

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:**
 - Sequência s , $n = |s|$
 - Sequência r , $m = |r|$
 - Obs.:
 - Supor que: $n \leq m$
 - Elementos da sequência indexados a partir de 1
 - **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

Cálculo da Distância de Edição entre 2 Sequências

- Exemplo:

- Entradas:

$$s = \begin{array}{|c|c|c|c|} \hline A & C & T & G \\ \hline \end{array} \quad n = 4$$

1 2 3 4

$$r = \begin{array}{|c|c|c|c|c|c|} \hline A & G & C & A & G & T \\ \hline \end{array} \quad m = 6$$

1 2 3 4 5 6

- 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 $s_4 = G$

A		C	T	G	
A	G	C	A	G	T

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 :**

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

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

		r						
		0	1	2	...	j	...	m
s	0	0	1	2	...	j	...	m
	1	1						
	2	2						
	\vdots	\vdots						
	i	i						
	\vdots	\vdots						
	n	n						

- **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 $1 \leq j \leq m$

$$d[i, j] = \min \begin{cases} d[i, j-1] + 1 & \text{(inserção de } r_j \text{ na posição } i+1 \text{ de } s) \\ d[i-1, j] + 1 & \text{(remoção de } s_i) \\ d[i-1, j-1] + t(s_i, r_j) & \text{(substituição ou correspondência)} \end{cases}$$

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

		r						
		0	1	...	$j-1$	j	...	m
s	0	0	1	...	$j-1$	j	...	m
	1	1						
	:	:						
	$i-1$	$i-1$						
	i	i						
	:	:						
	n	n						

Exemplo

- Entradas:

$$s = \begin{array}{|c|c|c|c|} \hline A & C & T & G \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array} \quad n = 4$$

$$r = \begin{array}{|c|c|c|c|c|c|} \hline A & G & C & A & G & T \\ \hline 1 & 2 & 3 & 4 & 5 & 6 \\ \hline \end{array} \quad m = 6$$

- Matriz d :

		r							
			A	G	C	A	G	T	
			0	1	2	3	4	5	6
s	A	0	0	1	2	3	4	5	6
	C	1	1						
	T	2	2						
	G	3	3						
		4	4						

- Saída:

- Distância de edição entre s e $r = d[4, 6] =$

Exemplo (continuação)

- Entradas:

$$s = \begin{array}{|c|c|c|c|} \hline A & C & T & G \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array} \quad n = 4$$

$$r = \begin{array}{|c|c|c|c|c|c|} \hline A & G & C & A & G & T \\ \hline 1 & 2 & 3 & 4 & 5 & 6 \\ \hline \end{array} \quad m = 6$$

- Matriz d :

		r							
			A	G	C	A	G	T	
		0	1	2	3	4	5	6	
s	A	0	0	1	2	3	4	5	6
	C	1	1	0	1	2	3	4	5
	T	2	2	1	1	1	2	3	4
	G	3	3	2	2	2	2	3	3
		4	4	3	2	3	3	2	3

- Saída:

