Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Alinhamento Paralelo de Sequências Biológicas com Poda Agilizada em GPU

Matheus Augusto S. Pinho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientadora
Prof. Dr. Alba Cristina M. A. de Melo

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Alinhamento Paralelo de Sequências Biológicas com Poda Agilizada em GPU

Matheus Augusto S. Pinho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Alba Cristina M. A. de Melo (Orientadora)
CIC/UnB

Prof.a Dr.a Edna Dias Carneiro
Universidade de Brasília

Prof. Dr. Marcelo Grandi Mandelli
Universidade de Brasília

Prof. Dr. João Luiz Azevedo de Carvalho
Coordenador do Curso de Engenharia da Computação

Brasília, 07 de fevereiro de 2023

Dedicatória

Dedico este trabalho primeiramente aos meus pais que, apesar de todas as dificuldades da vida, nunca pouparam esforços para que eu estudasse e chegasse até aqui. Dedico também aos meus irmãos, que são meus melhores amigos e maiores companheiros. Obrigado por andarem de mãos dadas comigo na montanha-russa de emoções dessa trajetória universitária. Nada seria possível sem vocês ao meu lado.

Resumo

A comparação de sequências biológicas é uma das principais tarefas da Bioinformática, existindo uma gama de algoritmos exatos que usam programação dinâmica para este fim. Estes algoritmos apresentam complexidade quadrática no tempo $O(n^2)$ e, dependendo do tamanho das sequências, podem consumir muito tempo de execução. Para amenizar este problema, foram propostas ferramentas paralelas. Este trabalho de graduação propõe e avalia o Módulo de Alinhamento com Poda Agilizada (APA) que, integrado à ferramenta paralela MASA-CUDAlign para uma GPU, modifica a otimização de descarte de blocos (*Block Pruning*), buscando agilizar a poda e aumentar área de descarte da matriz de programação dinâmica. Com esse fim, optou-se por utilizar o alinhador heurístico BLAST para a geração de um escore inicial a ser inserido automaticamente no MASA-CUDAlign, que originalmente iniciava o processamento com escore zero. Para os testes, foram utilizados pares de sequências de DNA de tamanho 3M, 5M, 7M, 10M, 23M e 40M. Os resultados obtidos evidenciam uma melhora no desempenho para comparações de sequências grandes e semelhantes.

Palavras-chave: MASA-CUDAlign, comparação de sequências biológicas, poda, GPU

Abstract

The comparison of biological sequences is one of the main tasks in Bioinformatics. There is a range of exact algorithms that use dynamic programming for this purpose. These algorithms have quadratic complexity in time $O(n^2)$ and can take a lot of execution time depending on the size of the sequences. To alleviate this problem, parallel tools have been proposed. This Undergraduate Project presents the Quick Pruning Alignment Module, which integrated to the one GPU MASA-CUDAlign parallel tool, modifies the Block Pruning optimization strategy, seeking to speed up pruning and increase the the dynamic programming matrix's discart area. For this purpose, the heuristic aligner BLAST is used to generate an initial score to be automatically inserted into MASA-CUDAlign, which originally started computing with a score of 0. For tests, pairs of 3M, 5M, 7M, 10M, 23M and 40M DNA sequences were used. The results show an improvement in performance for comparisons of large and similar sequences.

Keywords: MASA-CUDAlign,biological sequence comparison, Pruning, GPU

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Bioinformática é um campo interdisciplinar que relaciona, como o próprio nome já sugere, o uso de conhecimentos da Informática na busca de soluções para questões da Biologia. Inicialmente, este termo era utilizado em estudos de ecossistemas e ecologia, entretanto, o aumento de pesquisas relacionadas ao DNA [1], RNA [2] e proteínas gerou um enorme volume de dados que precisaram do auxílio da computação para se tornarem informações. Dessa forma, a demanda por ferramentas e métodos computacionais para processamento e análise de dados biológicos aumentou vertiginosamente nos últimos anos [7] .

Uma das tarefas mais corriqueiras da Bioinformática é a comparação de sequências, operação que busca mensurar e analisar a similaridade entre diferentes sequências biológicas. Com este propósito, foram criados algoritmos de alinhamento exatos e heurísticos. Os algoritmos exatos, como Smith Waterman [8], Needleman Wunsch [9], Gotoh [10] e Myers-Miller [2], obtêm o resultado ótimo. Já os heurísticos, como o BLAST [11], garantem apenas um resultado aproximado.

Para a comparação, sequências biológicas (DNA, RNA e proteínas) são traduzidas para caracteres e processadas com o auxílio de uma matriz de programação dinâmica, que é utilizada para calcular a similaridade entre as sequências por meio de um escore baseado em *match* (caracteres coincidentes), *mismatch* (caracteres distintos) e *gap* (espaçamento).

A complexidade quadrática de tempo ($O(n^2)$) é o maior impasse para a utilização dos algoritmos exatos, levando em conta que se as sequências processadas forem muito longas ou o número de sequências for muito grande, o processamento pode demorar bastante tempo. A fim de amenizar este problema, foram desenvolvidas ferramentas de processamento paralelo que, com o auxílio da plataforma CUDA [12] da NVIDIA, utilizam o alto

[1] Abreviação para ácido desoxirribonucleico.
[2] Abreviação para ácido ribonucleico.

poder de computação das GPUs (unidades de processamento gráfico) para o cálculo da comparação de sequências.

Dentre as ferramentas, destaca-se o MASA-CUDAlign [13], que utiliza do algoritmo de Smith Waterma [8] para a obtenção do escore ótimo. Ao longo do tempo, este programa evoluiu e recebeu diferentes otimizações para cada uma de suas versões. No MASA-CUDAlign 2.1 [4], por exemplo, foi implementada a poda (*Block Pruning*), técnica matemática que se baseia no melhor escore obtido até o momento para descartar blocos da matriz que não conduzem ao resultado ótimo. No MASA-CUDAlign, o melhor escore é inicializado com o valor zero e cresce à medida que melhores escores são calculados.

De maneira a agilizar ainda mais a obtenção do escore ótimo, o presente trabalho de graduação propõe, implementa e avalia uma solução para que o melhor escore comece com um valor maior que zero, o que muito provavelmente irá aumentar a área de poda não calculada e, consequentemente, reduzir o tempo de execução. Para tanto, foi extraído de um algoritmo heurístico o escore do melhor resultado e inserido no início da execução do MASA-CUDAlign em uma GPU, tudo isso de forma sequencial. A solução proposta compõe o Módulo de Alinhamento com Poda Agilizada (APA), cujo código está disponível no *github* do autor deste projeto [14], bem como as sequências de usadas nos testes.

O restante deste trabalho de graduação está dividido da seguinte maneira: o Capítulo 2 traz definições e algoritmos utilizados no alinhamento de sequências biológicas. O Capítulo 3 apresenta a evolução da ferramenta CUDAlign. O Capítulo 4 detalha o projeto desenvolvido neste trabalho. O Capítulo 5 expõe o ambiente de testes e analisa os resultados encontrados. O Capitulo 6, por fim, traz a conclusão e os trabalhos futuros sugeridos.

Capítulo 2

Comparação de Sequências Biológicas

No presente capítulo, são apresentados importantes conceitos e algoritmos que abrangem o estudo da comparação de sequências biológicas, uma das mais importantes tarefas da Bioinfomática, que visa encontrar semelhanças entre diferentes sequências. Inicialmente, a Seção 2.1 traz definições para assistir o entendimento acerca das sequências biológicas e seus algoritmos de comparação. Logo em seguida, a Seção 2.2 apresenta alguns dos principais algoritmos exatos de comparação de sequências.

2.1 Definições

Sequência biológica é o termo utilizado para descrever uma série de pequenas moléculas (monômeros) que compõe moléculas biológicas gigantes (polímeros). Exemplos de polímeros são o DNA, ácido nucleico portador do código genético de cada indivíduo, e as proteínas, macromoléculas que participam de todos os processos biológicos dos seres vivos [15].

Computacionalmente, as sequências biológicas são representadas por *strings* com caracteres que correspondem aos seus monômeros. No DNA, os caracteres $\Sigma = \{A, T, C, G\}$ equivalem às bases nitrogenadas presentes em seus nucleotídeos, já nas proteínas, $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ são representações de seus aminoácidos [16].

Mutações são importantes fontes de variabilidade genética e podem dar origem a novas espécies. Na natureza, ao invés de reinventadas, novas sequências biológicas são adaptadas de sequências preexistentes [17], fato que favorece análises computacionais, tendo em vista que é possível encontrar similaridades entre segmentos biológicos novos

e outros já conhecidos e, assim, verificar características evolutivas, comparar genomas e inferir utilidades.

A tarefa mais básica na análise de sequências é questionar se elas são relacionadas [17], para isso, utiliza-se o alinhamento, técnica que busca caracteres individuais ou padrões de caracteres na mesma ordem entre duas ou mais sequências [16]. O alinhamento pode ser classificado em global, semi-global e local. No global, as sequências são inteiramente alinhadas a fim de encontrar o pareamento com a maior quantidade de caracteres coincidentes. No semi-global, a comparação acontece excluindo a região inicial ou final de uma ou ambas as sequências. Já no alinhamento local, são emparelhados somente segmentos com maior similaridade.

Com o propósito de mensurar a qualidade dos alinhamentos, um escore é adotado atribuindo pontuações para os *matches* (caracteres coincidentes) e penalidades para os *mismatches* (caracteres diferentes) e *gaps* (espaços vazios na comparação). A depender do algoritmo de alinhamento empregado, pode-se utilizar o *linear gap*, em que a penalidade de *gap* é a mesma para todas as aparições (Equação 2.1), ou o *affine gap*, que considera uma penalidade de abertura G_{open} e outras de extensão G_{ext} de acordo com o tamanho da sequência de *gaps* (Equações 2.2 2.3). Nestas equações, K é o número de *gaps* consecutivos e G é a penalidade linear de *gap*.

$$\gamma(K) = -G \cdot K \tag{2.1}$$

$$\gamma(K) = -G_{open} - (K-1) \cdot G_{ext} \tag{2.2}$$

$$G_{open} = G_{first} + G_{ext} \tag{2.3}$$

A Figura 2.1 exemplifica os alinhamentos global, local e semiglobal entre as sequências $S_0 = \{TACAATGCT\}$ e $S_1 = \{TGTCAATTAT\}$. Visando calcular os escores dos três tipos de alinhamentos, foram atribuídas as pontuações +1 para *match*, -1 para *mismatch* e a penalidade 2 para os *linear gap*.

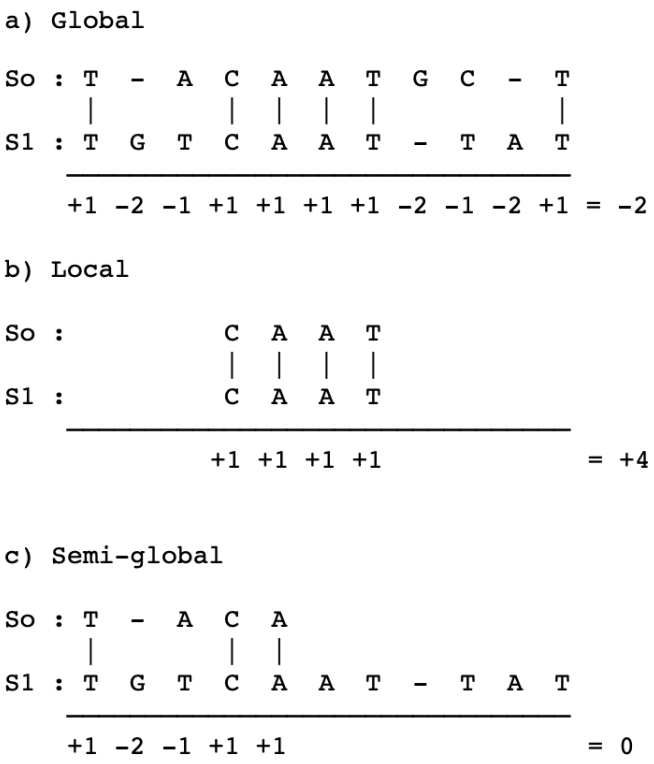


Figura 2.1: Alinhamentos global, local e semi-global.

2.2 Algoritmos de Alinhamento

Considerando sequências de mesmo tamanho n, a ocorrência de *gaps* viabiliza a existência de um número muito grande de possíveis alinhamentos [17], conforme explicitado na Equação 2.4. A fim de solucionar este problema e encontrar o melhor escore de um par de sequências em tempo hábil, são utilizados os algoritmos de alinhamento.

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \simeq \frac{2^{2n}}{\sqrt{\pi n}} \tag{2.4}$$

A seguir, nas Seções 2.2.1, 2.2.2, 2.2.3 e 2.2.4 são apresentados os principais algoritmos exatos de alinhamento. Na Seção 2.2.5, evidencia-se a ferramenta de alinhamento heurístico BLAST.

2.2.1 Algoritmo Needleman-Wunsch (NW)

O algoritmo de Needleman-Wunsch (NW) [9] visa obter o alinhamento global ótimo entre duas sequências S_0 e S_1 utilizando *gaps* com penalidade linear. A execução do algoritmo pode ser dividida em duas partes:

- **Cálculo da matriz de programação dinâmica**: Esta etapa tem como objetivo o cálculo de uma matriz H, em que cada um de seus elementos $H_{i,j}$ corresponde ao escore do melhor alinhamento entre os prefixos dos segmentos até ele. Para elaborar a matriz de programação dinâmica, os seguintes passos são aplicados:

  1. Inicializar o elemento $H_{0,0} = 0$, a primeira linha da matriz com $H_{0,j} = -j \cdot G$ e a primeira coluna com $H_{i,0} = -i \cdot G$, em que G é a penalidade linear de *gap*.
  2. Preencher a matriz do canto superior esquerdo ao inferior direito. O valor de cada célula $H_{i,j}$ é definido a partir da análise do elemento da diagonal imediatamente superior $H_{i-1,j-1}$, da linha imediatamente superior $H_{i-1,j}$ e da coluna imediatamente à esquerda $H_{i,j-1}$ usando a equação de recorrência 2.5. A Figura 2.2 ilustra as relações de um elemento com seus vizinhos para a definição do escore.

