SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 580

SW# - Biblioteka za poravnanje sljedova korištenjem grafičkih procesora

Matija Korpar

Zagreb, srpanj 2013.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da biste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Bioinformatika se kao znanstvena disciplina razvila s potrebom biologa za prikupljanjem, pohranjivanjem te obradom vrlo velikog broja informacija. Zbog količine podataka kojima se raspolaže (red veličine 10^{12} stanica u tijelu, tri milijarde nukleotida u genu), jednostavni algoritmi iziskuju previše resursa i vremena kod obrade informacija [1]. Kako se količina prikupljenih informacija eksponencijalno povećava, brzina, zahtjevi i točnost budućih algoritama odigrat će važnu ulogu u razvoju same biologije. Omogućiti obradu informacija uz potrošnju realnih resursa i vremena jedna je od zadaća bioinformatike.

Algoritmi ocjenjivanja sličnosti, tj. poravnanja sljedova igraju jednu od ključnih uloga u bioinformatici. Zadaća im je poravnanje dvaju ili više sljedova DNA, RNA i proteina te davanje ocjene sličnosti na temelju poravnanja, dok im je važnost u tome što njihovi rezultati mogu ukazivati na funkcionalne, strukturalne i evolucijske veze između danih sljedova [1]. Sljedove možemo shvatiti kao niz znakova abecede. Iako naizgled jednostavni, navedeni algoritmi zahtijevaju ogromne resurse i njihovo izvođenje traje veoma dugo. Algoritmi ocjenjivanja sličnosti primjenjuju se u:

- procjeni evolucijske srodnosti - ako primjena algoritama na dva slijeda DNA-a daje visoku ocjenu sličnosti, nositelji tih DNA se ocjenjuju evolucijski srodnima. Daljnje analize s drugim DNA sljedovima mogu pronaći i zajedničke pretke [1].
- predviđanju svojstava mutiranih ili novootkrivenih DNA, RNA i proteina - Pronalaženje sličnih sljedova s poznatim svojstvima pomoću algoritama pomaže pri procjeni svojstava novih sljedova [1].
- znanostima nevezanim uz bioinformatiku - poravnavanje sljedova znakova dugo je poznat problem u svijetu obrade teksta, zbog čega se algoritam primjenjuje u računarstvu.

Algoritmi poravnanja i ocjenjivana sljedova DNA-a, RNA-a i proteina čine jedan od središnjih stupova bioinformatike zbog svoje široke primjene. Problem poravna-

nja sljedova jedan je od najstarijih problema u bionformatici. Iako je prošlo nekoliko desetljeća, problem poravnanja je još uvijek aktualan. Trenutna rješenja su kompromisi između potrošnje vremena, memorije, točnosti i brzine. Smith-Watermanov i Needleman-Wunschov algoritam najpoznatiji su deterministički algoritmi poravnanja sljedova. Unatoč svojoj točnosti, navedeni algoritmi poravnanja rijetko se koriste samostalno zbog svoje velike memorijske i vremenske složenosti od $O(mn)$. Ipak, često se koriste kao komponente različitih heurističkih alata. U radu je predstavljena biblioteka SW#. Biblioteka sadrži implementaciju Smith-Watermanova i Needleman-Wunschova algoritma te algoritma poluglobalnog poravnanja na CUDA tehnologiji uz korištenje Hirschbegova i Ukkonenova algoritma, čime se postiže vremenska i memorijska učinkovitost, te implementaciju navedenih algoritama na CUDA tehnologiji prilagođenu poravnavanju skupova parova sljedova.

2. Algoritmi poravnanja sljedova

Poravnanje sljedova može se protumačiti kao način kako jedan slijed pretvoriti u drugi. Prva mjera iz navedenog područja koja govori koliko je operacija potrebno da bi se jedan slijed pretvorio u drugi je Levenshteinova udaljenost[2] (eng. *Levenshtein distance*), poznatija kao *edit distance*. Operacije mogu biti:

- brisanje elementa (eng. *deletion*)
- umetanje elementa (eng. *insertion*)
- zamjena elemenata (eng. *substitution*)