- Distância de edição entre s e $r = d[4, 6] = 3$

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

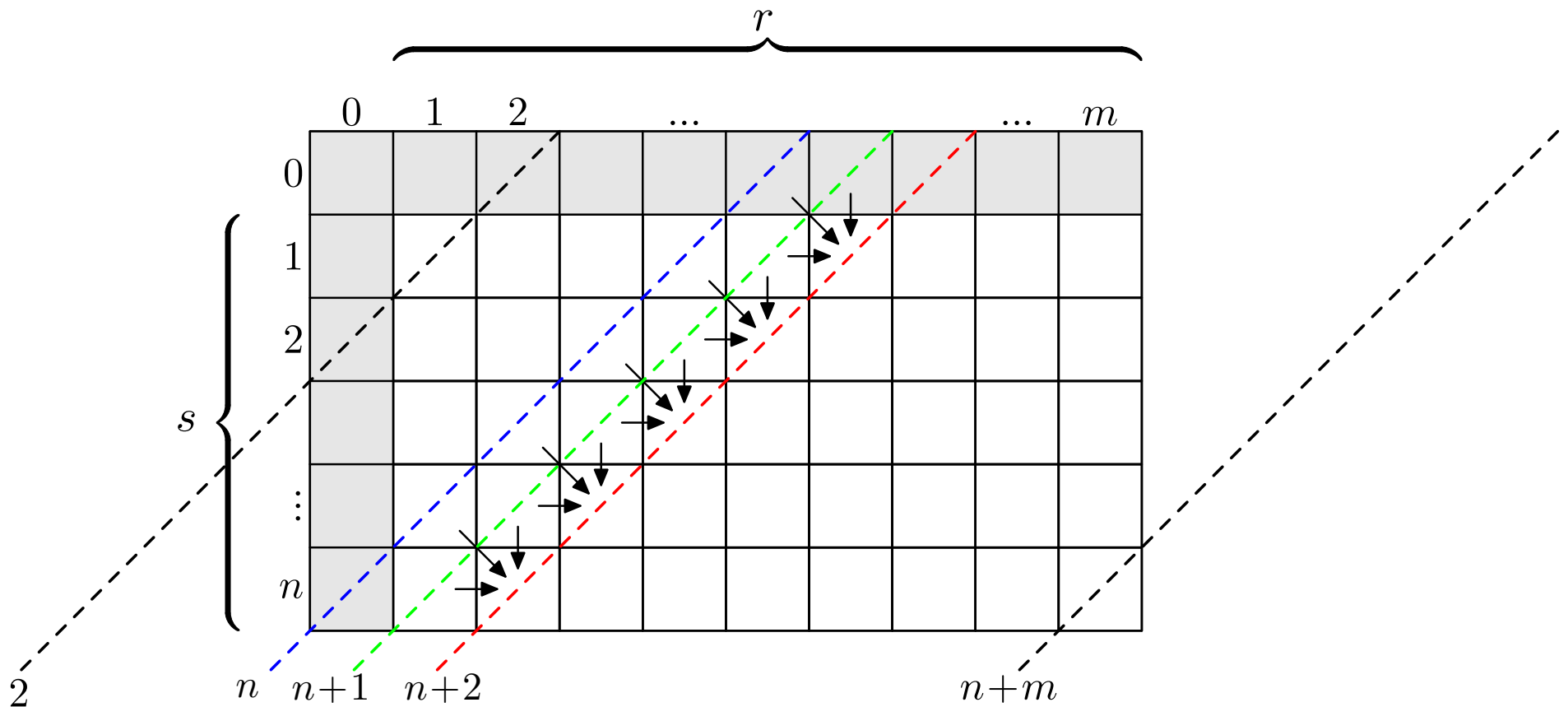
$t := 0$

$d[i, j] := \min(d[i, j - 1] + 1, d[i - 1, j] + 1, d[i - 1, j - 1] + t)$

return $d[n, m]$

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 $n \leq m$

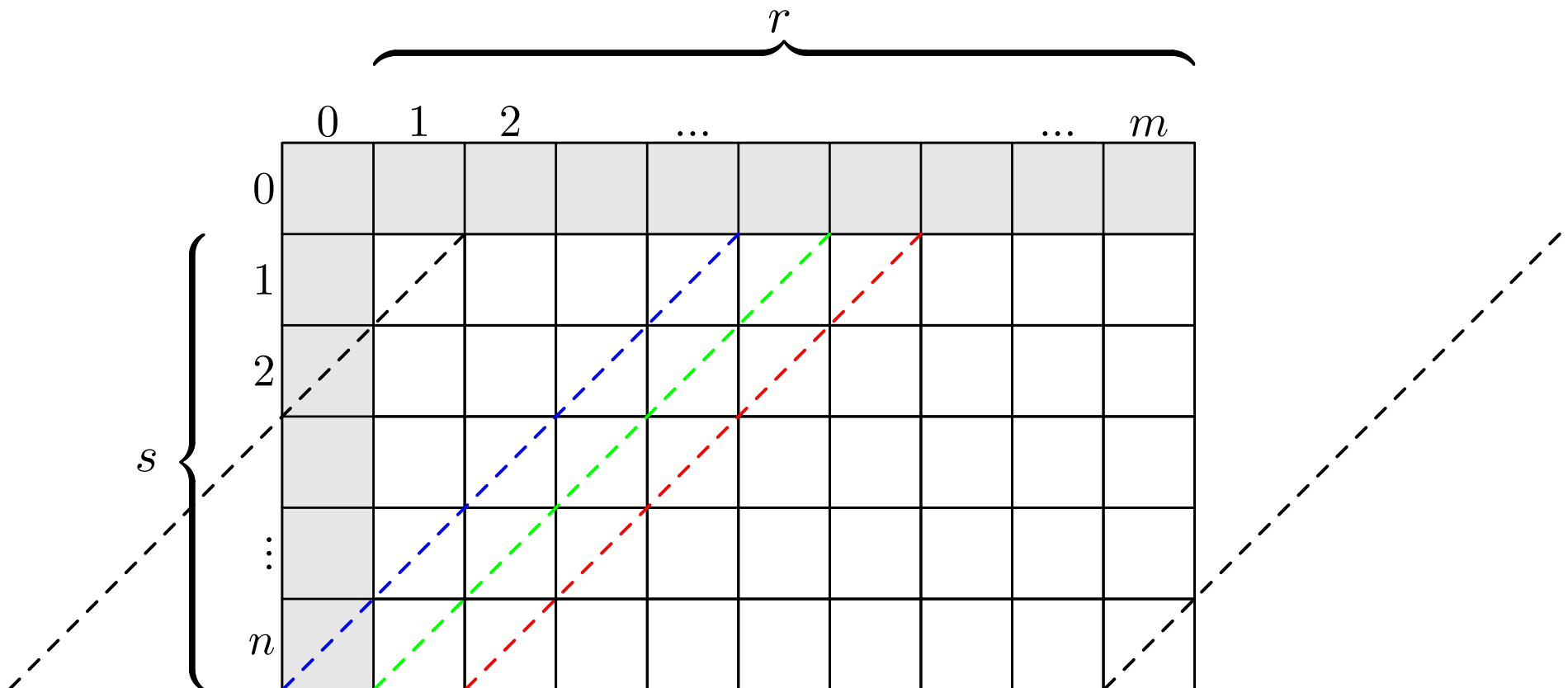
```
nADiag = n + m - 1; // Número de anti-diagonais
tamMaxADiag = n; // Tamanho máximo (número máximo de células) da anti-diagonal
// Para cada anti-diagonal (anti-diagonais numeradas de 2 a nADiag + 1)
for (aD = 2; aD <= nADiag + 1; aD++)
{
    // Para cada célula da anti-diagonal aD
    for (k = 0; 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

- **Inicialmente: entender solução sequencial**
 - Percurso por linhas
 - Percurso por anti-diagonais
- **Desenvolver solução CUDA inicial:** após entender programa sequencial
- **Desenvolver solução CUDA completa:** após solução inicial estar correta
- **Em ambas as soluções:**
 - 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: $d[i][j]$ equivale a $d[i * \text{numColunas} + j]$
 - **Sincronização entre threads do bloco:** `__syncthreads()`

Ideia da Solução CUDA Inicial

- **Supor que:** $n \leq m$ e $n \leq \text{maxThreadsBloco}$
- **Programa CUDA:** com **um único bloco, com n threads**