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + sbt(S_0[i], S_1[j])) \\ H_{i-1,j} - G \\ H_{i,j-1} - G \end{cases} \tag{2.5}$$

  A Equação 2.5 é repetidamente aplicada até que a matriz de programação dinâmica seja completamente calculada. É atribuído a cada elemento o maior valor encontrado entre os três casos apresentados. O termo *sbt* na equação representa uma matriz de substituição, matriz de duas dimensões que atribui uma pontuação numérica (*match* ou *mismatch*) para o par de elementos $S_0[i]$,$S_0[j]$.

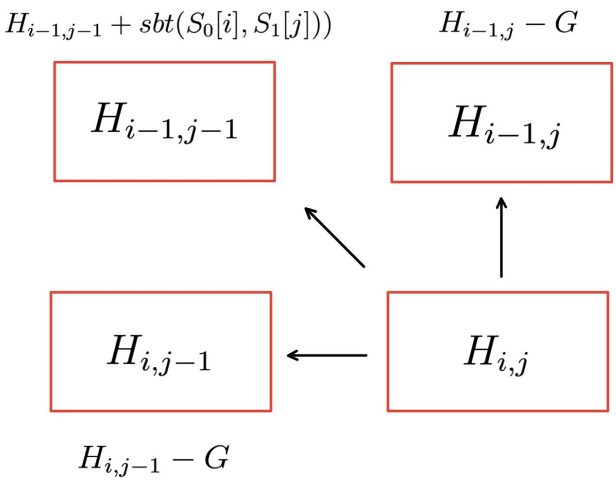


Figura 2.2: Representação das células vizinhas na matriz.

- ***Traceback:*** Ao ser calculada, cada célula da matriz de programação dinâmica possui também um ponteiro que indica de qual das suas vizinhas $H_{i-1,j-1}$, $H_{i-1,j}$, $H_{i,j-1}$ seu resultado se originou. Dessa forma, para encontrar o alinhamento global ótimo, basta percorrer a matriz do canto inferior direito ao superior esquerdo acompanhando os ponteiros definidos. É importante ressaltar que neste processo pode haver mais de um caminho possível, ou seja, mais de um alinhamento global ótimo. Na computação do melhor caminho dentre os gerados, prioriza-se o ponteiro diagonal, o superior e o esquerdo, respectivamente.

	*	A	T	G	A	A	T	C	C	A	T
*	**0**	−2	−4	−6	−8	−10	−12	−14	−16	−18	−20
T	**-2**	−1	−1	−3	−5	−7	−9	−11	−13	−15	−17
G	−4	**-3**	−2	0	−2	−4	−6	−8	−10	−12	−14
T	−6	−5	**-2**	−2	−1	−3	−3	−5	−7	−9	−11
C	−8	−7	−4	**-3**	−3	−2	−4	−2	−4	−6	−8
C	−10	−9	−6	−5	**-4**	−4	−3	−3	−1	−3	−5
A	−12	−9	−8	−7	−4	**-3**	−5	−4	−3	0	−2
T	−14	−11	−8	−9	−6	−5	**-2**	**-4**	−5	−2	1
T	−16	−13	−10	−9	−8	−7	−4	−3	**-5**	−4	−1
A	−18	−15	−12	−11	−8	−7	−6	−5	−4	**-4**	−3
T	−20	−17	−14	−13	−10	−9	−6	−7	−6	−5	**-3**

Figura 2.3: Matriz de programação dinâmica NW com ponteiros de *traceback*

Na Figura 2.3 é feito o alinhamento global entre as sequências $S_0 = \{A, T, G, A, A, T, C, C, A, T\}$ e $S_1 = \{T, G, T, C, C, A, T, T, A, T\}$ e encontrado o melhor caminho saindo do elemento $H_{10,10} = \{-3\}$ e chegando no $H_{0,0} = \{0\}$. Para a construção da matriz de programação dinâmica de NW, foram consideradas as pontuações +1 para *match*, -1 para *mismatch* e a penalidade 2 para *gap*. O alinhamento gerou um escore -3 conforme exposto na Figura 2.4.

O algoritmo de Needleman-Wunsch tem a complexidade *O(mn)* tanto no tempo quanto no espaço, considerando m e n o tamanho das sequências comparadas [9].

2.2.2 Algoritmo Smith-Waterman (SW)

Mais usual que o alinhamento global ótimo é o alinhamento local ótimo, que pode ser obtido usando o algoritmo de Smith-Waterman (SW) [8]. Proposto em 1981, este

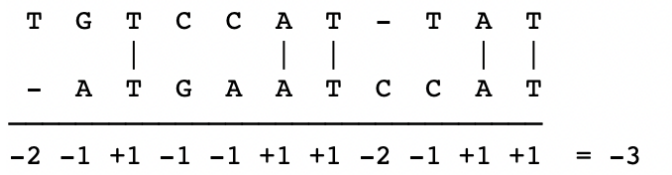


Figura 2.4: Alinhamento global ótimo (NW).

algoritmo se relaciona intimamente com o apresentado na Seção 2.2.1, trazendo algumas modificações.

Na matriz de programação dinâmica de SW, não há número negativo. Para tanto, todos os valores menores que zero são substituídos pelo zero, como evidencia a equação de recorrência 2.6. Ao inicializarmos a matriz, tanto os elementos da primeira linha $H_{0,j}$ quanto da primeira coluna $H_{i,0}$ recebem o zero.

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + sbt(S_0[i], S_1[j])) \\ H_{i-1,j} - G \\ H_{i,j-1} - G \\ 0 \end{cases} \tag{2.6}$$

Outra discrepância entre Smith-Waterman e Needleman-Wunsch diz respeito ao *traceback*. Ao invés do rastreamento começar no último elemento à esquerda da matriz $H_{m,n}$, ele se inicia no local com o maior valor e percorre o caminho até encontrar um elemento com o valor zero.

A Figura 2.5 traz um exemplo de alinhamento local ótimo entre as sequências $S_0 = \{A, T, G, A, A, T, C, C, A, T\}$ e $S_1 = \{T, G, T, C, C, A, T, T, A, T\}$. Findado o preenchimento da matriz de programação dinâmica, foi selecionada a célula $H_{7,10} = 5$ que possui o maior valor da matriz (escore ótimo) e a partir dela realizado o *traceback* até encontrar o zero.

	*	A	T	G	A	A	T	C	C	A	T
*	0	0	0	0	0	0	0	0	0	0	0
T	0	0	1	0	0	0	1	0	0	0	1
G	0	0	0	2	0	0	0	0	0	0	0
T	0	0	1	0	1	0	**1**	0	0	0	1
C	0	0	0	0	0	0	0	**2**	1	0	0
C	0	0	0	0	0	0	0	1	**3**	1	0
A	0	0	0	0	1	1	0	0	1	**4**	0
T	0	0	1	0	0	0	2	0	0	2	**5**
T	0	0	1	0	0	0	1	0	0	0	3
A	0	1	0	0	1	1	0	0	0	1	1
T	0	0	2	0	0	0	2	0	0	0	2

Figura 2.5: Matriz de programação dinâmica SW com *traceback*

```
T  G  T  C  T
|  |  |  |  |
T  C  C  A  T
-------------
+1 +1 +1 +1 +1  = +5
```

Figura 2.6: Alinhamento local ótimo (SW).

A Figura 2.6 ilustra o alinhamento local ótimo obtido pelo algoritmo SW que considera, assim como em NW, $match = +1$, $mismatch =$ -1 e *gap* com penalidade 2.

2.2.3 Algoritmo Gotoh

Os algoritmos de Needleman-Wunsch (Seção 2.2.1) e Smith-Waterman (Seção 2.2.2) apresentados anteriormente utilizam o *linear gap*, modelo em que cada *gap* gera a mesma penalidade para o escore. Diferentemente de seus antecessores, o algoritmo de Gotoh [10] computa o alinhamento global ou local ótimo com a penalidade de *gap* no modelo *affine gap* (Equação 2.2), em que é considerada uma penalidade de abertura G_{open} para o primeiro *gap* e outra de extensão G_{ext} para os subsequentes. Este algoritmo incorpora a suposição de que um único grande evento de inserção ou deleção em uma sequência biológica é mais provável de acontecer do que pequenas modificações [10].

Para a execução do algoritmo, são criadas três diferentes variáveis para cada posição (i,j), dando origem à matrizes H, E e F, conforme as equações de recorrência 2.7, 2.8 e 2.9. A matriz H é usada para calcular os *matches* e *mismatches* entre as sequências S_0 e S_1, a E, uma sequência de *gaps* em S_0 e a F, os *gaps* em S_1.

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + sbt(S_0[i], S_1[j])) \\ E_{i,j} \\ F_{i,j} \end{cases} \tag{2.7}$$

$$E_{i,j} = max \begin{cases} E_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{open} \end{cases} \tag{2.8}$$

$$F_{i,j} = max \begin{cases} F_{i-1,j} - G_{ext} \\ H_{i-1,j} - G_{open} \end{cases} \tag{2.9}$$

É importante salientar que para a obtenção do alinhamento local ótimo por meio do algoritmo de Gotoh, é preciso restringir o valor mínimo da Equação de recorrência 2.7 em 0.

	*	C	C	C	T	T	T	T	G
*	**0**	0	0	0	0	0	0	0	0
C	0	**2**	2	2	0	0	0	0	0
C	0	2	**4**	4	1	0	0	0	0
C	0	2	4	**6**	3	1	0	0	0
A	0	0	1	2	5	2	0	0	0
A	0	0	0	**1**	2	4	1	0	0
T	0	0	0	0	**3**	4	6	3	1
T	0	0	0	0	2	**5**	6	8	4
T	0	0	0	0	2	4	**7**	8	7
T	0	0	0	0	2	4	6	**9**	7
A	0	0	0	0	0	1	3	5	8

(a) Matriz H.

	*	C	C	C	T	T	T	T	G
*	–	–	–	–	–	–	–	–	–
C	$-\infty$	−4	−2	−2	−2	−3	−4	−4	−4
C	$-\infty$	−4	−2	0	0	−1	−2	−3	−4
C	$-\infty$	−4	−2	0	2	1	0	−1	−2
A	$-\infty$	−4	−4	−3	−1	1	0	−1	−2
A	$-\infty$	−4	−4	−4	−3	−2	0	−1	−2
T	$-\infty$	−4	−4	−4	−4	−1	0	2	1
T	$-\infty$	−4	−4	−4	−4	−2	1	2	4
T	$-\infty$	−4	−4	−4	−4	−2	0	3	4
T	$-\infty$	−4	−4	−4	−4	−2	0	2	5
A	$-\infty$	−4	−4	−4	−4	−4	−3	−1	1

(b) Matriz E.

	*	C	C	C	T	T	T	T	G
*	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
C	–	−4	−4	−4	−4	−4	−4	−4	−4
C	–	−2	−2	−2	−4	−4	−4	−4	−4
C	–	−2	0	0	−3	−4	−4	−4	−4
A	–	−2	0	**2**	−1	−3	−4	−4	−4
A	–	−3	−1	**1**	1	−2	−4	−4	−4
T	–	−4	−2	0	0	0	−3	−4	−4
T	–	−4	−3	−1	−1	0	2	−1	−3
T	–	−4	−4	−2	−2	1	2	4	0
T	–	−4	−4	−3	−2	0	3	4	3
A	–	−4	−4	−4	−2	0	2	5	3

(c) Matriz F.

C	*C*	*C*	*A*	*A*	*T*	*T*	*T*	*T*
C	*C*	*C*	_	_	*T*	*T*	*T*	*T*
2	2	2	−4	−1	2	2	2	2

(d) Alinhamento Gotoh (escore=9).

Figura 2.7: Matrizes Gotoh H, F e E para sequências $S_0 = CCCAATTTTA$ e $S_1 = CCCTTTTG$. Fonte: [1] .

A Figura 2.7 apresenta as matrizes H, E e F de Gotoh para o alinhamento local ótimo com *affine gap* entre as sequências $S_0 = CCCAATTTTA$ e $S_1 = CCCTTTTG$. Nela são utilizados os valores +2 para *match*, -1 para *mismatch* e penalidades de 4 para abertura

de *gap* e 1 para extensão. É possível notar que com auxílio da Matriz F, dois *gaps* na sequência S_0 são penalizados com valores diferentes.

2.2.4 Algoritmo Myers-Miller (MM)

Com o objetivo de resolver o problema do alinhamento local ou global ótimo entre sequências longas, foi proposto o algoritmo de Myers-Miller [2], que concilia a solução de Gotoh (Seção 2.2.3) com o algoritmo em espaço linear para a obtenção da maior subsequência em comum (LCS – *Longest Common Subsequence*) proposto por Hischberg [18]. Dessa forma, é possível obter o alinhamento ótimo em espaço linear $O(m+n)$.

A ideia principal do MM é encontrar o ponto médio (*crosspoint*) pelo qual o alinhamento ótimo passa. Para isso, o algoritmo calcula inicialmente a matriz entre as sequências S_0 e S_1 até a linha $\frac{m}{2}$, armazenando os *scores* terminados em *match* e *mismatch* no vetor CC e os terminados em gap no DD. Posteriormente, a mesma coisa é feita com as sequências invertidas S'_0 e S'_1, armazenando *scores* em CC' e DD'.

Após todas estas etapas, calcula-se os *scores* dos elementos da linha $\frac{m}{2}$ e coluna j, conforme a Equação 2.10 .

$$C_j = max \begin{cases} CC_j + CC'_j \\ DD_j + DD'_j - G_{open} \end{cases} \tag{2.10}$$

Assim, o maior valor entre os C_j será o ponto médio por onde o alinhamento ótimo passa. Este procedimento de divisão da matriz é executado sucessivamente para as submatrizes resultantes até que o problema se torne trivial, possibilitando que o alinhamento global ótimo seja encontrado, conforme a Figura 2.8.

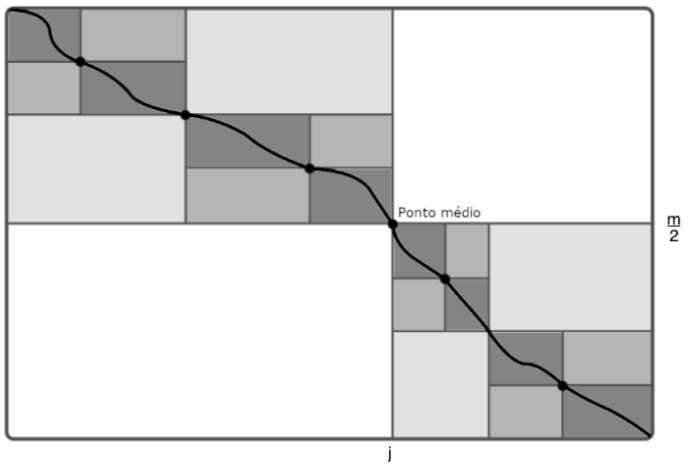


Figura 2.8: Representação do algoritmo de Myers-Miller (Adaptado de [2]) .