Na primjer, Levenshteinova udaljenost sljedova *čudna* i *cuda* je jednaka dva, tj. potrebno je obaviti dvije operacije (brisanje elementa *n* i zamjena elementa *č* s *c*) kako bi se iz prvog slijeda dobio drugi. Levenshteinova udaljenost pri ocjenjivanju poravnanja sljedova u bionformatici nije dovoljno robusna. Bilo je potrebno razviti sustav koji bi navedene operacije mogao puno fleksibilnije ocjenjivati. Usto, u bioinformatici uz samu ocjenu poravnanja potrebno je izračunati i samo poravnanje sljedova. Prvi algoritam s navedenim svojstvima je Needleman-Wunschov algoritam[3]. Za razliku od Levenshteinove udaljenosti, Needleman-Wunschov algoritam prima ocjene za brisanje i umetanje elemenata te se za ocjenjivanje supstitucija elemenata koristi tablicom koja sadrži ocjene za svaki par elemenata.

k	o	-	-	r	u	z	a
k	u	k	u	r	u	z	-
3	-1	-2	-2	3	3	3	-2

$\sum=5$

Slika 2.1: Primjer rješenja Needleman-Wunschova algoritma.

Uzmimo za primjer da se brisanje i umetanje elemenata ocjenjuje s -2 te da tablica supstitucija vraća ocjenu 3 za iste i ocjenu -1 za različite elemente. Na slici 2.1 prikazano je poravnanje slijedova *kuruza* i *kukuruz* dobiveno korištenjem Needleman-Wunschova algoritma. Ocjena poravnanja jednaka je 5, obavljena je jedna suptitucija,

dva umetanja i jedno brisanje. Ocjena poravnanja Needleman-Wunschova algoritma s ocjenama brisanja i umetanja jednakim -1 te tablicom supstitucija koja vraća 1 za iste elemente i -1 za različite jednaka je Levenshteinovoj udaljenosti. Način ocjenjivanja brisanja i umetanja naziva se ocjenjivanje procjepa.

Needleman-Wunschov algoritam je algoritam globalnog poravnanja, tj. kod poravnanja sudjeluju svi elementi oba slijeda. Smith-Watermanov algoritam obavlja lokolano poravnanje, tj. kod poravnanja ne moraju sudjelovati svi elementi oba slijeda. Algoritam poluglobalnog poravnanja u poravnanju koristi sve elemente jednog slijeda te ne mora koristiti sve elemente drugog slijeda. Svi navedeni algoritmi, metode ocjenjivanja procjepa i metode ocjenjivanja supstitucija opisani su u nastavku.

2.1. Metode ocjenjivanja procjepa

Određivanje cijene, tj. funkcije cijene procjepa kod dizajniranja algoritama poravnanja uvelike ovisi o njenoj primjeni. Npr. za primjene u bioinformatici potrebno je pronaći funkciju cijene procjepa koja će dovoljno dobro opisivati kompleksne funkcije dobivene eksperimentima. Važna svojstva funkcije cijene procjepa su njezina brzina, jednostavnost i mogućnost generalizacije.

Prilikom korištenja funkcija procjepa s deterministčkim algoritmima poravnanja svaka funkcija cijene procjepa mora biti subaditivna i pozitivna, tj. za navedenu funkciju mora vrijediti:

$$\forall \alpha, \beta : f(\alpha + \beta) \leq f(\alpha) + f(\beta) \tag{2.1}$$

$$\forall f(x) : f(x) > 0 \tag{2.2}$$

Kao konačna cijena procjepa duljine n tada se uzima $-f(n)$. Postoje tri česta modela funkcije cijene procjepa:

- konstantna ocjena procjepa - $f(x) = \alpha, \alpha > 0$, ova metoda je u počecima razvoja algoritama ocjenjivanja procjepa bila popularna zbog svoje jednostavnosti, no njeno implementiranje u determinističke algoritme poravnanja je kompleksno te zbog svoje jednostavnosti često daje rezultate netolerantno različite od idealnog modela.
- linearna ocjena procjepa - $f(x) = \alpha x, \alpha > 0$, ova metoda se često koristi zbog svoje jednostavnosti u primjeni s algoritmima poravnanja.

– afina ocjena procjepa - $f(x) = \alpha + \beta(x-1), \alpha, \beta > 0$, ova metoda se često koristi zbog mogućnosti opisivanja većeg broja željenih modela usprkos tome što komplicira sam algoritam poravnanja.

