Trabalho 2: Programação Paralela para Processador Many-core (GPU) Usando CUDA

- **Cálculo da distância de edição entre 2 sequências**:
  - Sequências de DNA: cadeias de bases nitrogenadas (A, C, G e T)
  - **Entradas**: Sequências S e R, $n = |S|$ e $m = |R|$, $n \leq m$
  - **Saída**: Distância de edição entre S e R
    (nº mínimo de operações de edição necessárias para transformar S em R)
  - **Operações de edição permitidas**:
    - Inserção de um símbolo na sequência
    - Remoção de um símbolo da sequência
    - Substituição de um símbolo da sequência por outro
- **Exemplo**:

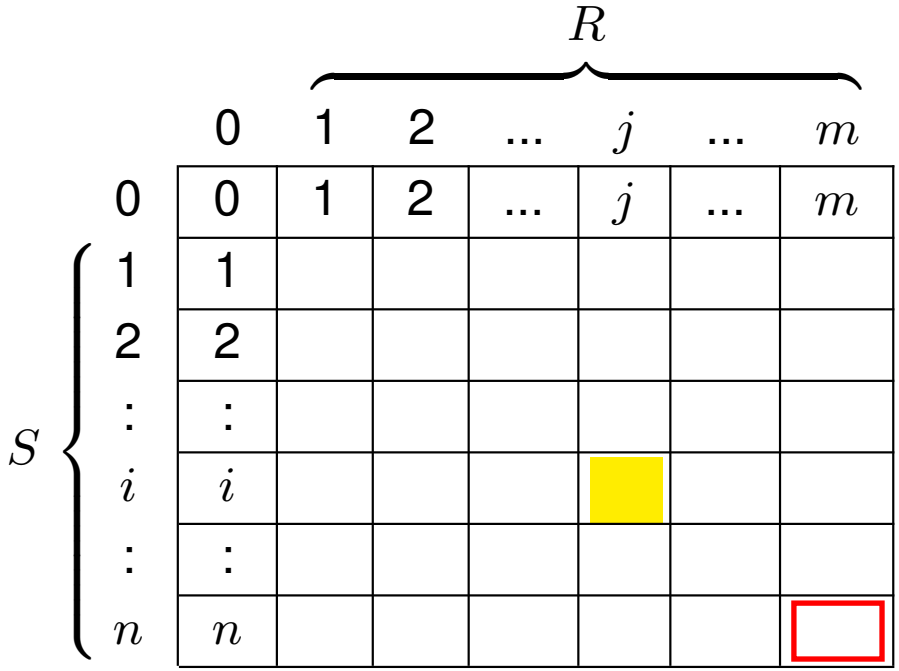
  $1\ 2\ 3\ 4$
  - **Entradas**: $S = ACTG$ e $R = AGCAGT$
  - **Saída**: Distância de edição entre S e $R = 3$
  - **Operações de edição**: Inserção de G após $S_1 = A$
    Substituição de $S_3 = T$ por A
    Inserção de T após $S_4 = G$

Algoritmo para Cálculo da Distância de Edição

- Usa técnica de programação dinâmica
- **Entradas**: Sequências S e R, $n = |S|$ e $m = |R|$, $n \leq m$
- **Calcula matriz** D: de tamanho $(n+1) \times (m+1)$
  - $D[i,j]$ = distância de edição entre prefixo $S_{1..i}$ e prefixo $R_{1..j}$
- **Saída**: Distância de edição entre S e $R = D[n,m]$
- **Inicialização da matriz** D:

$$D[i,0] = i, \quad \text{para } 0 \leq i \leq n$$

$$D[0,j] = j, \quad \text{para } 1 \leq j \leq m$$



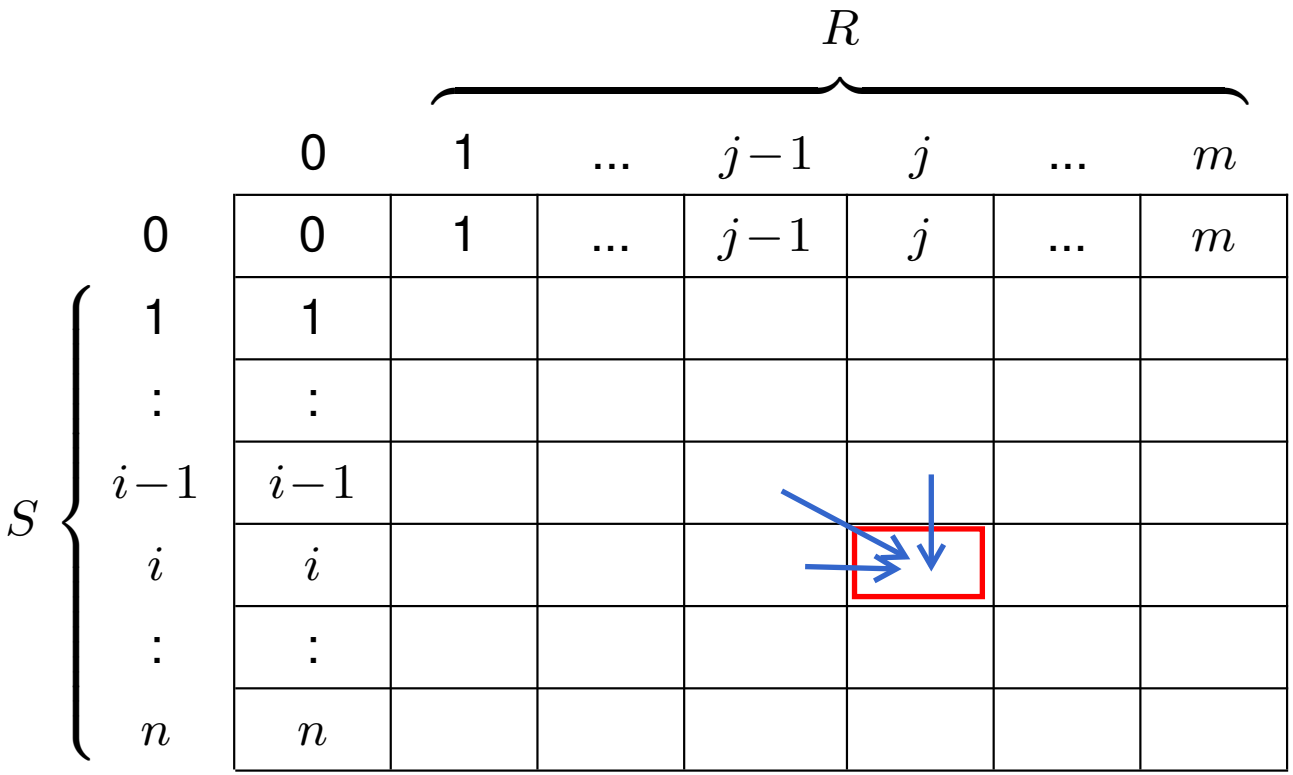
  - **Ideia da inicialização**:
    - Para transformar $S_{1..i}$ em uma sequência vazia: faz i remoções
    - Para transformar uma sequência vazia em $R_{1..j}$: faz j inserções

Algoritmo para Cálculo da Distância de Edição

- **Cálculo de** $D[i,j]$: para $1 \leq i \leq n$ e $1 \leq j \leq m$

$$D[i,j] = \min \begin{cases} D[i,j-1]+1 & \text{(inserção de } R_j \text{ na posição } i+1 \text{ de } S) \\ D[i-1,j]+1 & \text{(remoção de } S_i) \\ D[i-1,j-1]+t(S_i,R_j) & \text{(substituição ou correspondência)} \end{cases}$$

$$t(S_i,R_j) = \begin{cases} 1 \quad \text{para} \quad S_i \neq R_j & \text{(substituição de } S_i \text{ por } R_j) \\ 0 \quad \text{para} \quad S_i = R_j & \text{(correspondência entre } S_i \text{ e } R_j) \end{cases}$$