2.2.5 BLAST

Diferentemente dos algoritmos apresentados anteriormente, o BLAST [11] (*Basic Local Alignment Search Tool*) é uma ferramenta heurística, ou seja, não garante a obtenção da melhor solução. Ele utiliza uma versão simplificada do algoritmo Smith-Waterman (Seção 2.2.2) para comparar sequências biológicas, identificar regiões semelhantes e gerar o alinhamento local ótimo. A ferramenta é amplamente utilizada em pesquisas em biologia molecular, genômica e outras áreas relacionadas.

O funcionamento do BLAST realiza buscas heurísticas comparando regiões similares entre as sequências contra um grande banco de dados. Sua execução se baseia em 3 etapas:

1. **Busca por palavras-chave:** nesta etapa, a sequência de consulta é dividida em palavras-chave de um determinado tamanho K, que são usadas para pesquisa no banco de dados de sequências.
2. **Extensão de palavra-chave:** as palavras-chave que encontram correspondências significativas no banco de dados são estendidas em ambas as direções.
3. **Avaliação da similaridade:** os segmentos estendidos são avaliados e pontuados.

O BLAST possui inúmeras variações, dentre elas destaca-se o BLASTn, versão utilizada para comparação de sequências de DNA.

Capítulo 3

A Evolução do CUDAlign

O alinhamento em tempo razoável de sequências com tamanho na ordem de milhões de bases nitrogenadas se tornou um grande desafio para a bioinformática, visto que demanda um alto poder de processamento e muita memória. Nesta circunstância, foi desenvolvido o CUDAlign, ferramenta que alinha grandes sequências em GPU (placas gráficas) com escore ótimo e tempo satisfatório.

A elaboração do CUDAlign se baseou na arquitetura CUDA (*Compute Unified Device Architecture*) da NVIDIA [12]. Esta API é desenhada para que desenvolvedores, através de linguagens de alto nível como C, C++ e Fortran, utilizem do alto potencial de processamento de suas GPUs para produzir projetos de GPGPU [1], computação paralela e computação heterogênea.

Este capítulo apresenta a evolução do CUDAlign e as mudanças implementadas em cada uma de suas versões.

3.1 CUDAlign 1.0

O CUDAlign 1.0 [19] foi elaborado com o objetivo de obter o escore local ótimo do alinhamento de longas sequências de DNA utilizando o algoritmo Gotoh (*affine gap*) (Seção 2.2.3). Como só é processada a fase de cálculo da matriz, o algoritmo se executa em memória linear. Para tanto, o CUDAlign 1.0 implementa o *wavefront*, técnica que possibilita a paralelização do cálculo das equações de recorrência do algoritmo citado.

Na computação da matriz de programação dinâmica, um elemento $H_{i,j}$ precisa acessar apenas $H_{i-1,j-1}$, $H_{i-1,j}$ e $H_{i,j-1}$ (Figura 2.2) para ser calculado, assim é possível notar que os elementos das anti-diagonais independem entre si. A técnica $wavefront$ empregada no CUDAlign usa deste artifício para processar paralelamente cada diagonal da matriz.

[1] Unidade de Processamento Gráfico de Propósito Geral

O *wavefront* foi utilizado em dois níveis: paralelismo externo (Seção 3.1.1) e paralelismo interno (Seção 3.1.2).

3.1.1 Paralelismo Externo

No paralelismo externo, as células da matriz de programação dinâmica são agrupadas em blocos para serem processadas. Cada um dos blocos tem R linhas e C colunas ($R \times C$ elementos), resultando em uma matriz segmentada por $\frac{m}{R} \times \frac{n}{C}$ blocos que formam um *grid* G, tal que m e n são o tamanho das sequências S_0 e S_1. A Figura 3.1 ilustra a divisão em blocos.

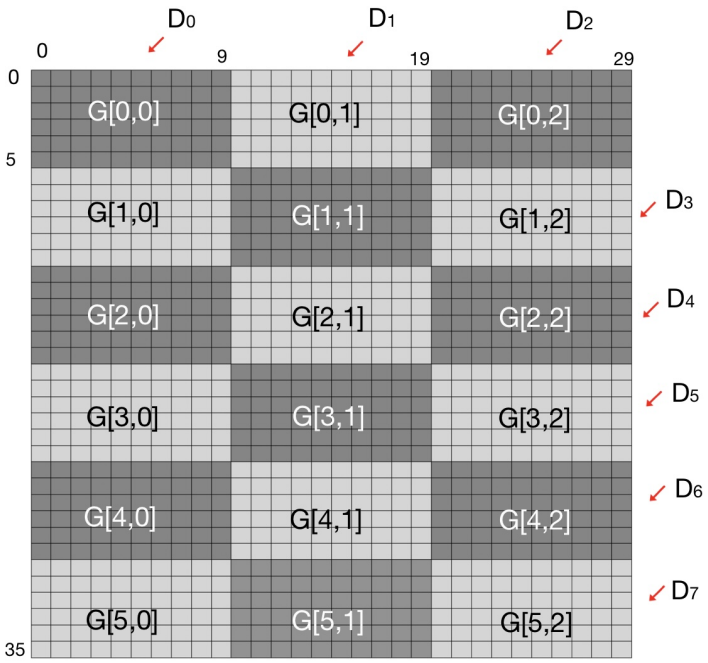


Figura 3.1: Matriz segmentada em blocos dando origem a um *Grid*. As setas vermelhas indicam cada uma das diagonais externas.

No agrupamento dos blocos, a escolha dos valores de R e C acontece mediante o número de blocos concorrentes B, número de *threads* por bloco T e o número de linhas que cada *thread* é responsável por processar α, valores que por sua vez são definidos de acordo com as especificações das GPU. Assim, são executadas as Equações 3.2 e 3.1.

$$R = \alpha \cdot T \tag{3.1}$$

$$C = \frac{n}{B} \tag{3.2}$$

$$D_k = \{G_{i,j} | i + j = k\} \tag{3.3}$$

Cada bloco do *grid* é agrupado em uma anti-diagonal, denominada diagonal externa, $D_k = \{1, 2, 3, 4...\}$ seguindo a Equação 3.3. Estas anti-diagonais são processadas com a técnica de *wavefront* seguindo os seguintes passos [19]:

1. A CPU faz a chamada de execução para o *kernel* processar cada um dos blocos de uma diagonal externa com B blocos e T *threads* por bloco.
2. A GPU finaliza a execução de cada um dos blocos de uma diagonal D_k.
3. A CPU faz a sincronização global e invoca novamente o *kernel* para o processamento de uma nova diagonal externa.

Este processo é repetido até que as diagonais externas sejam todas processadas.

3.1.2 Paralelismo Interno

O paralelismo interno baseia-se na mesma ideia apresentada na Seção 3.1.1, levando em conta que agora são calculadas as diagonais internas a cada um dos blocos G_k. Cada diagonal interna $d_k = \{1, 2, 3, 4, 5, 6...\}$ é composta pelos elementos $d_k = \{(i, j) | \frac{i}{\alpha} + j = k\}$ dado que os valores de i e j têm como referência o elemento superior esquerdo do bloco.

Há em cada bloco uma quantidade *threads* T_k processando da linha $\alpha \cdot k$ até a $\alpha \cdot k + \alpha \cdot k - 1$ paralelamente como dita a estratégia *wavefront*. Dessa forma, ao mesmo tempo que uma thread T_k processa em j, uma *thread* T_{k+1} processará em $j - 1$.

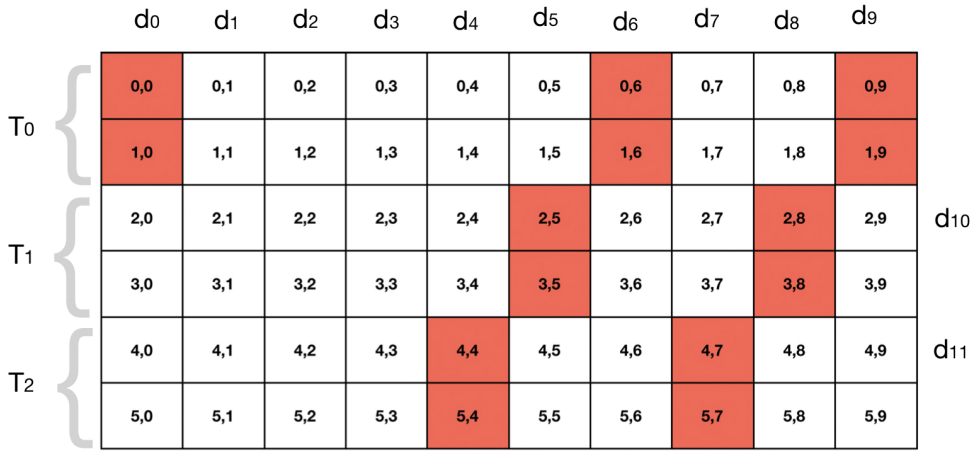


Figura 3.2: Bloco retangular com 60 elementos. As diagonais internas d_0, d_6 e d_9 estão destacadas.

A Figura 3.2 ilustra a execução com 3 *threads* de um dos blocos 6×10 da matriz apresentada na Figura 3.1, com cada *thread* calculando duas linhas. Nela estão destacadas as diagonais d_0 (apenas a *thread* T_0 executa), d_6 e d_9 (ambas processadas paralelamente por T_0, T_1 e T_2).

3.1.3 Estrutura de Memória

Para o melhor desempenho da comparação de grandes sequências de DNA, é necessário que as estruturas de dados em memória sejam bem projetadas de acordo com o disponível na arquitetura CUDA. Esta Seção apresenta a forma com que os dados do CUDAlign 1.0 foram organizados.

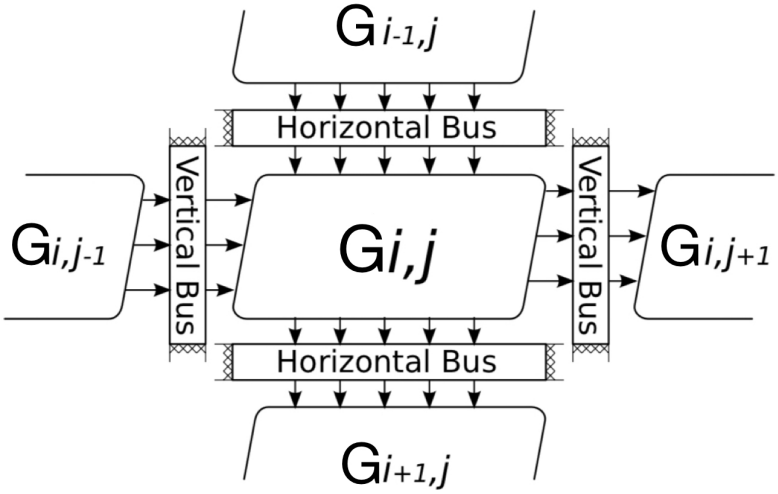


Figura 3.3: Representação dos barramentos verticais e horizontais. Adaptado de [3] .

- **Sequências**: As sequências S_0 e S_1 são armazenadas em uma memória somente de leitura denominada textura.
- **Memória Compartilhada**: A memória compartilhada é usada para a transferência dos valores H e F da última linha de cada *thread* para as posteriores, com exceção da última *thread* que utiliza a memória global e encaminha seus valores à primeira *thread* do próximo bloco .
- **Barramento Horizontal**: O barramento horizontal é utilizado para a transferência de dados da última linha de um bloco ao bloco imediatamente inferior. Neste processo, a escrita é feita utilizando a memória global enquanto a leitura usa a textura. A Figura 3.3 ilustra a tranferência de dados através do barramento horizontal. Nela o bloco $G_{i-1,j}$ tranfere dados para o $G_{i,j}$, que tranfere para o bloco imediatamento inferior a ele, o $G_{i+1,j}$.
- **Barramento Vertical**: O barramento vertical é aplicado para a transferência de valores da última coluna de cada bloco ao bloco imediatamente à direita usando a memória global. A Figura 3.3 ilustra também a tranferência de dados através do barramento vertical. Nela o bloco $G_{i,j-1}$ tranfere dados para o $G_{i,j}$, que tranfere para o bloco imediatamento à direita, o $G_{i,j+1}$.

3.1.4 Otimizações

Visando alcançar melhores níveis de paralelismo e possibilitar um melhor desempenho na arquitetura CUDA, foram propostas no CUDAlign 1.0 algumas estratégias de otimização. A primeira delas foi a delegação de células, técnica que propõe uma segmentação em paralelogramo para a execução das diagonais internas ao bloco, atingindo paralelismo máximo na maior parte do cálculo da matriz. Nela, um bloco R×C será executado até a diagonal C, fazendo com que suas diagonais subsequentes sejam executados pelo bloco à direita. A Figura 3.4 ilustra a forma com que a divisão dos blocos acontece no modelo de delegação de células.

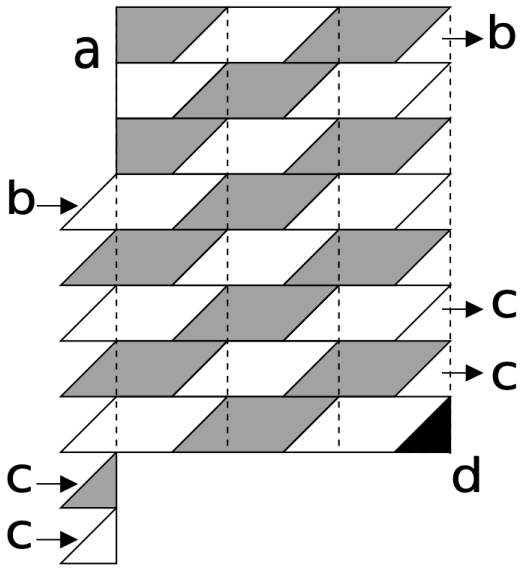


Figura 3.4: Delegação de células do CUDAlign 1.0 [3] .

Apesar de muito eficiente, o método de delegação de células cria um problema: como há áreas de um bloco delegadas para a diagonal externa seguinte, os blocos de uma mesma diagonal externa podem agora possuir dependência entre si.

A fim de eliminar este problema, é feito o cálculo de todas as células pendentes antes que precisem ser lidas e, ao mesmo tempo, a sincronização de execução dos blocos [3]. Para isso, divide-se a execução em duas fases, a fase curta e a fase longa.