U praksi se koriste linearna i afina ocjena procjepa. Afina ocjena procjepa je najzastupljenija zbog svoje robusnosti. Također, ako pogledamo definicije funkcija, afina funkcija generalizira linearnu, tj. za $\alpha = \beta$, afin model je jednak linearnom. Također, u biološkim primjenama empirijski je dokazano da afin model ocjenjivanja procjepa dobro odgovara empirijskom modelu bioloških procjepa.

2.2. Metode ocjenjivanja supstitucija

Metode ocjenjivanja supstitucija koriste se u algoritmima poravnanja sljedova za procjenu sličnosti pojedinih elemenata sljedova. Najčešće se koriste dva modela, model konstantnog ocjenjivanja supstitucija te model tabličnog zamjenjivanja supstitucija.

Konstantni model ocjenjivanja supstitucija definira se kao:

$$f(x,y) = \begin{cases} m & x = y \\ n & \text{inače} \end{cases} \tag{2.3}$$

gdje su x i y elementi dvaju sljedova, m ocjena sličnosti te n ocjena različitosti. Ovakav model je često korišten u bioinformatici pri poravnavanju nukleotida s parametrima $m = 1$ te $n = -3$.

Tablični model ocjenjivanja supstitucija definira se kao:

$$f(x,y) = M(x,y) \tag{2.4}$$

gdje su x i y elementi dvaju sljedova te je M simetrična tablica gdje su zapisane ocjene sličnosti pojedinih elemenata. Ovakav model je često korišten u bioinformatici pri poravnavanju proteina. Neke od najpoznatijih matrica korištene pri poravnavanju proteina su: BLOSUM_45, BLOSUM_50, BLOSUM_62, BLOSUM_80, BLOSUM_90, PAM_30, PAM_70 te PAM_250 [4]. Ocjene sličnosti u ovim matricima dobivene su empirijskim mjerenjima. Ovaj model također može generalizirati konstantni model ocjenjivanja.

2.3. Algoritmi poravnavanja sljedova

2.3.1. Terminologija

Uvedimo terminologiju za poravnanje sljedova:

- s_1 - prvi slijed
- s_2 - drugi slijed
- m - duljina s_1
- n - duljina s_2
- d - ocjena novog procjepa kod afina ocjenjivanja
- e - ocjena produženja procjepa kod afina ocjenjivanja
- sim(x, y) - funkcija koja vraća ocjenu sličnosti elemenata x i y, $x \in s_1, y \in s_2$

Rješenje algoritama poravnanja (R) može se prikaziti kao uređeni skup:

$$R = \{b_1, e_1, b_2, e_2, s, p\}, \tag{2.5}$$

gdje je:

- b_1 - indeks početka poravnanja na slijedu s_1
- e_1 - indeks kraja poravnanja na slijedu s_1
- b_2 - indeks početka poravnanja na slijedu s_2
- e_2 - indeks kraja poravnanja na slijedu s_2
- s - ocjena poravnanja
- p - put poravnanja

Put poravnanja p je skup koraka kojim se opisuje poravnanje od pozicije b_1 na s_1 i b_2 na s_2 do e_1 na s_1 i e_2 na s_2. Put poravnanja sadrži korake:

- p_i - označava procjep na s_1
- p_d - označava procjep na s_2
- p_a - označava poravnanje elemenata
- p_s - označava kraj puta poravnanja

Svaki od navedenih algoritama ima dva osnovna koraka, korak rješavanja i korak rekonstrukcije. U koraku rješavanja dolazimo do vrijednosti e_1, e_2 i s konačnog rješenja R. U koraku rekonstrukcije dolazimo do ostalih vrijednosti.

