];
    for(int i=0; i<nthreads; i++) {
        pthread_create(&tid[i], NULL,
                      calculaElementoVetor, (void*) args);
    }
    for(int i=0; i<nthreads; i++)
        pthread_join(tid[i], NULL);
}
```

O que fizemos foi reduzir a complexidade de processamento de $O(N)$ para $O(1)$, assumindo que N fluxos de execução poderão estar ativos ao mesmo tempo

..mas, se não tivermos processadores em número suficiente para permitir que todos os fluxos de execução estejam ativos ao mesmo tempo?

- em geral, os ambientes de execução concorrente se encarregam de “enfileirar” as execuções
 - **qual seria o custo e a vantagem de se criar tantos fluxos?**
-
- então, vale a pena pensar em outras formas de dividir a tarefa principal?
 - quais seriam essas outras formas? (*propor alternativas*)

- **Tempo de processamento**
- **Aceleração:**

$$A(n, p) = T_s(n) / T_p(n, p)$$

- **linear, sublinear e superlinear**
- **Eficiência:**

$$E(n, p) = A(n, p) / p$$

- como a carga de trabalho extra do algoritmo concorrente cresce quando o número de fluxos de execução aumenta

- 1 Introdução à programação paralela em GPU para a implementação de métodos numéricos, D. Alfaro e S. Rossetto, Notas em Matemática Aplicada, vol 84, 2016
- 2 *A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures*, C. A. Navarro, N. Hitschfeld-Kahler e L. Mateu, Communications in Computational Physics, vol 15, 2014