- **Fase Curta**: Nesta fase, são processadas todas as células pendentes. São calculados os elementos das $T - 1$ primeiras diagonais internas dos blocos e, em seguida, o controle é reestabelecido à CPU para a sincronização.
- **Fase Longa**: Na fase longa acontece o processamento das $C - (T - 1)$ diagonais internas restantes do bloco.

3.2 CUDAlign 2.0

O CUDAlign 2.0 [20] tem como principal ideia encontrar algumas das coordenadas presentes no alinhamento ótimo de sequências longas de DNA e incrementá-las iterativamente até que o alinhamento ótimo completo seja obtido. Para isso, são combinados os algoritmos de Gotoh (Seção 2.2.3) e Myers-Miller (MM) (Seção 2.2.4), gerando uma variante que é executada em 6 estágios, sendo alguns deles ignorados em caso de grande similaridade entre as sequências.

1. **Obtenção do Escore Ótimo**: Neste estágio, são calculadas as matrizes de Gotoh utilizando a técnica de *wavefront* com a delegação de células explicada na Seção 3.1. Além disso, são geradas linhas especiais com coordenadas armazenadas em disco e o escore ótimo é obtido.

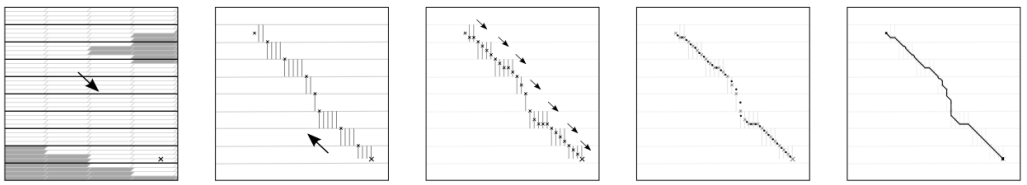
2. **Traceback Parcial**: É executado um alinhamento semi-global no sentido reverso a partir das linhas especiais e do ponto na matriz onde ocorre o escore máximo. Para tanto, utiliza-se a execução ortogonal (variante de MM) com o processamento coluna a coluna, ao invés de linha a linha, e alguns *crosspoints* (Seção 2.2.4) são obtidos.

3. **Divisão de Partições**: São obtidos mais *crosspoints* com partições definidas com início e fim.

4. **Myers-Miller otimizado**: Nesta etapa, o algoritmo de Myers-Miller é otimizado, executando na CPU entre cada par de *crosspoints*.

5. **Obtenção do alinhamento completo**: O alinhamento completo é obtido a partir da concatenação de cada um dos alinhamentos das partições.

6. **Visualização**: Neste estágio opcional, é possível visualizar o alinhamento de forma textual e gráfica.



(a) Estágio 1 (b) Estágio 2 (c) Estágio 3 (d) Estágio 4 (e) Estágios 5/6

Figura 3.5: Estágios de execução do CUDAlign 2.0 [4] .

3.3 CUDAlign 2.1

Com o intuito de acelerar o processamento da matriz de programação dinâmica, foi desenvolvido o CUDAlign 2.1 [4] com a técnica de descarte de blocos (*Block Prunning*). Esta versão baseia-se no CUDAlign 2.0 (Seção 3.2) com a modificação de processamento no Estágio 1, em que o escore ótimo só é conhecido no final do processamento.

O *Block Prunning* consiste em calcular a possibilidade de o melhor alinhamento passar por determindado bloco. Dependendo da similaridade existente entre as sequências comparadas, muitos blocos podem ser descartados, evitando assim, cálculos desnecessários de blocos que não contribuem para o alinhamento ótimo.

3.3.1 Block Pruning

Ao considerarmos as sequências S_0 de tamanho m e S_1 de tamanho n a serem comparadas, podemos definir diversos termos [5]:

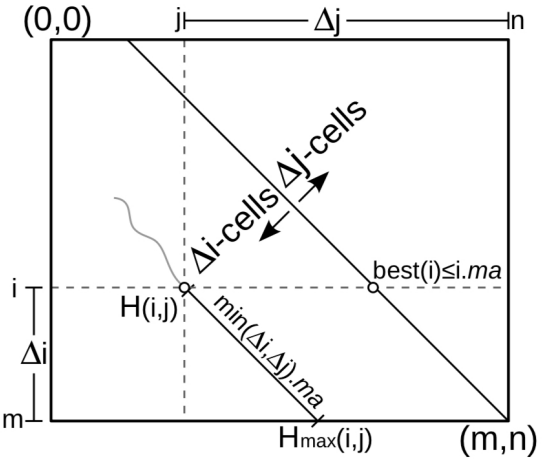


Figura 3.6: Definições geométricas do *Pruning* (Fonte:[5]) .

- Função $p(i, j)$: pontuação obtida entre $S_0[i]$ e $S_1[j]$, que pode ser igual a ma (*match*) ou mi (*mismatch*).
- Distância-i (Δ_i): Distância entre a linha i e a última linha da matriz.
- Distância-j (Δ_j): Distância entre a coluna j e a última coluna da matriz.
- $\Delta_i - cell$: Elemento que está mais próximo da última linha do que da última coluna ($\Delta_i < \Delta_j$).
- $\Delta_j - cell$: Elemento que está mais próximo da última coluna do que da última linha ($\Delta_j < \Delta_i$).

Calcula-se o score máximo $H(i, j)$ de um alinhamento local que passa por uma célula (i, j) somando o seu score atual ao das células em sua diagonal, considerando um *match* perfeito entre as *substrings* conforme exposto na Equação 3.4.

$$H_{max} = H(i, j) + min(\Delta_i, \Delta_j).ma \tag{3.4}$$

O cálculo do maior escore possível em uma célula de determinada linha i, acontece considerando o match perfeito entre as *substrings* que a antecedem e pode ser definido como $best(i) = i.ma$. Uma célula é podada se $H_{max}(i, j) \leq best(i)$, sem preocupações no alinhamento final. A Figura 3.6 ilustra as definições da geometria da poda explicitadas.

No *Block Pruning*, técnica implementada no CUDAlign 2.1, o procedimento de poda é aplicado a blocos de células utilizando o melhor escore obtido até o momento. A verificação de *pruning* é feita uma vez para todo o bloco considerando o pior cenário, aquele em que a célula com maior escore se encontra no canto superior direito.

3.4 CUDAlign 3.0

Mesmo com a melhora de desempenho proporcionada pelo CUDAlign 2.1 (Seção 3.3), foi constatado que longas sequências ainda apresentavam um tempo de execução consideravelmente alto. A fim de amenizar este problema, foi necessário o desenvolvimento do CUDAlign 3.0 com a introdução do processamento de matrizes em múltiplas GPUS para longas sequências biológicas.

No CUDAlign 3.0, cada GPU calcula uma quantidade de colunas da matriz de programação dinâmica de acordo com a sua capacidade de processamento de linhas por segundo. O balanceamento de carga proporciona uma distribuição uniforme para GPUs iguais e heterogênea para GPUs com capacidades diferentes, alinhando assim o processamento.

Para que seja possível a execução, as máquinas com GPU executam 3 *threads* no *host* (CPU). Uma das *threads* é a gerente, responsável por invocar a execução na GPU e transferir os valores de células de borda para as duas outras *threads*, que são de comunicação e transferem valores de forma assíncrona entre as GPUS por *sockets*.

Priorizando mascarar a latência da comunicação entre as GPUs, foi implementado um sistema de *buffer* circular entre *threads* para entrada e saída de dados. Os *buffers* O_{j-1} são de saída e os I_j são os de entrada, fazendo com que o sistema funcione transferindo informações do *buffer* O_{j-1} de cada máquina com GPU para o de entrada I_j da próxima máquina.

3.5 CUDAlign 4.0

O CUDAlign 4.0 [4] propôs a paralelização da recuperação do alinhamento das sequências com múltiplas GPUs (*traceback*), que primariamente era feito de forma sequencial. Para tanto, foram projetados os métodos *Pipelined Traceback* (PT), que estabelece uma execução em *pipeline* das etapas de 2 a 4 do CUDAlign 2.0 (Seção 3.2) em cada GPU, e o *Incremental Speculative Traceback* (IST), que utiliza das GPUs ociosas para especular *crosspoints* que possam gerar o alinhamento ótimo de maneira mais rápida.

3.5.1 *Pipelined Traceback* (PT)

Na estratégia *Pipelined Traceback*, um *pipeline* para as etapas 2, 3 e 4 do CUDAlign é executado paralelamente em várias GPUs.

Findada a execução do primeiro estágio do CUDAlign, inicia-se o *pipeline* na última GPU, que executa os cálculos do estágio 2 e encaminha à unidade da esquerda (GPU_{i-1}) o *crosspoint* de sua coluna intermediária. Este procedimento se repete até alcançar a primeira GPU e gera uma interdependência entre as partições e as unidades. Concluído o estágio 2, todas as GPUs podem prosseguir sozinhas em *pipeline* com nas etapas 3 e 4.

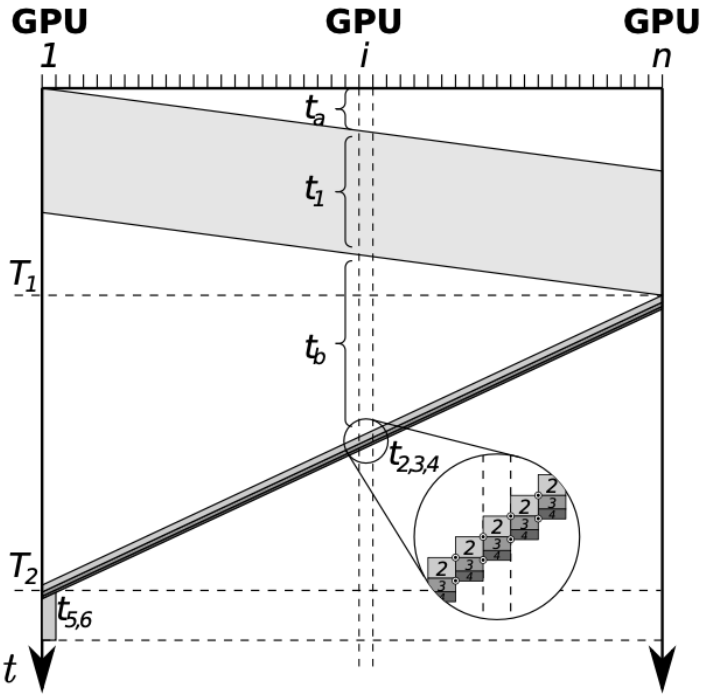


Figura 3.7: *Pipelined Traceback* no CUDAlign 4.0 (Fonte: [4]) .

A Figura 3.7 traz um gráfico das GPUs (eixo x) em relação ao tempo de execução (eixo y). Nele, a área t_1 evidencia o preenchimento do *wavefront* por cada GPU (estágio

1) e as áreas $t_{2...6}$ representam o *traceback*. As regiões t_a e t_b retratam o tempo ocioso da GPU_i nos estágios 1 e 2, respectivamente.

3.5.2 *Incremental Speculative Traceback* (IST)

O *Incremental Speculative Traceback* é uma técnica de *traceback* que aproveita dos momentos de ociosidade da GPU no estágio 2 e do fato que os *crosspoints* encontrados nas colunas intermediárias costumam coincidir com os locais de máximo destas colunas para especular um alinhamento ótimo. Com essa especulação, torna-se possível o processamento em paralelo da etapa 2.

Assim como explicado na Seção 3.5.1, o cálculo dos *crosspoints* também será executado a partir da última GPU. Ao mesmo tempo, as outras GPUs calculam seus *crosspoints* com o escore especulado.

Ao receber o *crosspoint* calculado pela GPU_{i+1}, a GPU_i verifica se os *crosspoints* calculados por ela estão corretos. Se estiverem, ela passa os *crosspoints* para a GPU_{i-1} sem a necessidade de mais processamento. Senão, a partição é recalculada com o *crosspoint* correto.

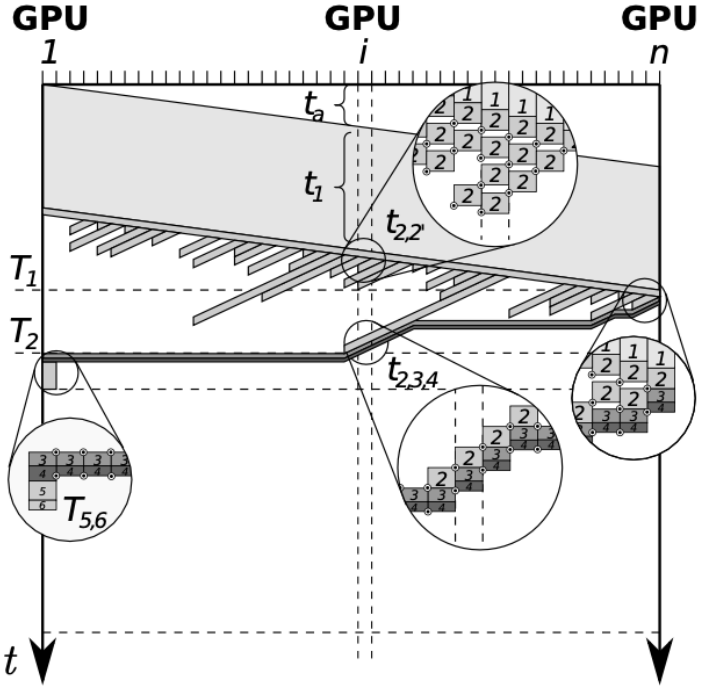


Figura 3.8: Gráfico GPUs x Tempo para o *Incremental Speculative Traceback* (Fonte: [4])
.

Diferentemente do que acontece na Figura 3.7, na etapa $t_{2,2'}$ da Figura 3.8 é possível notar a tentativa das GPUs de encontrar *crosspoints* que pertençam ao alinhamento

ótimo. Neste processo, elas estabelecem comunicações com as unidades adjacentes e podem refazer a etapa 2 diversas vezes. É evidente também a economia de tempo na comparação das técnicas PT e IST [21].

3.5.3 A Arquitetura MASA

O *Multi-platform Architecture for Sequence Aligners* (MASA) é uma arquitetura de *software* feita para facilitar a criação de ferramentas de alinhamento de DNA, como o CUDAlign, em diferentes plataformas de hardware e software. O caráter modular do MASA permite que desenvolvedores integrem novos algoritmos e métodos no *framework* e criem ferramentas personalizadas de alinhamento.