2.3.2. Needleman-Wunschov algoritam

Needleman-Wunschov algoritam je najpoznatiji deterministički algoritam globalnog poravnanja, prvi put predstavljen u [3]. Globalno poravnanje osigurava da poravnanje sadrži sve elemente iz s_1 i s_2. Unaprijeđeni Needleman-Wunschov algoritam koji koristi afin model ocjenjivanja procjepa naziva se Gotoh algoritam [5]. Formalna definicija algoritma:

$$H_{i,j} = \begin{cases} 0 & i = 0 \wedge j = 0 \\ -(d + (i-1)e) & j = 0 \\ -(d + (j-1)e) & i = 0 \\ max \begin{cases} E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + \text{sim}(s_{1,i-1}, s_{2,j-1}) \end{cases} & \text{inače} \end{cases} \tag{2.6}$$

$$E_{i,j} = \begin{cases} -\infty & i = 0 \\ max \begin{cases} E_{i,j-1} - e \\ H_{i,j-1} - d \end{cases} & \text{inače} \end{cases} \tag{2.7}$$

$$F_{i,j} = \begin{cases} -\infty & j = 0 \\ max \begin{cases} F_{i-1,j} - e \\ H_{i-1,j} - d \end{cases} & \text{inače} \end{cases} \tag{2.8}$$

$$M_{i,j} = \begin{cases} p_s & i = 0 \wedge j = 0 \\ p_i & H_{i,j} = E_{i,j} \vee i = 0 \\ p_d & H_{i,j} = F_{i,j} \vee j = 0 \\ p_a & \text{inače} \end{cases} \tag{2.9}$$

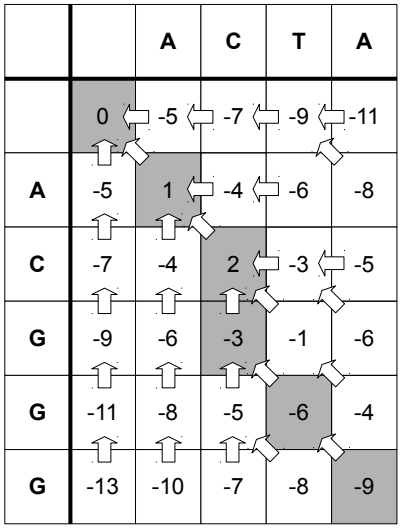
gdje je $i \in [0, m]$, $j \in [0, n]$. Matrice E i F su dodatno uvedene zbog korištenja afinog modela ocjene procjepa. U njima su pohranjene ocjene dobivene isključivo procjepima. Matrica E odgovara procjepima na slijedu s_1, dok matrica F odgovara procjepima na slijedu s_2. Put p se dobiva praćenjem puta iz elementa $M_{m,n}$ do elementa $M_{0,0}$, na način da pojedini korak puta označava sljedeće:

- p_d - umetanje procjepa u s_2
- p_i - umetanje procjepa u s_1

– p_a - poravnanje elemenata s_1 i s_2

– p_s - kraj puta

Korak p_s se izostavlja iz puta. Također, uvode se matrice G_i i G_d koje pamte duljine procjepa na s_1 i s_2 u svakom elementu matrice. Ove matrice su potrebne zbog mogućnosti da procjep nije optimalan cjelom svojom duljinom. Rješenje algoritma tada je $R = \{0, 0, m-1, n-1, H_{m,n}, p\}$. Vremenska i memorijska složenost ovog algoritma su $O(mn)$.



Slika 2.2: Matrica rješavanja Needleman-Wunschova algoritma.

Na slici 2.2 dan je primjer matrice rješavanja Needleman-Wunschova algoritma. Na slici su prikazane vrijednosti matrice H, te pomaci definirani matricom M. Parametri su $d = 5$ i $e = 2$ te je funkcija $\text{sim}(x, y) = 1$ ako $x = y$, -3 inače. Sivim ćelijama označen je put. Rješenje je $R = \{0, 0, 4, 3, -9, \{p_a, p_a, p_d, p_a, p_a\}\}$.

A	C	G	G	G
A	C	–	T	A
1	1	-3	-3	-5

$\sum = -9$

Slika 2.3: Rješenje Needleman-Wunschova algoritma sa slike 2.2.