Algoritmo Sequencial para Cálculo da Distância de Edição

Input: Sequências S e R, com $n = |S|$ e $m = |R|$, $n \leq m$

Output: Distância de edição entre S e R

begin

- **for** $i := 0$ **to** n **do**
  - $D[i, 0] := i$
- **for** $j := 1$ **to** m **do**
  - $D[0, j] := j$
- **for** $i := 1$ **to** n **do**
  - **for** $j := 1$ **to** m **do**
    - **if** $S_i \neq R_j$ **then**
      - $t := 1$
    - **else**
      - $t := 0$
    - $D[i, j] := \min(D[i, j-1] + 1, D[i-1, j] + 1, D[i-1, j-1] + t)$
- **return** $D[n, m]$

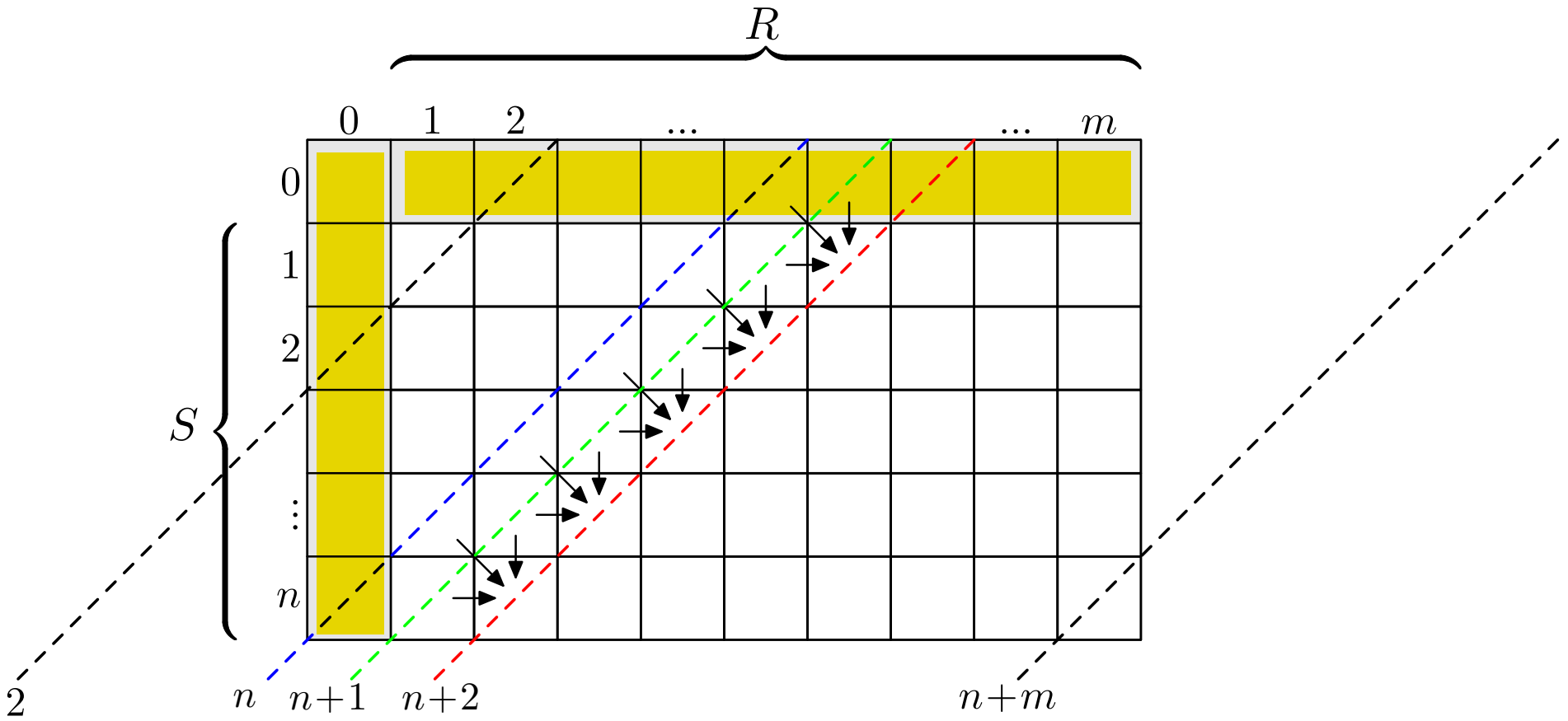
Exemplo

- **Entradas**: $S = ACTG$, $R = AGCAGT$, $n = 4$, $m = 6$
- **Saída**: Distância de edição entre S e $R = D[4, 6] = 3$
- **Matriz** D:

			R					
			A	G	C	A	G	T
		0	1	2	3	4	5	6
		0	1	2	3	4	5	6
S	A	1	0	1	2	3	4	5
	C	2	1	1	1	2	3	4
	T	3	2	2	2	2	3	3
	G	4	3	2	3	3	2	3

Ideia da Paralelização do Cálculo da Matriz D

- **Células de uma mesma anti-diagonal**: Podem ser calculadas em paralelo
- **Sucessivas anti-diagonais**: Devem ser calculadas sequencialmente
- **Supor que**: $n \leq m$



Algoritmo Sequencial para Cálculo da Distância de Edição

- **Percurso por anti-diagonal**: supondo que $n \leq m$

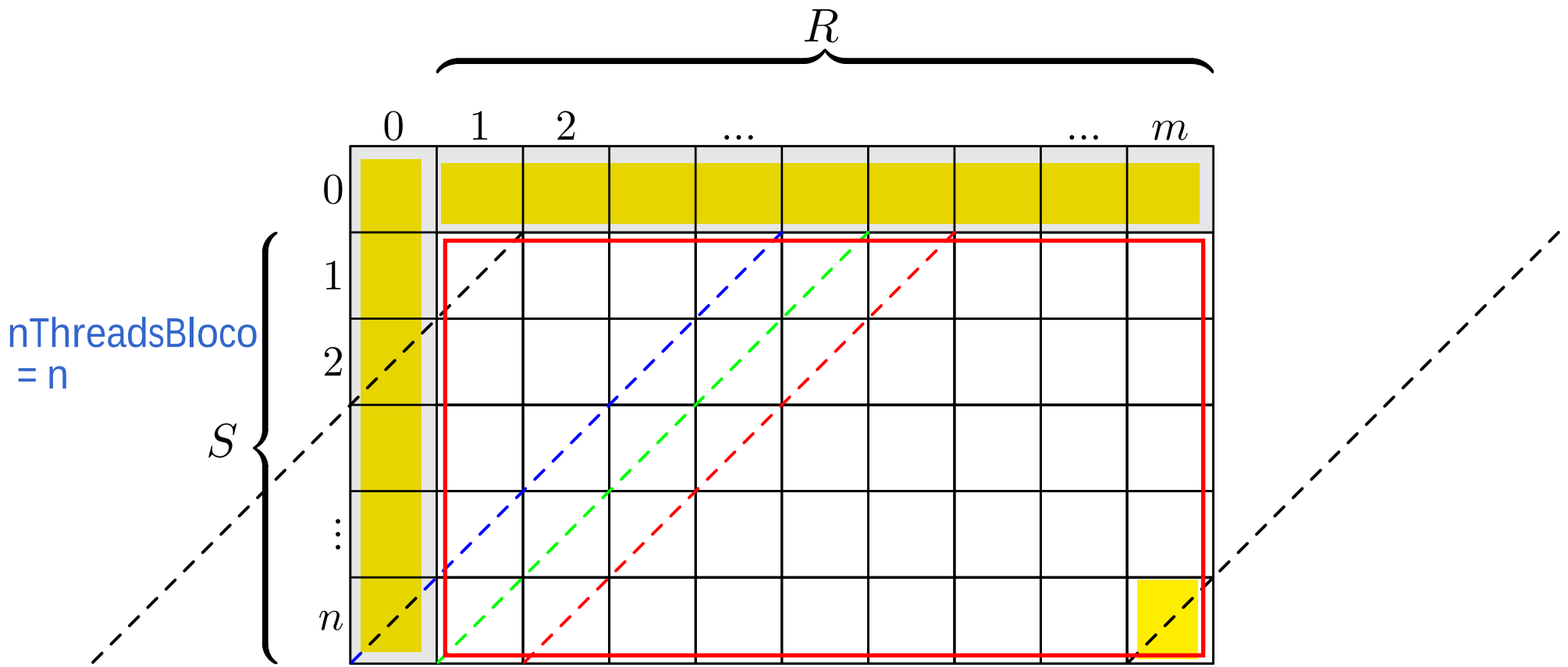
```
nADiag = n + m - 1;  // Número de anti-diagonais
tamMaxADiag = n;  // Tamanho máximo (número máximo de células) da anti-diagonal
// Para cada anti-diagonal (anti-diagonais numeradas de 2 a nADiag + 1)
for (aD = 2; aD <= nADiag + 1; aD++)
{   #pragma omp parallel for private(i,j)
    // Para cada célula da anti-diagonal aD
    for (k = 0; k < tamMaxADiag; k++)
    {
        i = n - k;  // Calcula índices i e j da célula (linha e coluna)
        j = aD - i;
        if (j > 0 && j <= m)  // Se é uma célula válida
        {
            t = (s[i] != r[j] ? 1 : 0);
            a = d[i][j-1] + 1;
            b = d[i-1][j] + 1;
            c = d[i-1][j-1] + t;
            // Calcula d[i][j] = min(a, b, c)
            d[i][j] = ...
        }
    }
}
```