O CUDAlign foi integrado ao MASA, gerando o MASA-CUDAlign. A Figura 3.9 apresenta a arquitetura MASA e suas implementações. O bloco (a) da figura evidencia os módulos do MASA, incluindo as otimizações que podem ser selecionadas pelos desenvolvedores como *Block Pruning* e *Parallelization Strategy* . Em (b), são ilustrados os múltiplos alinhadores desenvolvidos a partir do MASA [6].

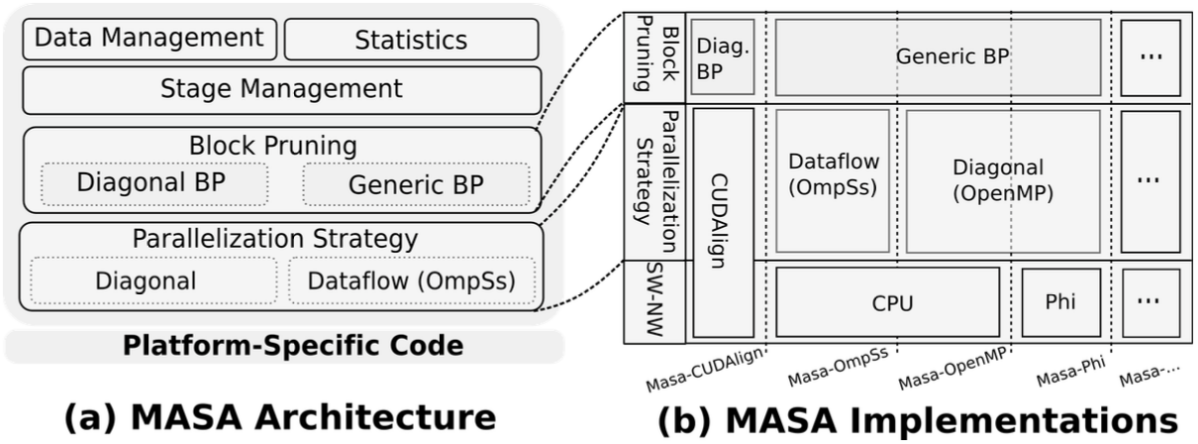


Figura 3.9: Arquitetura MASA e suas implementações [6].

Capítulo 4

Projeto do Módulo de Alinhamento com Poda Agilizada (APA)

Neste capítulo, serão apresentados detalhes acerca do desenvolvimento do Módulo APA, projeto proposto neste trabalho de graduação. A Seção 4.1 descreve o objetivo do projeto. Em 4.2, uma visão geral do módulo é relatada. Logo em seguida, as Seções 4.3 e 4.4 detalham o desenvolvimento de cada Sub-Módulo arquitetado.

4.1 Objetivo

O processamento da matriz de programação dinâmica com *Block Pruning* (descarte de blocos) foi uma otimização proposta no MASA-CUDAlign 2.1 (Seção 3.3) que objetivou agilizar o alinhamento de sequências. Com esta melhoria, blocos de células que matematicamente não contribuem para o resultado final do alinhamento podem ser descartados.

Ao se analisar a inicialização do processamento do MASA-CUDAlign (1 GPU) com o *Block Pruning*, é possível constatar que o escore inicial de poda adotado é zero e é incrementado conforme a matriz é processada. À vista deste fato, surgiu a proposta de inicialização da ferramenta com um escore inicial mais próximo do escore ótimo, aspirando um maior descarte de blocos e uma possível redução no tempo de execução. Tal proposta se baseia na suposição de que o escore inserido serviria como uma antecipação do escore que o MASA-CUDAlign alcançaria em algum momento do processamento. Dessa forma, blocos que inicialmente seriam processados pelo MASA-CUDAlign por conta dos cálculos baseados em baixos escores, agora poderiam ser podados com o escore obtido pelo módulo APA.

Uma das melhores formas de se obter um escore inicial mais próximo ao ótimo é recorrer aos algoritmos heurísticos de comparação, visto que trazem um escore próximo ao ótimo e com certa agilidade. Combinado a isto, destacou-se também a possibilidade de

realizar o *trim* das sequências de DNA muito longas, reduzindo seu tamanho e agilizando ainda mais o processamento do algoritmo heurístico.

Dessa forma, este trabalho de graduação teve como objetivo a criação de um módulo capaz de opcionalmente reduzir o tamanho sequências, obter escores heurísticos e inseri-los no MASA-CUDAlign para execução, tudo isso de maneira sequencial e automatizada.

4.2 Visão Geral

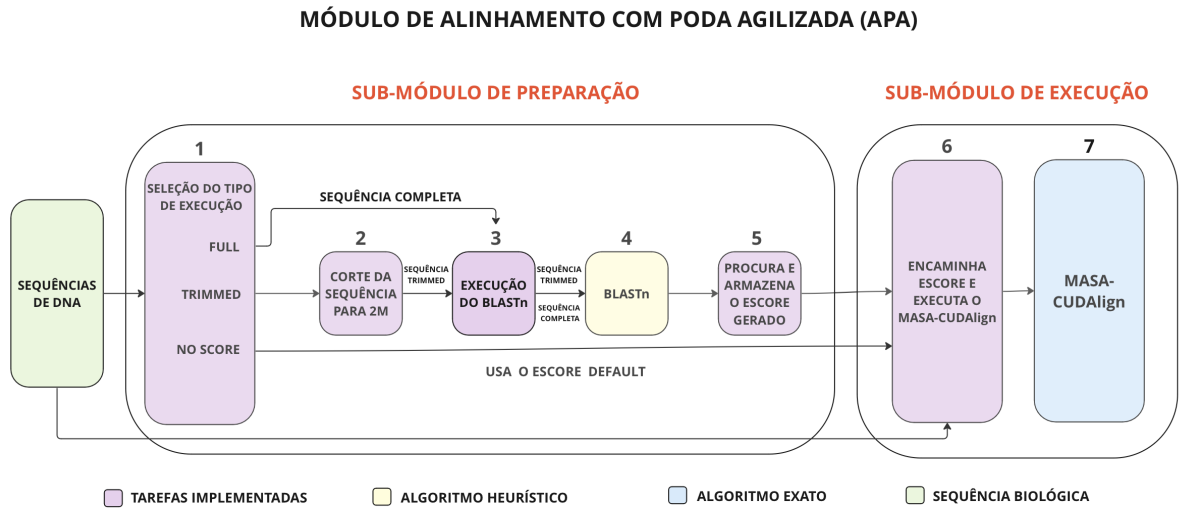


Figura 4.1: Arquitetura do Módulo APA com seus Sub-Módulos.

Buscando a melhoria de desempenho almejada pela inserção de um escore inicial heurístico no *Block Pruning* do MASA-CUDAlign, foi desenvolvido o Módulo APA em C++. A escolha desta linguagem de programação foi feita considerando a familiaridade com suas estruturas e a paridade com a linguagem usada no próprio MASA-CUDAlign.

A arquitetura que baseou o desenvolvimento do projeto foi pensada em duas etapas, Preparação e Execução. Na etapa de Preparação, são feitos todos os trâmites necessários para se obter o escore heurístico a ser inserido no MASA-CUDAlign. Nela, o usuário pode obter resultados distintos partindo de 3 diferentes opções de escolha. Já na etapa de Execução, executa-se o MASA-CUDAlign com o escore colhido na etapa anterior.

A Figura 4.1 apresenta a arquitetura do módulo desenvolvido. É possível observar nela os dois sub-módulos do projeto. Cada um deles pode ser composto por tarefas implementadas no projeto (cor roxa), pela ferramente BLASTn, que executa o algoritmo heurístico, (cor amarela) e pela ferramenta MASA-CUDAlign, que executa o algoritmo exato (cor azul).

O sub-módulo de Preparação incorpora as seguintes funções numeradas de 1 a 5 na figura:

1. **Seleção do Tipo de Execução:** Esta rotina fornece opções para a geração do escore de alinhamento aplicado ao MASA-CUDAlign, facilitando a comparação dos resultados gerados. O usuário pode escolher entre as opções *Full*, *Trimmed* e *No Score* .
2. **Corte na Sequência (TRIM):** Esta função foi projetada para manter apenas os 2 *megabytes* iniciais de cada uma das sequências e descartar o restante.
3. **Execução do BLASTn:** O algoritmo de alinhamento heurístico BLASTn é invocado automaticamente através da linha de comando para o cálculo do escore entre sequências completas ou *trimmed*.
4. **BLASTn:** Executa o alinhamento heurístico entre as sequências selecionadas e gera um arquivo *output* com o escore e o alinhamento.
5. **Busca e Armazenamento do Escore Gerado:** Encontra o escore heurístico no arquivo *output* gerado.

O sub-módulo de Execução incorpora as rotinas 6 e 7 :

6. **Execução do MASA-CUDAlign**: Chama o MASA-CUDAlign através da linha de comando passando o escore inicial e diversos outros parâmetros.
7. **MASA-CUDAlign:** Algoritmo de alinhamento exato que foi modificado para receber o escore inicial de *Block Prunning* através da linha de comando.

Ao longo das Seções 4.3 e 4.4, serão apresentados detalhadamente os sub-módulos de Preparação e Execução, assim como as rotinas e algoritmos que os compõe.

4.3 Sub-Módulo de Preparação

O Sub-Módulo de Preparação foi a principal e mais dispendiosa etapa de desenvolvimento do Módulo APA, haja vista que é ele o responsável por fornecer o escore inicial heurístico a ser inserido no MASA-CUDAlign. Para tal, a preparação conta com 5 diferentes tarefas, cada uma com suas funções e especificidades.

A Seleção do Tipo de Execução (marcada com número 1 na Figura 4.1) é a primeira rotina a atuar no sub-módulo e, apesar de sua relevância, foi a última a ser adicionada ao projeto. Ela é responsável por dar ao usuário a opção de escolher a forma com que é

gerado o escore inicial heurístico a ser inserido no MASA-CUDAlign. As opções variam entre *Full*, *Trimmed* e *No Score*.

Ao selecionar a opção *Full*, palavra do inglês para cheio ou completo, o usuário indica que o escore deve ser gerado a partir das sequências completas, ou seja, o algoritmo heurístico recebe as sequências de DNA originais sem cortes e as processa inteiramente, gerando o escore correspondente. O maior problema deste modo é que, dependendo do tamanho das sequências, o alinhamento demanda bastante tempo para ser executado.

A preocupação com o tempo que levaria para o processamento de grandes sequências combinada com o conhecimento de que no início das sequências o aumento do escore é mais intenso, fez com que o modo *Trimmed* (aparado) fosse o primeiro a ser desenvolvido. Ao selecioná-lo, o escore inicial será gerado a partir de sequências aparadas para o tamanho definido empiricamente de 2 Megabytes, buscando uma pontuação significativa com um tempo de execução razoável. Dessa forma, o modo realiza um alinhamento heurístico semi-global.

O modo *No Score* (sem escore), mais simples dos três, apenas executa o MASA-CUDAlign com seu escore padrão de menos infinito, não trazendo nenhuma alteração a execução original.

Em grande parte do desenvolvimento do Módulo APA, o *Trimmed* era a única opção de execução. Somente ao final do trabalho, viu-se necessária a implementação das outras opções para a obtenção e comparação de resultados, culminando no surgimento da rotina de Seleção do Tipo de Execução. Na linha de comando, os modos são escolhidos utilizando as *flags* "-t", "-f" e "-ns" para *Trimmed*, *Full* e *No Score* respectivamente.

A segunda rotina do sub-módulo (número 2 na Figura 4.1) é a de Corte na Sequência (TRIM) e tem como papel receber as sequências completas e reduzi-las a 2MB. Para tanto, o comando do *Listing* 4.1 foi executado no terminal em AWK [1], linguagem de programação interpretada que permite a manipulação de textos a partir de padrões, para cortar o arquivo da linha 1 à 29538 e descartar o restante. Como os arquivos contendo as bases nitrogenadas são padronizados, realizou-se apenas uma regra de três simples para chegar em 29538 como o número de linhas que corresponderiam a um arquivo de 2MB.

Listing 4.1: Comando AWK para Trim das Sequências

```
awk 'NR==1, NR==29538' arquivo.fasta > arquivo_trimmed.fasta
```

A terceira etapa da Preparação é a Execução do BLASTn (número 3 na Figura 4.1). Ela atua invocando o executável do algoritmo através da linha de comando 4.2, passando como parâmetro dois arquivos contendo as sequências e os valores 5 para G_{open}, 2 para

[1] baeldung.com/linux/awk-guide

G_{ext}, -3 para *mismatch* e 1 para *match* e direcionando o resultado para a um arquivo de saída *output*.

Listing 4.2: Comando para Execução do BLASTn

```
./blastn -subject arquivo1.fasta -query arquivo2.fasta
-gapopen 5 -gapextend 2 -penalty -3 -reward 1
-num_alignments 1 > output
```

O BLAST (Basic Local Alignment Search Tool), ferramenta básica de pesquisa de alinhamento local, foi desenvolvido para realizar buscas heurísticas através da comparação de sequências biológicas contra um banco de dados que contém uma grande quantidade de informações, retornando sequências similares à submetida [11]. Sua variação para nucleotídeos, o BLASTn, compõe a quarta etapa do Sub-Módulo de Preparação (número 4 na Figura 4.1) e foi escolhido por ser a ferramenta heurística mais utilizada na Bioinformática. Além disso, ele é rápido e facilmente parametrizável, sendo possível definir os valores para *match*, *mismatch* e *gaps* através da linha de comando. Após a execução do alinhamento das sequências, a ferramenta retorna o resultado do processamento contendo o escore da comparação e o alinhamento.

Por fim, a quinta e última rotina é a de Procura e Armazenanmento do Escore Gerado (número 5 na Figura 4.1) e tem como objetivo extrair o escore de alinhamento do arquivo de saída do BLASTn. Como o *output* é padronizado, foi possível verificar que escore fica contido na primeira expressão "Score = escore em bits (escore)" que aparece, dessa forma, a lógica adotada para a rotina foi de percorrer todas as linhas do arquivo à procura da que tem "Score =" e, em seguida, salvar o escore que se encontra entre os parênteses. A Figura 4.2 é exemplo de parte de um arquivo de saída do BLASTn, nele é possível verificar da linha 10 à 27, informações gerais sobre as sequências a serem comparadas, na linha 29, o padrão "Score = 1.236e+07 bits (6235710)" contendo o escore 6235710 que necessita ser extraído e, a partir da linha 33, o alinhamento dos nucleotídeos.