NW$(s_1, s_2, m, n, d, e, \text{sim}())$:
$H(0,0) = 0$;
$M(0,0) = p_s$;
$G_i(0..m, 0..n) = 0$;
$G_d(0..m, 0..n) = 0$;
for $i \leftarrow 1$ **to** m **do**
    $H(i,0) = -(d + e(i-1))$;
    $M(i,0) = p_d$;
    $E(i,0) = -\inf$;
end
for $j \leftarrow 1$ **to** n **do**
    $H(0,j) = -(d + e(j-1))$;
    $M(0,j) = p_i$;
    $F(0,j) = -\inf$;
end
for $i \leftarrow 1$ **to** m **do**
    for $j \leftarrow 1$ **to** n **do**
        $f = \max(H(i-1,j) - d, F(i-1,j) - e)$;
        $e = \max(H(i,j-1) - d, E(i,j-1) - e)$;
        $h = H(i-1,j-1) + \text{sim}(s_1(i-1), s_2(j-1))$;
        if $f = F(i-1,j) - e$ **then** $G_d(i,j) = G_d(i-1,j) + 1$;
        if $f = E(i,j-1) - e$ **then** $G_i(i,j) = G_i(i,j-1) + 1$;
        $H(i,j) = \max(e, f, h)$;
        if $H(i,j) = e$ **then** $M(i,j) = p_i$;
        if $H(i,j) = f$ **then** $M(i,j) = p_d$;
        if $H(i,j) = h$ **then** $M(i,j) = p_a$;
    end
end
$i = m, j = n$;
$p = \{\}$;
while $M(i,j) != p_s$ **do**
    if $M(i,j) = p_i$ **then** $p = \underbrace{\{p_i, p_i, \ldots, p_i\}}_{G_i(i,j)+1} + p, j = j - G_i(i,j) - 1$;
    if $M(i,j) = p_d$ **then** $p = \underbrace{\{p_d, p_d, \ldots, p_d\}}_{G_d(i,j)+1} + p, i = i - G_d(i,j) - 1$;
    if $M(i,j) = p_a$ **then** $p = M(i,j) + p, j = j - 1, i = i - 1$;
end
return $\{0, 0, m-1, n-1, H(m,n), p\}$;

Slika 2.4: Pseudokod Needleman-Wunschova algoritma.

2.3.3. Smith-Watermanov algoritam

Smith-Watermanov algoritam je najpoznatiji deterministički algoritam lokalnog poravnanja [6]. Lokalno poravnanje određuje da poravnanje ne mora sadržavati sve elemente iz s_1 i s_2. U nastavku je prikazana inačica Smith-Watermanova algoritma koja koristi afin model ocjenjivanja procjepa. Formalna definicija algoritma:

$$H_{i,j} = \begin{cases} 0 & i = 0 \vee j = 0 \\ max \begin{cases} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + \text{sim}(s_{1,i-1}, s_{2,j-1}) \end{cases} & \text{inače} \end{cases} \tag{2.10}$$

$$M_{i,j} = \begin{cases} p_s & H_{i,j} = 0 \vee i = 0 \vee j = 0 \\ p_i & H_{i,j} = E_{i,j} \\ p_d & H_{i,j} = F_{i,j} \\ p_a & \text{inače} \end{cases} \tag{2.11}$$

gdje je $i \in [0, m]$, $j \in [0, n]$. Matrica E jednaka je formuli 2.7, a matrica F jednaka je formuli 2.8. Označimo s S element matrice s najvećom vrijednosti H matrice, tj. $S = argmax(H)$. Put p se dobiva slično kao i kod Needleman-Wunschova algoritma, no umjesto kretanja iz elementa $M_{m,n}$, rekonstrukcija kreće iz elementa S. Također, rekonstrukcija ne staje u elementu $M_{0,0}$ već staje na prvom elementu gdje je $M_{i,j} = p_s$. Označimo taj element s P. Rješenje algoritma tada je $R = \{P_i, P_j, S_i, S_j, H_{S_i,S_j}, p\}$. Vremenska i memorijska složenost ovog algoritma su $O(mn)$.

		A	C	T	A
	0	0	0	0	0
A	0	1	0	0	1
C	0	0	2	0	0
G	0	0	0	0	0
G	0	0	0	0	0
G	0	0	0	0	0

Slika 2.5: Matrica rješavanja Smith-Watermanova algoritma.

Na slici 2.5 dan je primjer matrice rješavanja Smith-Watermanova algoritma. Na

slici su prikazane vrijednosti matrice H te pomaci definirani matricom M. Parametri su $d = 5$ i $e = 2$ te je funkcija $\text{sim}(x, y) = 1$ ako $x = y$, -3 inače. Sivim ćelijama označen je put. Rješenje je $R = \{0, 0, 1, 1, 2, \{p_a, p_a\}\}$.