Sugestões para o Desenvolvimento do Programa Paralelo (1)

- **Inicialmente**: Entender programa sequencial
  - Percurso por linhas
  - Percurso por anti-diagonais
- **Desenvolver solução inicial**: após entender programa sequencial
  - Supor que: $n \leq m$ e $n \leq$ maxThreadsBloco
  - Desenvolver programa CUDA com um único bloco de threads
  - Sincronização entre threads do bloco: `__syncthreads()`
  - Usar apenas memória global da GPU
  - **Transferências entre host e GPU**:
    - **Entradas**: Host $\Rightarrow$ GPU
      - Vetores das sequências S e R
      - Matriz D: precisa ser transferida inteira ?
    - **Saída**: GPU $\Rightarrow$ Host
      - Apenas $D[n, m]$
  - **Manipulação de matriz**: Linearizada como vetor
    - Ver ex17.cu (soma de matrizes em CUDA) d[i][j] equivale d[i*nColunas + j]

Ideia da Solução CUDA Inicial

- Supor que: $n \leq m$ e $n \leq$ maxThreadsBloco
- Um único bloco de threads
- Sincronização entre threads do bloco: `__syncthreads()`

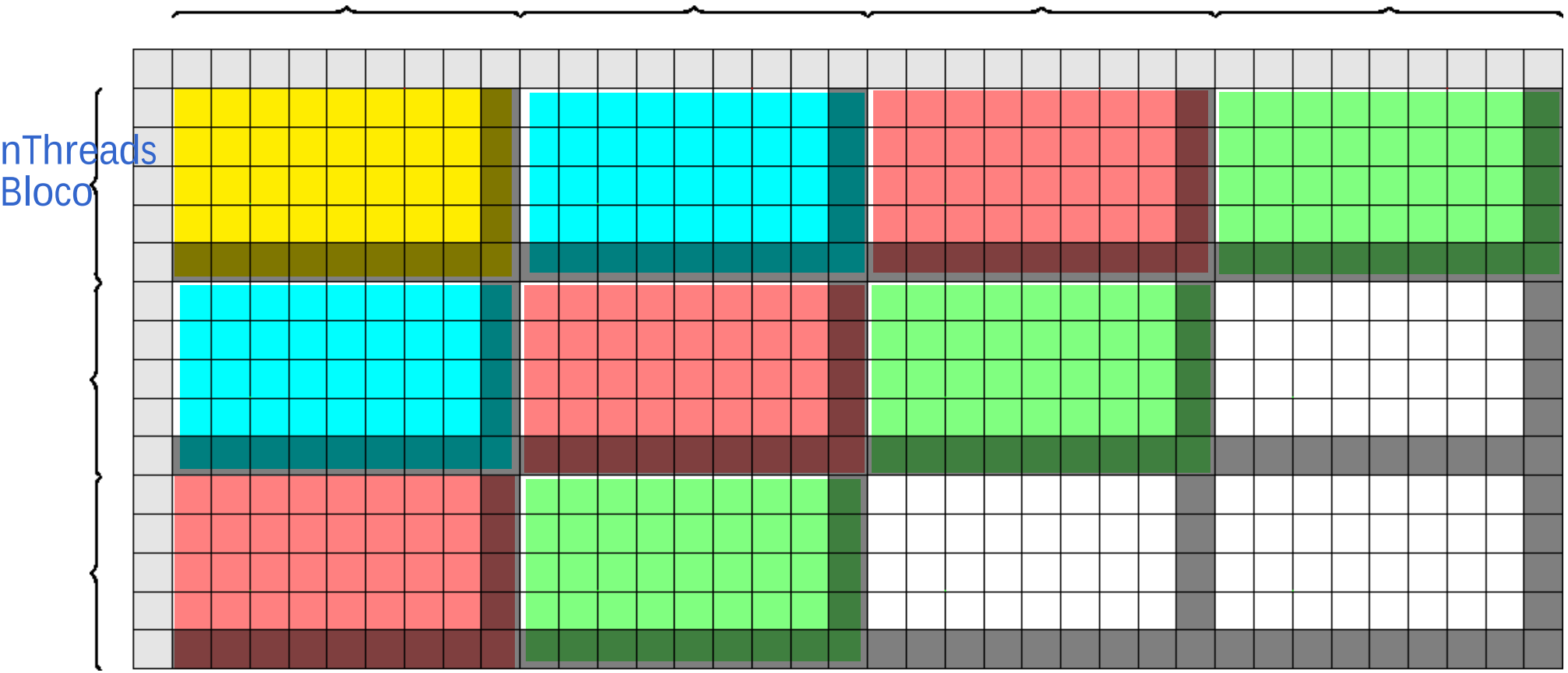


Sugestões para o Desenvolvimento do Programa Paralelo (2)

- **Desenvolver solução completa**: após solução inicial estar correta
  - Supor que: $n \leq m$
  - Permitir sequências maiores
  - Generalizar programa CUDA para ter vários blocos de threads
  - Sincronização entre blocos: realizada encerrando execução do kernel
  - Ter várias invocações do kernel
  - Usar apenas memória global da GPU
  - **Transferências entre host e GPU**: Igual solução inicial