O Pseudocódigo 1 representa a função "findScore", elaborada para encontrar e armazenar o escore do arquivo de saída do BLASTn. Ela foi desenvolvida centrada em um *loop* "while" (linha 2) que roda enquanto o arquivo não chega ao final e a variável "score" tiver o valor padrão de menos infinito [2]. Este *loop* lê linha a linha e, caso a expressão "Score =" seja encontrada (linha 4), o escore contido entre parênteses é armazenado na variável "score" (linha 5). Caso o arquivo termine sem encontrar a expressão, a mensagem "Não foi possível gerar um escore pelo BLASTn. Foi aplicado o escore padrão -INF." é impressa (linha 10). Ao final da função, a pontuação é retornada (linha 12).

[2] Apesar do valor inicial de escore ser teoricamente zero, na ferramenta MASA-CUDAlign ele é definido por padrão como menos infinito (-999999999).

Algorithm 1 Função findScore

```
1: score = -INF
2: while o arquivo não chegar ao final e score = -INF do
3:    lê a linha do arquivo
4:    if "Score =" na linha then
5:       score = número contido entre "(" e ")"
6:       print "Best Score gerado pelo BLASTn:" + score
7:    end if
8: end while
9: if score = -INF then
10:   print "Não foi possível gerar um escore pelo BLASTn. Foi aplicado o escore padrão
      -INF."
11: end if
12: return score
```

4.4 Sub-Módulo de Execução

O Sub-Módulo de Execução, segundo a compor o Módulo APA, é responsável por executar o MASA-CUDAlign com o escore inicial do BLASTn. Uma importante escolha que guiou o desenvolvimento tanto deste sub-módulo como de todo o projeto, foi a de utilizar MASA-CUDAlign (1 GPU). Tal decisão foi tomada levando em consideração que esta versão da ferramenta possibilita a geração não só do escore ótimo mas também do alinhamento ótimo, o que pode ter bastante utilidade para pesquisadores interessados na comparação de partes do material genético de diferentes organismos. Além disso, a versão multi-GPU do MASA-CUDAlign (Seção 3.4) não foi escolhida por não adotar o *Block Pruning*.

Apesar de ser uma etapa com um número menor de tarefas, a Execução também foi bastante trabalhosa, uma vez que ainda não existia a funcionalidade de inserção de um escore inicial para *Block Pruning* no código do MASA-CUDAlign. Sendo assim, várias modificações no programa original tiveram que ser realizadas.

Primeiramente, as modificações foram feitas com o objetivo de possibilitar o recebimento do valor do escore inicial através da linha de comando. Para tanto, foram realizadas alterações na biblioteca *libmasa* para a criação do parâmetro - -opening-score, que indica o recebimento de uma *string* na linha de comando contendo o escore, conforme explicitado no *Listing* 4.3. Criou-se também, na classe *Job* do MASA-CUDAlign, a variável *opening_score* para o armazenamento do valor de entrada e o método *setOpeningScore* para a transformação da *string* recebida em um inteiro.

Posteriormente, viu-se a necessidade de inserção do escore armazenado na variável *opening_score* na classe *AbstractBlockPruning* do MASA-CUDAlign, presente em um arquivo contendo rotinas gerais utilizadas por todas as opções de poda existentes. Com

```
BLASTN 2.13.0+


Reference: Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb
Miller (2000), "A greedy algorithm for aligning DNA sequences", J
Comput Biol 2000; 7(1-2):203-14.



Database: User specified sequence set (Input: bases/10M/NC_014318.1.fasta).
           1 sequences; 10,236,715 total letters



Query= gi|384145136|ref|NC_017186.1| Amycolatopsis mediterranei S699
chromosome, complete genome

Length=10236779
                                                                      Score        E
Sequences producing significant alignments:                          (Bits)     Value

gi|300781937|ref|NC_014318.1| Amycolatopsis mediterranei U32 chro...  1.236e+07  0.0


> gi|300781937|ref|NC_014318.1| Amycolatopsis mediterranei U32
chromosome, complete genome
Length=10236715

 Score = 1.236e+07 bits (6235710),  Expect = 0.0
 Identities = 6236381/6236572 (99%), Gaps = 47/6236572 (0%)
 Strand=Plus/Plus

Query  3451908  GGGTCAGCAGCCGCGCCGCCCCGGCCATCGCGGGGCGGGCGGCCGGATCCCGGTGGAGCA  3451967
                ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  3451880  GGGTCAGCAGCCGCGCCGCCCCGGCCATCGCGGGGCGGGCGGCCGGATCCCGGTGGAGCA  3451939
```

Figura 4.2: Exemplo de arquivo de saída do BLASTn.

esse fim, criou-se a variável global *openingScore* para receber e atribuir o valor ao parâmetro *best_score*, usado no *AbstractBlockPruning* para armazenar o melhor escore do momento. Com todas essas modificações foi possível, enfim, integrar o Módulo APA com o MASA-CUDAlign.

Para invocar o MASA-CUDAlign (número 6 na Figura 4.1), o comando 4.3 é executado passando os parâmetros *–ram-size=500M e –disk-size=2G*, que servem para limitar a quantidade de memória utilizada para o armazenamento de dados temporários para 500MB e o espaço de disco utilizado em 2GB. Além disso, a diretiva *–opening-score=37192*, mencionada anteriormente, é usada para transmitir o escore inicial, que nesse caso é 37192. Logo em seguida, são passados os endereços dos arquivos contendo as sequências completas a serem comparadas. O arquivo contendo o alinhamento ótimo

Listing 4.3: Comando para a Execução MASA-CUDAlign

```
./cudalign --ram-size=500M --disk-size=2G
--opening-score=SCORE arquivo1.fasta arquivo2.fasta
```

será autimaticamente salvo em ./work.tmp/alignment.00.txt.

A Figura 4.3 apresenta o início do alinhamento produzido pelo MASA-CUDAlign e a Figura 4.4 apresenta as estatísticas referentes ao tempo de execução.

```
Query: gi|47118314|dbj|BA000035.2| Corynebacterium efficiens YS-314 DNA, complete genome (3147090)
Sbjct: gi|41349502|emb|BX927147.1| Corynebacterium glutamicum ATCC 13032, IS fingerprint type 4-5, complete genome [1..3282708](3282708)

Query:  2977391 TTAAGTTGTTGGTCGGCGGTCACCTACTCTCCCACACCCTCCCGAGTGCAGTACCATCAG  2977451
                ||||||| |||||||||||| ||||||||||||||||||||||||||||||||||||| [52/52]
Sbjct:  2675375 TTAAGTTTTTGGTCGGCGGTAACCTACTCTCCCACACCCTCCCGAGTGCAGTACCATCAG  2675435


Query:  2977451 CGTAACCAGGCTTAGCTTCCGGGTTCGGAATGGGACCGGGCGTTTCCCTGCTACTATCAC  2977511
                |||||||||||||||||||||||||||||| |||||||||||||||||||||||||||| [56/108]
Sbjct:  2675435 CGTAACCAGGCTTAGCTTCCGGGTTCGGAAAGGGACCGGGCGTTTCCCTGCTACTATCAC  2675495


Query:  2977511 CACCGACACACCTATCACGGTGCACCAACCCAAC---CATGTTGGGCGTGTTGTGCCAGA  2977568
                |||||||||| | | |||     ||||| |||||   || |||||  |||||||| |||| [-4/104]
Sbjct:  2675495 CACCGACACAACCA-CAC-----ACCAAACCAACCAACAGGTTGGTTGTGTTGTGTCAGA  2675549


Query:  2977568 CACTGCATAGTGGACGCGAACA----ACGTGTTCAAATTGTGT--GTTACATAAGCGCTT  2977622
                ||||||||| ||||||||||||    | ||||| || |||| |  ||||| |  |||  | [-9/95]
Sbjct:  2675549 CACTGCATAATGGACGCGAACACTTTATGTGTT-AATTTGTTTTCGTTACGT-TGCG--T  2675605


Query:  2977622 TAACATACCCCAACACCATTAGTGGGGTGTTTGTGTCGGTAAATTAGTACCAGTCACCTC  2977682
                | |||    |||| |||  || |||  ||||  | |||||||||||||||||||||| [-1/94]
Sbjct:  2675605 T-ACA----CCAA-ACC-GTAATGGTTTGTTGTTATCGGTAAATTAGTACCAGTCACCTC  2675658

```

Figura 4.3: Arquivo do MASA-CUDAlign contendo o alinhamento.

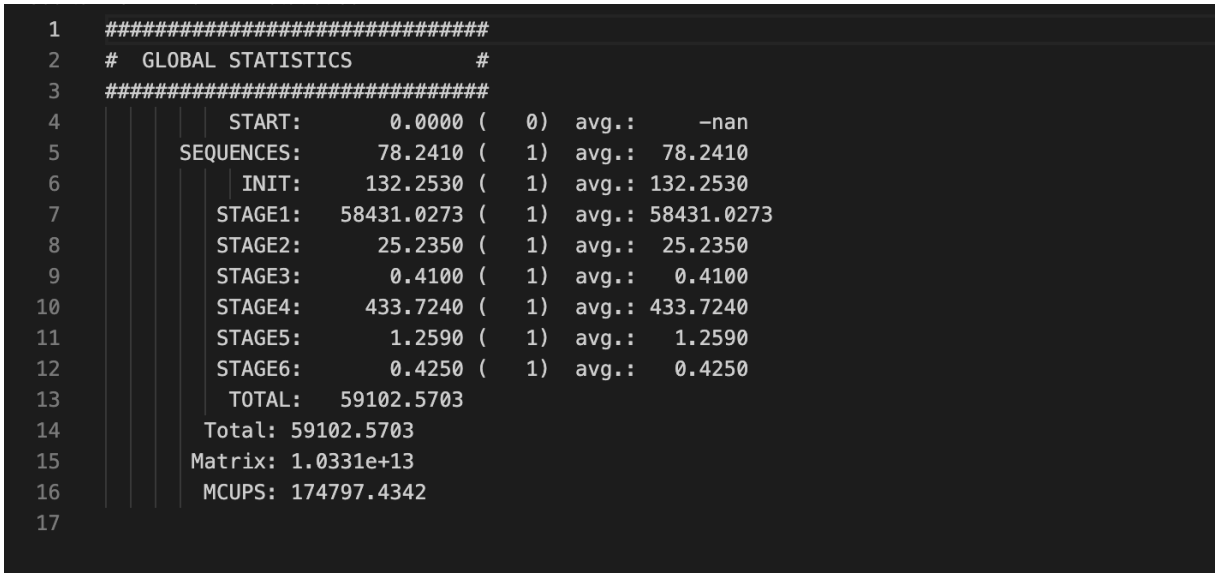


Figura 4.4: Arquivo do MASA-CUDAlign contendo estatísticas.

Capítulo 5

Resultados Experimentais

O Módulo APA, integrado ao MASA-CUDAlign, foi testado em diversas situações e gerou vários resultados experimentais que serão apresentados e discutidos detalhadamente ao longo deste capítulo. A Seção 5.1 descreve o ambiente em que o projeto foi desenvolvido e testado. Na Seção 5.2, as sequências utilizadas para a comparação são apresentadas. Por fim, a Seção 5.3 traz os resultados.

5.1 Ambiente de Testes

O desenvolvimento e os testes do projeto proposto neste trabalho de graduação foram feitos integralmente no Laboratório de Sistemas Integrados e Concorrentes (LAICO) do Departamento de Ciência da Computação (CIC) da Universidade de Brasília. Utilizou-se uma máquina Dell com 16 GiB de memória RAM, processador Intel Core i7-9700, placa gráfica NVidia GeForce RTX 2060 e 1,3 TB de disco. Além disso, o computador conta com um sistema operacional Linux Ubuntu 20.04.1 e compilador gcc na versão 9.4.0 .

Na etapa de Preparação, foi realizado o download do BLASTn (versão 2.13.0) compatível com o sistema operacional Linux, diretamente do site do *National Center of Biotechnology Information* (NCBI) [22]. Já na Execução, fez-se necessária a adequação do ambiente CUDA 10.1, previamente instalado no computador do laboratório, através da mudança de diretório de bibliotecas e arquivos. O código fonte do MASA-CUDAlign foi clonado do *github* do Edans F. O. Sandes na versão 4.0.2.128 [23].

A fim de executar o módulo desenvolvido neste trabalho de graduação, foram utilizados os comandos *bash* apresentados no *Listing* 5.1. O comando 1 utiliza o *git* para clonar o repositório contendo o módulo de alinhamento com poda agilizada diretamente do *github* do autor deste trabalho [14]. Com o comando 2, entra-se no diretório geral do módulo e depois no específico com o MASA-CUDAlign. A instrução 3 configura os diretórios do computador para a execução da ferramenta, apontando os locais onde se encontram a

Listing 5.1: Comandos para download e execução do módulo APA com o MASA-CUDAlign

```
1 git clone https://github.com/mataugustosp/masa-cudalign-apa.git
2 cd masa-cudalign-apa/masa-cudalign
3 ./configure --with-cuda=/usr/lib/cuda --with-nvcc=usr/bin
4 make
5 cd ..
6 g++ modulo_apa.cpp -o apa
7 ./apa [options] seq1.fasta seq2.fasta
```

biblioteca do CUDA (–with-cuda=/usr/lib/cuda) e o compilador nvcc da NVidia (–with-nvcc=usr/bin). Findando a compilação do MASA-CUDAlign aciona-se, no comando 4, o processo automatizado de compilação (make). Depois disso, retorna-se ao diretório base com o comando 5.

A compilação do módulo, feita com o compilador GNU para C++, é invocada com o comando 6, definindo "apa" como nome para o executável gerado. Por fim, com a instrução 7, o módulo é executado passando as opções -f (Full) , -t (Trimmed) ou -ns (No Score) e os arquivos com as sequências.

5.2 Sequências Utilizadas

Para que os testes do Módulo APA integrado ao MASA-CUDAlign fossem efetuados, utilizou-se 6 pares de sequências, conforme mostrado na Tabela 5.1. Todas elas, assim como as de outros diversos organismos, podem ser encontradas no site do National Center for Biotechnology Information (NCBI) em diversos formatos. Para facilitar a buscas de sequências de bases nitrogenadas no site, foi de grande relevância o uso da diretiva apresentada no *Listing* 5.2, dado que ela possibilita a busca de sequências de acordo com o tamanho.