A	C
A	C
1	1

$\underbrace{\qquad\qquad}_{\sum=2}$

Slika 2.6: Rješenje Smith-Watermanova algoritma sa slike 2.5.

```
1  SW(s_1, s_2, m, n, d, e, sim()):
2  ...
3  for i ← 1 to m do
4  |   H(i,0) = 0, M(i,0) = p_s;
5  |   ...
6  end
7  for j ← 1 to n do
8  |   H(0,j) = 0, M(0,j) = p_s;
9  |   ...
10 end
11 s = −inf, e_1 = 0, e_2 = 0;
12 for i ← 1 to m do
13 |   for j ← 1 to n do
14 |   |   ...
15 |   |   H(i,j) = max(0, e, f, h);
16 |   |   ...
17 |   |   if H(i,j) = 0 then  M(i,j) = p_s ;
18 |   |   if H(i,j) > s then  s = H(i,j), e_1 = i, e_2 = j ;
19 |   end
20 end
21 i = e_1, j = e_2;
22 ...
23 return {i, j, e_1 − 1, e_2 − 1, H(e_1, e_2), p};
```

Slika 2.7: Pseudokod Smith-Watermanova algoritma.

2.3.4. Algoritam poluglobalnog poravnanja

Algoritam je nastao kao mješavina Smith-Watermanova i Needleman-Wunschova algoritma. Uzevši svojstva lokalnog i globalnog poravnanja, ovaj algoritam kao rješenje daje poluglobalno poravnanje. Ovo svojstvo osigurava da poravnanje sadrži sve elemente s_1 dok ono ne mora sadržavati sve elemente s_2. U nastavku je prikazana inačica algoritma poluglobalnog poravnanja koja koristi afin model ocjenjivanja procjepa.

Formalna definicija algoritma:

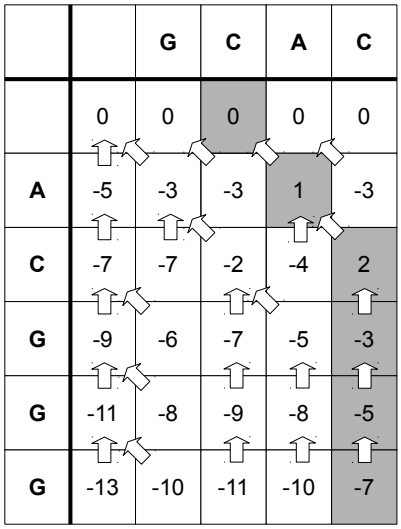
$$H_{i,j} = \begin{cases} 0 & i = 0 \\ -(d + (i-1)e) & j = 0 \wedge i > 0 \\ max \begin{cases} E_{i,j} & \\ F_{i,j} & \text{inače} \\ H_{i-1,j-1} + \text{sim}(s_{1,i-1}, s_{2,j-1}) & \end{cases} & \end{cases} \tag{2.12}$$

$$M_{i,j} = \begin{cases} p_s & H_{i,j} = 0 \vee i = 0 \\ p_i & H_{i,j} = E_{i,j} \\ p_d & H_{i,j} = F_{i,j} \vee j = 0 \\ p_a & \text{inače} \end{cases} \tag{2.13}$$

gdje je $i \in [0, m]$, $j \in [0, n]$. Matrica E jednaka je formuli 2.7, a matrica F jednaka je formuli 2.8. Označimo s S element matrice s najvećom vrijednosti zadnjeg retka H matrice, tj. $S = argmax(H_m)$. Elementi P i put računaju se jednako kao u Smith-Watermanovu algoritmu. Rješenje algoritma tada je $R = \{0, P_j, m, S_j, H_{m,S_j}, p\}$. Vremenska i memorijska složenost ovog algoritma su $O(mn)$.

```
1  HW(s_1, s_2, m, n, d, e, sim()):
2  ...
3  for j ← 1 to n do
4  |   H(0, j) = 0, M(0, j) = p_s;
5  |   ...
6  end
7  s = −inf, e_2 = 0;
8  for i ← 1 to m do
9  |   for j ← 1 to n do
10 |   |   ...
11 |   |   if i = m and H(i, j) > s then s = H(i, j), e_2 = j ;
12 |   end
13 end
14 i = m, j = e_2;
15 ...
16 return {0, j, m − 1, e_2 − 1, H(m, e_2), p};
```

Slika 2.8: Pseudokod algoritma poluglobalnog poravnanja.