 - Threads do bloco calculam “em paralelo” células da mesma anti-diagonal
- **Idealmente:** **uma única invocação do kernel**
(**NÃO** ter uma invocação para cada anti-diagonal)
- **Sincronização entre threads do bloco:** entre 2 anti-diagonais sucessivas

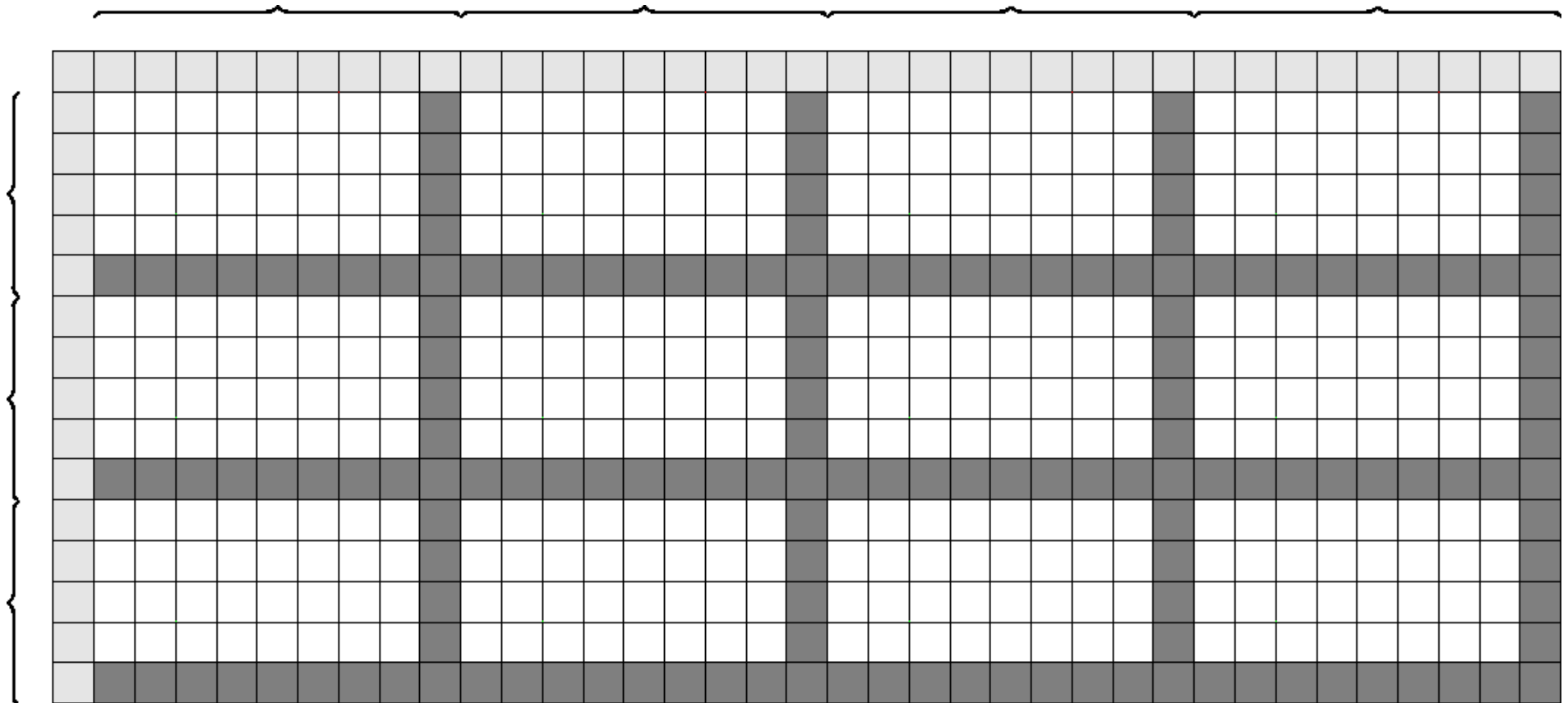


Ideia da Solução CUDA Completa

- **Supor que:** $n \leq m$
- **Permitir sequências maiores:**
 - Eliminar restrição $n \leq \text{maxThreadsBloco}$
- **Generalizar programa CUDA:** **vários blocos de threads**
- **Ter várias invocações do kernel**
- **Idealmente:** **NÃO** ter uma invocação para cada anti-diagonal
- **Sincronização:**
 - **Entre threads do bloco:** `__syncthreads()`
 - **Entre blocos:** encerra execução do kernel e `cudaDeviceSynchronize()`
- **Transferências entre host e GPU:**
 - Igual solução inicial
 - Entre uma invocação do kernel e outra invocação:
 - Não é necessária nenhuma transferência
(matriz d permanece na memória global da GPU)

Ideia da Solução CUDA Completa (continuação)

- **Laço sequencial no host:** sucessivas anti-diagonais de sub-matrizes
- **Invoca kernel c/ vários blocos de threads:** p/ cada antidiagonal de submatrizes
 - **Cada bloco:** calcula uma sub-matriz
 - Threads do bloco calculam “em paralelo” células da mesma anti-diagonal de uma sub-matriz