    - Entre uma invocação do kernel e outra invocação:
      - Não é nexessária nenhuma transferência
  - **Manipulação de matriz**: Igual solução inicial

Ideia da Solução CUDA Completa

- Supor que: $n \leq m$
- Vários blocos de threads
- Sincronização entre blocos: encerrando execução do kernel
- Várias invocações do kernel



Programa a ser Desenvolvido

- **Programa deve ser em C ou C++** (em CUDA)
- **Programa sequencial fornecido**:
  - Cálculo de D com percurso por linhas
  - Cálculo de D com percurso por anti-diagonais
- **Desenvolver programa paralelo, usando CUDA**:
  - Explorar paralelismo no cálculo da matriz D
- **Interface de execução do programa**:
  - Por linha de comando com argumento:

    ```
    dist_seq entrada.txt
    dist_par entrada.txt
    ```

- **Submissão**: um único arquivo .zip com programas fonte paralelo
  - **Programa deve ter no cabeçalho**:
    - Nome dos alunos do grupo (máximo 2 alunos)

Entradas e Saídas do Programa

- **Entrada**: Em um único arquivo texto
  - $n \quad m$
  - Sequência S
  - Sequência R
- **Saída**: Na tela
  - Distância de edição
  - Tempo de execução (em ms)
- **Arquivos de entrada fornecidos**:
  - Entradas 1 a 4: Muito pequenas $\Rightarrow$ Apenas para teste e depuração
  - Entrada 3: Duas sequências iguais

Arquivo	n	m	Distância
entrada1.txt	4	6	3
entrada2.txt	20	30	14
entrada3.txt	32	32	0
entrada4.txt	250	500	280
entrada5.txt	500	1000	578
entrada6.txt	512	1024	597
entrada7.txt	8000	10000	4953
entrada8.txt	16384	16384	8468
entrada9.txt	30000	35000	17313