Slika 2.9: Matrica rješavanja algoritma poluglobalnog poravnanja.

Na slici 2.9 dan je primjer matrice rješavanja algoritma poluglobalnog poravnanja. Na slici su prikazane vrijednosti matrice H te pomaci definirani matricom M. Parametri su $d = 5$ i $e = 2$ te je funkcija $\text{sim}(x, y) = 1$ ako $x = y$, -3 inače. Sivim ćelijama označen je put. Rješenje je $R = \{0, 2, 4, 3, -7, \{p_a, p_a, p_d, p_d, p_d\}\}$.

A	C	G	G	G
A	C	–	–	–
1	1	-5	-2	-2

$\sum = -7$

Slika 2.10: Rješenje algoritma poluglobalnog poravnanja sa slike 2.9.

2.4. Ostale metode ocjenjivanja poravnanja

Pri ocjenjivanju poravnanja jednog slijeda s bazom sljedova, ako se gleda isključivo ocjena poravnanja kao veličina za vrednovanje poravnanja, može doći do neželjenih rezultata prilikom uspoređivanja poravnanja. Kako su u bazama sljedova često sljedovi različitih duljina, često dolazi do problema *slučajnih* poravnanja. Ako je slijed koji se poravnava s bazom veoma kratak, tada se on može *slučajno* poravnati s duljim sljedovima u bazi. Iz navedenog razloga, uvode se pojmovi P vrijednost i E vrijednost. E vrijednost je najraširenija način ocjenjivanja jednog slijeda s bazom sljedova koja se koristi u brojnim popularnim alatima za poravnanje [7] [8].

P vrijednost predstavlja vjerojatnost da se događaj dogodi s određenom slučajnošću. U kontekstu poravnanja sljedova, P vrijednost neke ocjene poravnanja s ozna-

čava vjerojatnost da se s određenom slučajnošću pojavi ocjena poravnanja x veća ili jednaka ocjeni s. Matematički se P vrijednost definira kao:

$$P(s) = Ke^{-\lambda s} \tag{2.14}$$

E vrijednost predstavlja prepravak parametra P pri višestrukom ocjenjivanju. E vrijednost, u kontekstu ocjenjivanja jednog slijeda s bazom sljedova, je broj nezavisnih poravnanja, s većom ili jednakom ocjenom od s, koji se očekuje da će se dogoditi pri poravnanju s određenom slučajnošću. Što je E vrijednost niža, to je poravnanje s ocjenom s značajnije. Matematički se E vrijednost definira kao:

$$E(s) = mnP(s) = mnKe^{-\lambda s}, \tag{2.15}$$

gdje je m duljina slijeda koji se poravnava s bazom, dok je n zbroj duljina svih sljedova u bazi. Parametri K i λ su unaprijed izračunati statistički parametri. Najčešće se koriste parametri definirani u radu [9]. Iako E vrijednost definirana formulom 2.15 pruža bolju usporedbu poravnanja od same ocjene, u praksi se koriste kompliciraniji modeli za njeno računanje. U nastavku rada prilikom referenciranja na E vrijednost podrazumijevat će se računanje E vrijednosti alata BLAST [7].

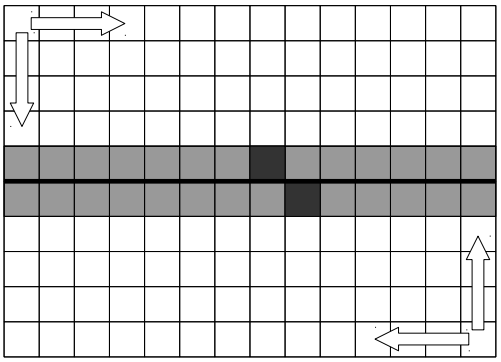
3. Optimizacijski algoritmi primjenjivi na algoritme poravnanja sljedova