	Sequência A	**Sequência B**	**Escore**		
Comparação	**Acession**	**Acession**	**Ótimo**	**Heurístico**	**Heurístico T.** [1]
3M-3M	BA000035.2	BX927147.1	4226	3888	3888
5M-5M	AE016879.1	AE017225.1	5220960	849975	831053
7M - 7M	NC_003997.3	NC_005027.1	172	119	NA
10M -10M	NC_014318.1	NC_017186.1	10235188	6235710	2067218
23M-23M	NT_033779.4	NT_037436.3	9063	9059	8153
40M-40M	OX253915.1	OX254009.1	9247283	119799	119799

Tabela 5.1: Sequências de testes e seus respectivos escores.

Listing 5.2: Diretiva para busca de sequências por tamanho no NCBI

```
1 tamanho minimo:tamanho maximo[Sequence Length]
```

A Tabela 5.1 apresenta cada um dos 6 pares de sequências utilizados para testes, cada uma com seus respectivos identificadores (*acession*) e escores de comparação. O escore foi dividido em 3 diferentes colunas para registrar o valor do escore de alinhamento ótimo proveniente do MASA-CUDAlign, do escore heurístico do BLASTn e do escore heurístico com sequências cortadas para 2M (*trimmed*).

	Sequência A	**Sequência B**
Tam.	**Nome**	**Nome**
3M	*Corynebacterium efficiens*	*Corynebacterium glutamicum*
5M	*Bacillus anthracis str. Ames*	*Bacillus anthracis str. Sterne*
7M	*Bacillus anthracis str. Ames*	*Rhodopirellula baltica*
10M	*Amycolatopsis mediterranei U32*	*Amycolatopsis mediterranei S699*
23M	*Drosophila melanogaster*	*Drosophila melanogaster*
40M	*Vitis cinerea var. helleri x Vitis rupestris*	*Vitis cinerea var. helleri x Vitis riparia*

Tabela 5.2: Sequências de testes e seus respectivos nomes.

É importante ressaltar que cada comparação é feita com o DNA inteiro ou segmentado de organismos reais como plantas, animais e bactérias. Pensando nisso, a Tabela 5.2 traz as sequências expostas anteriormente com seus nomes científicos.

5.3 Tempos de Execução

Nesta seção, serão apresentados resultados dos testes realizados no MASA-CUDAlign-APA com as sequências ilustradas na Tabela 5.1. Para cada um dos pares, foram extraídos resultados para as opções *No Score* (sem escore inicial, MASA-CUDAlign original), *Trimmed Score* (escore inicial gerado pelo BLASTn com as sequências de 2M) e *Full Score* (escore inicial gerado pelo BLASTn com as sequências completas). A fim de mensurar as mudanças, utilizou-se como referência a porcentagem de blocos podados, o tempo total de execução em segundos, o tempo do MASA-CUDAlign em segundos e a velocidade total em GCUPS, unidade de medida que significa *Giga Cells Per Second*, ou seja, a quantidade de células da matriz de programação dinâmica processadas por segundo na ordem do bilhão.

[1]Heurístico *Trimmed*

É relevante destacar que a extração do tempo de execução para o modo *No Score* contabilizou apenas o tempo do MASA-CUDAlign, ao passo que nos modos *Trimmed Score* e *Full Score*, este valor foi somado ao tempo do Módulo APA, que conta com a computação do escore heurístico do BLASTn e outros processamentos.

5.3.1 Sequências de 3M

SEQUÊNCIAS 3M				
	Prunned Blocks (%)	**Tempo Total (s)**	**Tempo Cudalign (s)**	**Velocidade (GCUPS)**
No Score	0,12	60,34	60,34	171,18
Trimmed Score	0,12	62,05	60,49	170,77
Full Score	0,12	62,90	60,42	170,97

Tabela 5.3: Porcentagem de blocos podados, tempo de execução e velocidade do processamento de sequências de 3M

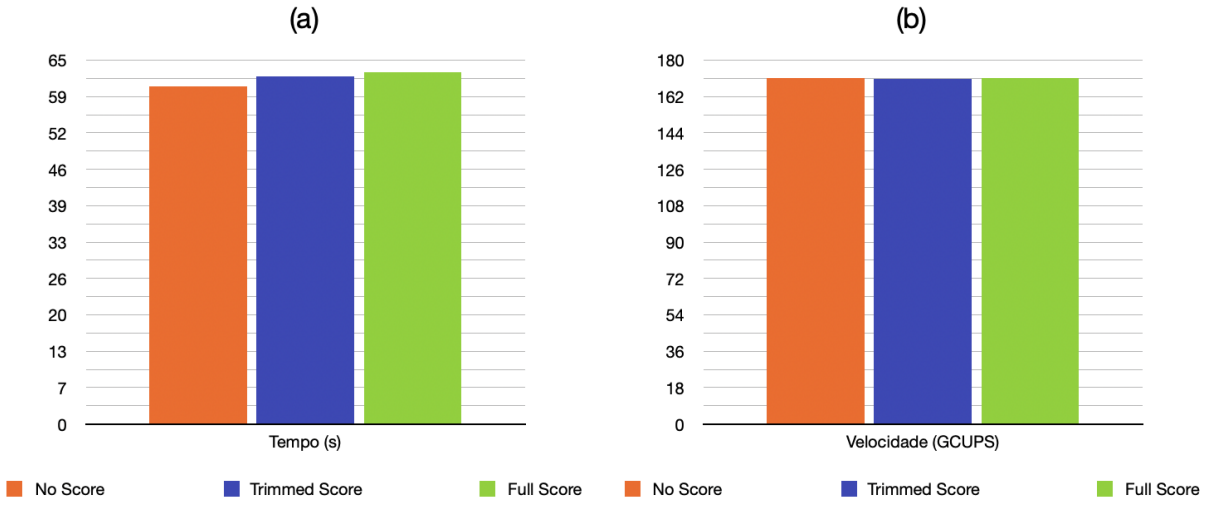


Figura 5.1: Gráficos de tempo de execução (a) e de velocidade (b) para sequências de 3M.

Os alinhamentos do MASA-CUDAlign-APA foram realizados das sequências menores para as maiores. O primeiro experimento foi feito com o genoma completo das bactérias *Corynebacterium efficiens* (cepa YS-314) e *Corynebacterium glutamicum* (cepa ATCC 13032), que são bastante distintas.

Através das pontuações registradas na Tabela 5.1, é possível perceber que os escores heurísticos *Full* e *Trimmed* têm o mesmo valor, acontecimento que pode ser justificado pelo fato de o tamanho dos arquivos originais (3M) ser muito próximo ao tamanho dos

arquivos cortados (2M) e as sequências não serem muitos similares. Este escore é 92% do escore ótimo do MASA-CUDAlign.

Com o auxílio dos resultados registrados na Tabela 5.3, é possível perceber que a porcentagem de blocos podados é baixa e se manteve a mesma em todos os modos. Isso acontece porque as sequências são muito diferentes, o que faz com que o escore seja muito baixo se comparado ao tamanho das sequências. Como o *pruning* é feito com base no maior escore obtido até o momento e o escore entre essas sequências é pequeno, a área de poda é pequena e tende a se manter a mesma.

Em relação aos parâmetros de velocidade e tempo, percebe-se um pequeno aumento no tempo total de execução e uma pequena diminuição da velocidade, mudanças geradas em função do tempo de execução adicional proveniente do Módulo APA. Através dos gráficos da Figura 5.1, nota-se que as diferenças de tempo e velocidade, apesar de existirem, são pouco significativas.

Nesta comparação, o melhor tempo de execução continuou sendo o *No Score* . Assim, é possível inferir que o uso do Módulo APA não é vantajoso para sequências pequenas e dissemelhantes.

5.3.2 Sequências de 5M

SEQUÊNCIAS 5M				
	Prunned Blocks (%)	**Tempo Total (s)**	**Tempo Cudalign (s)**	**Velocidade (GCUPS)**
No Score	53,69	133,33	133,33	205
Trimmed Score	54,35	128,22	125,25	218
Full Score	54,37	129,64	124,06	220

Tabela 5.4: Porcentagem de blocos podados, tempo de execução e velocidade do processamento de sequências de 5M

As sequências de 5M correspondem aos genomas das bactérias *Bacillus anthracis str. Ames* e *Bacillus anthracis str. Sterne* (Tabela 5.2), organismos bastante semelhantes biologicamente. O alinhamento delas gerou um alto escore ótimo e escores heurísticos com valores significativamente grandes, conforme exposto na Tabela 5.1.

Diferentemente do o que acontece nas sequências de 3M, o alinhamento dos genomas de 5M conta com uma porcentagem alta de blocos podados, chegando a 53,691% do total. A inserção dos escores *Trimmed* e *Full* na computação foi capaz de aumentar ainda mais esta porcentagem.

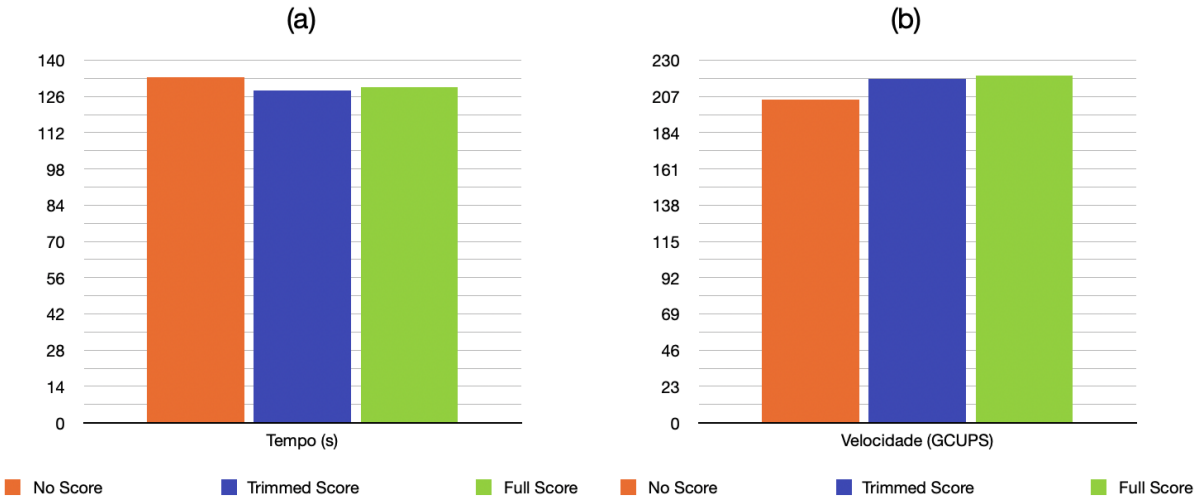


Figura 5.2: Gráficos de tempo de execução (a) e de velocidade (b) para sequências de 5M.

O tempo de execução e a velocidade acompanharam as melhorias relatadas. Os casos *Trimmed Score* e *Full Score*, mesmo com a adição do tempo do Módulo APA, atingiram tempo total menor que o do *No Score*. No quesito velocidade, o progresso também foi notável, apresentado ganhos acima de 6% e tendo como melhor caso a versão *Full*. Os gráficos da Figura 5.2 ilustram as diferenças de tempo e velocidade entre os três modos. O melhor caso registrado foi o do *Trimmed Score* por ter o menor tempo de execução.

Todas melhoras registradas podem ser justificadas pela grande semelhança entre as sequências. Em um cenário com sequências maiores e igualmente similares, inferiu-se que os resultados seriam ainda mais animadores, por isso, um experimento parecido foi realizado para sequências de 10M (Seção 5.3.4).

5.3.3 Sequências de 7M

SEQUÊNCIAS 7M				
	Prunned Blocks (%)	**Tempo Total (s)**	**Tempo Cudalign (s)**	**Velocidade (GCUPS)**
No Score	0	212,47	212,47	176
Trimmed Score	0	217,94	216,51	173
Full Score	0	215,98	213,87	175

Tabela 5.5: Porcentagem de blocos podados, tempo de execução e velocidade do processamento de sequências de 7M

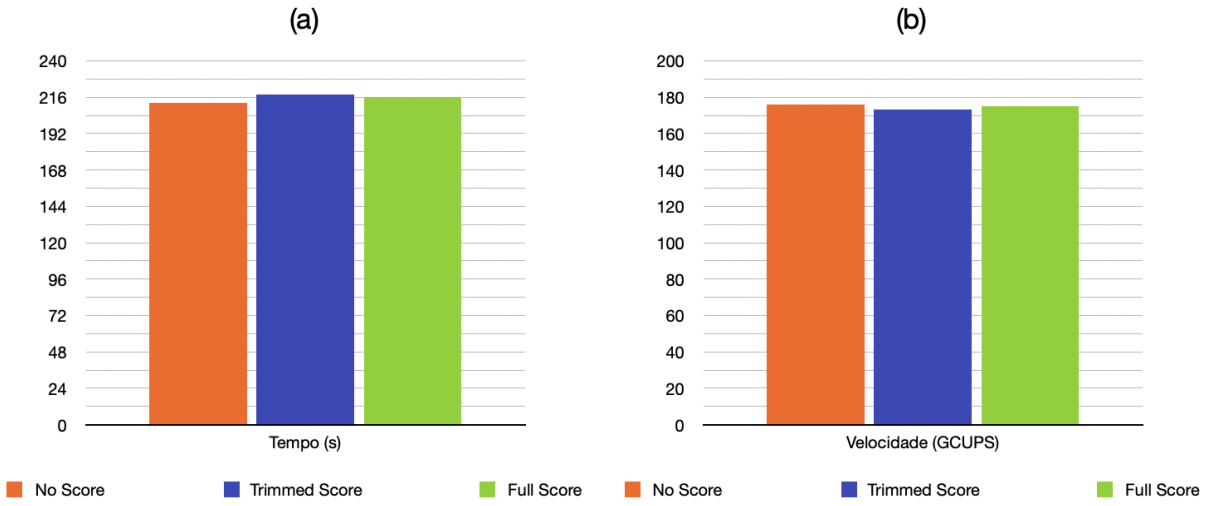


Figura 5.3: Gráficos de tempo de execução (a) e de velocidade (b) para sequências de 7M.

Foram analisadas nesta etapa as sequências genéticas das bactérias *Bacillus anthracis str. Ames* e *Rhodopirellula baltica*, que são extremamente diferentes. Por este motivo, pontuações obtidas para os alinhamentos ótimo e heurístico *full* mostraram-se ínfimas em relação ao tamanho das sequências. Além disso, o BLASTn não foi capaz de gerar um escore para as sequências cortadas em 2M (trimmed).