Trabalho 2

- **Programa sequencial fornecido:**
 - Cálculo de d com percurso por linhas
 - Cálculo de d com percurso por anti-diagonais
- **Grupos:** de 2 alunos
- **Desenvolver programa paralelo:**
 - Programa em CUDA C ou CUDA C++
 - Explorar paralelismo no cálculo da matriz d
 - Programa deve ter nome `dist_par.cu`
- **Comando de compilação usado:**
`nvcc dist_par.cu -o dist_par`
- **Interface de execução do programa:**
 - Por linha de comando com argumento:
`dist_seq entrada.txt`
`dist_par entrada.txt`

Entradas e Saídas do Programa

- **Entrada:** em um único arquivo texto
 - n m
 - Sequência s
 - Sequência r
- **Saída:** na tela
 - Distância de edição
 - Tempo de execução (em ms)
- **NÃO ter outras impressões na tela**
- **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

Submissão do Trabalho

- **Submeter:**
 - Um único arquivo `trab2.zip` com:
 - Arquivos do programa fonte paralelo
 - **Programa deve informar no cabeçalho:**
 - Nome dos alunos do grupo
 - Se o programa implementa:
 - Apenas solução inicial (com grid de um único bloco); ou
 - Solução completa (com grid de vários blocos)
 - Outras informações importantes, se necessário
- **NÃO submeter:**
 - Programa executável, programa sequencial, arquivos de entrada
- **Prazo:**