O baixo escore gerou uma porcentagem de blocos podados de zero para todos os modos. Enquanto isso, o tempo de execução aumentou e a velocidade diminuiu. Tendo em vista o que se passou com os resultados para sequências de 3M, este comportamento já era esperado. As semelhanças entre ambos os casos pode ser comparada com auxílio dos gráficos das Figuras 5.3 e 5.1

5.3.4 Sequências de 10M

SEQUÊNCIAS 10M				
	Prunned Blocks (%)	**Tempo Total (s)**	**Tempo Cudalign (s)**	**Velocidade (GCUPS)**
No Score	53,33	442,88	442,88	237
Trimmed Score	54,91	424,44	421,19	249
Full Score	68,85	370,68	349,38	300

Tabela 5.6: Porcentagem de blocos podados, tempo de execução e velocidade do processamento de sequências de 10M

Figura 5.4: Gráficos de tempo de execução (a) e de velocidade (b) para sequências de 10M.

Após os bons resultados obtidos no alinhamento das sequências de 5M, que são similares, questionou-se sobre o que aconteceria na computação de sequências maiores com a mesma característica. Nesta etapa, foram comparadas duas montagens dos genomas da bactéria *Amycolatopsis mediterranei* (U32 e S699), sequências de 10M bastante parecidas. Os resultados encontram-se na Tabela 5.6.

A primeira análise feita foi a do escore (Tabela 5.1), que se mostrou excelente tanto para o caso ótimo quanto para os heurísticos. Consequentemente, a porcentagem de blocos podados, que já era de 53,5% no caso base, foi elevada para 54,91% para o modo em que há inserção de um escore heurístico vindo das sequências reduzidas (*Trimmed Score*) e para 68,85% com o escore para sequências completas (*Full Score*).

O tempo de execução também foi bastante aprimorado, diminuindo significativamente com o uso dos escores iniciais heurísticos. Proporcionalmente, a quantidade de células processadas por segundo aumentou. O melhor caso dentre os três analisados foi o do *Full Score*, tendo em vista que seu tempo foi aproximadamente 16,3% menor que o do caso base. Nesse caso, o tempo foi reduzido de 7 minutos e 22 segundos para 6 minutos e 10 segundos.

Os gráficos da Figura 5.4 mostram o decrescimento do tempo e aumento da velocidade, comportamento vislumbrado na criação do Módulo APA. Os resultados obtidos com as sequências de 10M foram os melhores do projeto.

5.3.5 Sequências de 23M

SEQUÊNCIAS 23M				
	Prunned Blocks (%)	**Tempo Total (s)**	**Tempo Cudalign (s)**	**Velocidade (GCUPS)**
No Score	0,04	3.211	3.211	176
Trimmed Score	0,04	3.247	3.229	174
Full Score	0,04	3.405	3.225	175

Tabela 5.7: Porcentagem de blocos podados, tempo de execução e velocidade do processamento de sequências de 23M

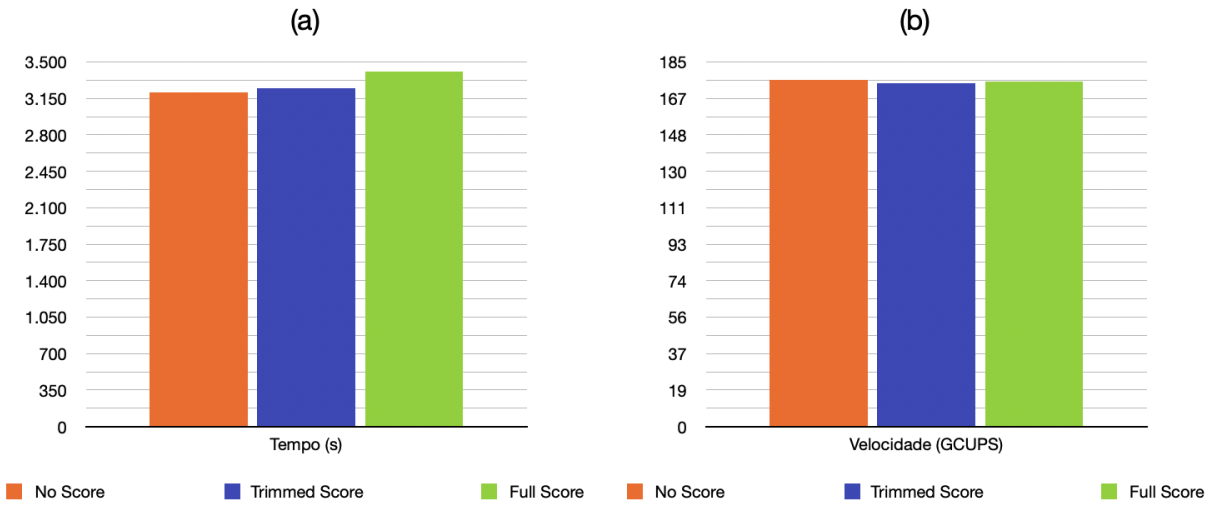


Figura 5.5: Gráficos de tempo de execução (a) e de velocidade (b) para sequências de 23M.

Dois cromossomos distintos da mosca *Drosophila melanogaster*, com tamanho de 23M, foram utilizados para esta comparação. Com auxílio da Tabela 5.1, é possível notar os baixos escores gerados para esta comparação, padrão já visto em sequências muito distintas como as de 3M e 7M.

Acompanhando o escore pequeno, a porcentagem de blocos podados também é baixa, apresentando o valor de 0,04% e não crescendo ao longo da execução . Enquanto isso, o tempo total aumentou e a velocidade diminuiu. Todos os dados das comparações estão ilustrados na Tabela 5.7.

O melhor caso aconteceu no modo *No Score*, haja vista que seu tempo de execução foi o menor. O estudo das sequências de 23M foi muito importante para o projeto porque

evidenciou o fato de que, independentemente do quão grande as sequências são, a poda não melhora se elas forem muito diferentes.

5.3.6 Sequências de 40M

SEQUÊNCIAS 40M				
	Prunned Blocks (%)	**Tempo Total (s)**	**Tempo Cudalign (s)**	**Velocidade (GCUPS)**
No Score	35,32	7.005	7.005	235
Trimmed Score	35,32	7.252	7.070	236
Full Score	35,32	7.972	7.185	236

Tabela 5.8: Porcentagem de blocos podados, tempo de execução e velocidade do processamento de sequências de 40M

Figura 5.6: Gráficos de tempo de execução (a) e de velocidade (b) para sequências de 40M.

Os resultados experimentais para as sequências de 40M foram semelhantes aos das de 3M, 7M e 23M, entretanto, por um motivo diferente. As plantas *Vitis cinerea var. helleri x Vitis rupestris* e *Vitis cinerea var. helleri x Vitis riparia* são similares e sua comparação gera um escore ótimo significativamente alto, conforme exposto na Tabela 5.1. Ao mesmo tempo, as comparações efetuadas pelo BLASTn mostram-se pouco eficientes, originando escores baixos que não colaboraram para o aumento da porcentagem de blocos *prunned*, mantida para todos os modos do experimento.

Acompanhando o esperado para um baixo escore heurístico, o tempo de execução sofreu um aumento e a velocidade se manteve estável. Estes resultados podem ser notados a partir da análise dos gráficos da Figura 5.6.

Capítulo 6

Conclusão e Trabalhos Futuros

O presente trabalho de graduação propôs, implementou e avaliou o Módulo de Alinhamento com Poda Agilizada (APA) associado de forma sequencial ao MASA-CUDAlign. O desenvolvimento do projeto consistiu em obter na ferramenta BLASTn o melhor escore heurístico para sequências completas ou reduzidas e introduzi-lo de forma automática ao escore inicial do MASA-CUDAlign.

Os resultados com o Módulo APA para sequências de DNA de 3M a 40M mostraram que em sequências pouco semelhantes, como é o caso das de 3M, 7M, 23M, o baixo escore ótimo, por conta das próprias características do *Block Pruning*, não viabiliza um aumento na área de poda mesmo com a inserção de um escore inicial heurístico, que obrigatoriamente também é pequeno. Isso acontece porque a poda depende diretamente do melhor escore obtido até o momento, dessa forma, quanto menor o escore, menor o descarte de blocos. Além disso, o tempo total de execução para a comparação destas sequências sofreu pouca ou nenhuma diminuição devido ao período de execução do próprio APA, que agrupa todo o processamento do BLASTn e outras atribuições.

Em contrapartida, sequências semelhantes como as de 5M e 10M trouxeram bons resultados. A comparação entre sequências de 5M apresentou escores próximos ao ótimo tanto para alinhamentos ótimos quanto heurísticos, possibilitando uma leve melhoria na área de poda e na redução do tempo de execução. Já a comparação entre as sequências de 10M gerou mudanças mais significativas na poda, velocidade e tempo de execução. A partir da análise desses casos, foi possível perceber que o APA performa resultados mais próximos do almejado em sequências grandes e semelhantes.

Como trabalho futuro, sugere-se o estudo da possibilidade do processamento do Módulo APA em paralelo ao MASA-CUDAlign, abordagem pensada para diminuir a latência entre o cálculo do escore inicial heurístico e sua atribuição no programa principal. Além disso, levando em consideração o baixo escore heurístico gerado nas sequências de 40M, também seria interessante realizar a busca por algoritmos heurísticos que trouxessem

resultados melhores do que o BLAST, o que proporcionaria um escore mais acurado e aumentaria a poda em comparações com sequências semelhantes. Por fim, propõe-se transformar o Módulo APA em um *plugin*, integrando-o melhor ao próprio MASA-CUDAlign.

Referências

[1] Figueirêdo Júnior, M. A. C. de: *Comparação Paralela de Sequências Biológicas em Múltiplas GPUs com Descarte de Blocos e Estratégias de Distribuição de Carga.* Tese de Doutoramento, Universidade de Brasília, Brasília, Brasil, 2021. viii, 10

[2] Myers, E. e W. Miller: *Optimal alignments in linear space.* Computer applications in the biosciences : CABIOS, 4 1:11–7, 1988. viii, 1, 11

[3] Sandes, E. F. O. e A. C. M. A. Melo: *Cudalign: using gpu to accelerate the comparison of megabase genomic sequences.* In 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP, páginas 137–146, 2010. viii, 16, 17

[4] Sandes, E. F. O. e A. C. M. A. Melo: *Algoritmos Paralelos Exatos e Otimizações para Alinhamento de Sequências Biológicas Longas em Plataformas de Alto Desempenho.* Tese de doutorado, Universidade de Brasília, Brasília, Brasil, 2015. viii, 2, 18, 19, 21, 22

[5] Oliveira Sandes, Edans Flavius de e Alba Cristina Magalhaes Alves de Melo: *Retrieving smith-waterman alignments with optimizations for megabase biological sequences using GPU.* IEEE Trans. Parallel Distributed Syst., 24(5):1009–1021, 2013. `https://doi.org/10.1109/TPDS.2012.194`. viii, 19

[6] Oliveira Sandes, Edans Flavius de, Guillermo Miranda, Xavier Martorell, Eduard Ayguadé, George Teodoro e Alba Cristina Magalhaes Alves de Melo: *MASA: A multiplatform architecture for sequence aligners with block pruning.* ACM Trans. Parallel Comput., 2(4):28:1–28:31, 2016. viii, 23

[7] Diniz, W J S e F Canduri: *Bioinformatics: an overview and its applications.* Genetics and molecular research : GMR, 16(1), March 2017, ISSN 1676-5680. `https://doi.org/10.4238/gmr16019645`. 1

[8] Smith, T. F. e M. S. Waterman: *Identification of common molecular subsequences.* Journal of molecular biology, 147 1:195–7, 1981. 1, 2, 7

[9] Needleman, S. B. e C. D. Wunsch: *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* Journal of molecular biology, 48 3:443–53, 1970. 1, 5, 7

[10] Gotoh, O.: *An improved algorithm for matching biological sequences.* Journal of molecular biology, 162 3:705–8, 1982. 1, 9

[11] Altschul, Stephen F., Warren Gish, Webb Miller, Eugene W. Myers e David J. Lipman: *Basic local alignment search tool*. Journal of Molecular Biology, 215(3):403–410, 1990, ISSN 0022-2836. https://www.sciencedirect.com/science/article/pii/S0022283605803602. 1, 12, 28

[12] NVIDIA: *Cuda*. https://www.nvidia.com/pt-br/technologies/cuda-x/, acesso em 2023-01-25. 1, 13

[13] Sandes, E. O. F. e A. C. M. A. Melo: *Algoritmos paralelos exatos e otimizações para alinhamento de sequências biológicas longas em plataformas de alto desempenho*. Em *Anais do XXIX Concurso de Teses e Dissertações*, páginas 417–422, Porto Alegre, RS, Brasil, 2016. SBC. https://sol.sbc.org.br/index.php/ctd/article/view/9141. 2

[14] Pinho, M. A. S.: *Masa-cudalign*. https://github.com/mataugustosp/masa-cudalign-apa.git, acesso em 2023-02-07. 2, 33

[15] NCBI: *Biological sequences*. https://www.ncbi.nlm.nih.gov/IEB/ToolBox/SDKDOCS/BIOSEQ.HTML, acesso em 2023-01-30. 3

[16] Mount, David W: *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 2004. 3, 4

[17] Durbin, Richard, Sean R. Eddy Anders Krogh e Graeme Mitchison: *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998. 3, 4, 5

[18] Hischberg, D.: *A linear space algorithm for computing maximal common subsequences. communications of the acm*. 1975. 11

[19] Sandes, E. F. O.: *Comparação paralela de sequências biológicas longas utilizando unidades de processamento gráfico (gpus)*. Tese de Mestrado, Universidade de Brasília, Brasília, Brasil, 2011. 13, 15

[20] Sandes, E. F. O. e A. C. M. A. Melo: *Smith-waterman alignment of huge sequences with gpu in linear space.* In IEEE International Parallel Distributed Processing Symposium, páginas 1199–1211, 2011. 18

[21] Oliveira Sandes, Edans Flavius de, Guillermo Miranda, Xavier Martorell, Eduard Ayguadé, George Teodoro e Alba Cristina Magalhaes Alves de Melo: *Cudalign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in GPU clusters*. IEEE Trans. Parallel Distributed Syst., 27(10):2838–2850, 2016. https://doi.org/10.1109/TPDS.2016.2515597. 23

[22] Medicine, National Library of: *Blastn*. https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/, acesso em 2023-01-17. 33

[23] Sandes, E. F. O.: *Masa-cudalign*. https://github.com/edanssandes/MASA-CUDAlign, acesso em 2023-01-18. 33