3.1. Hirschbergov algoritam

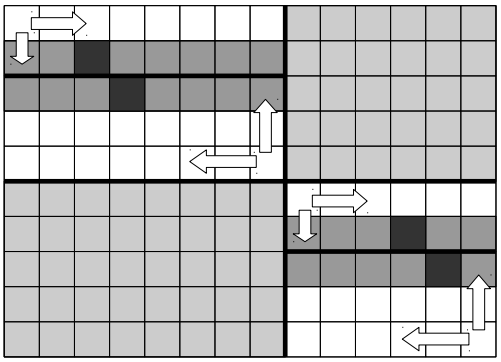
Prilikom razvoja deterministčkih algoritama poravnanja sljedova, kao problem pokazao se njihov veliki zahtjev za računalnim resursima. Algoritmi navedeni u 2.3 imaju memorijsku i vremensku složenost $O(mn)$. Iako je vremenska složenost visoka, mnogo je teže ispuniti zahtjev za kvadratnom količinom memorije. Stoga su se počeli razvijati algoritmi orijentirani na smanjenje potrošnje memorije navedenih algoritama. Jedan od najpoznatijih algoritama s navedenim svojstvima predstavio je Dan Hirschberg u [10] te ga nazvao po sebi Hirschbergov algoritam. Njegov algoritam optimira Needleman-Wunschov algoritam, zadržava vremensku složenost $O(mn)$ (konkretnije 2 puta je sporiji), dok mu je memorijska složenost linearna $O(n)$. Linearizacijom memorijske složenosti, Needleman-Wunschov algoritam, uz korištenje Hirschbergova algoritma, može poravnavati mnogo veće sljedove. Razvojem algoritama i pojavom Gotohova algoritma koji proširuje Needleman-Wunschov algoritam za rad s afinim modelom procjepa, pojavila se i implementacija Hirschbergova algoritma uz Gotohove preinake. Navedeni algoritam naziva se Myers-Millerov algoritam, po njegovim autorima, te je objavljen u [11]. U nastavku je objašnjena upravo Myers-Millerova inačica Hirschbergova algoritma.

Algoritam dijeli matricu rješavanja horizontalno na dvije polovice te radi na principu spajanja polovica puteva rekonstrukcija u jednu cjelinu. Kako bi se dijelovi rekonstrukcije pravilno spojili, potrebno je donju polovicu matrice rješavati u suprotnom smjeru. Rješavanjem Needleman-Wunschova algoritma na gornjoj i donjoj polovici matrice rješavanja dobivamo zadnje retke obje podmatrice. Nadalje, valja pronaći maksimalnu sumu odgovarajućih elemenata dobivenih redaka. Elementi kojima odgovara najveća navedena suma dijelovi su puta rekonstrukcije rješenja. Navedeni postupak vidljiv je na slici 3.1a. Zadnji retci podmatrica označeni su svijetlosivom bojom dok

su elementi kojima odgovara najveća navedena suma označeni tamnosivom bojom. Također, na slici su naznačeni smjerovi rješavanja podmatrica. Postupak se nastavlja rekurzivno tako da se matrica dijeli na podmatrice prema poziciji elemenata kojima odgovra najveća suma. Na slici 3.1b prikazan je drugi korak algoritma. Vidimo da su elementi podijeljeni prema poziciji elemenata s najvećom sumom, označenih na slici 3.1a tamnosivom bojom. Najsvjetlijom sivom bojom označeni su dijelovi matrica koje u drugom koraku više nije potrebno izračunavati. Postupak se nastavlja rekurzivno dok podjela podmatrica više nije moguća.



(a) Prvi korak Hirschbergova algoritma.



(b) Drugi korak Hirschbergova algoritma.

Slika 3.1: Prva dva koraka Hirschbergova algoritma.

U nastavku su prvo objašnjene potrebene funkcije za izvedbu algoritma te je tada detaljnije objašnjen sam Hirschbergov algoritam.

3.1.1. Needleman-Wunschov algoritam s mogućnosti kontrole rubnih procjepa

Needleman-Wunschov algoritam s mogućnosti kontrole rubnih procjepa jednostavna je modifikacija originalnog Needleman-Wunschova algoritma. Razlog navođenja navedenog algoritma je njegova potreba u koracima Hirschbergova algoritma.
Koristimo terminologiju navedenu u 2.3.1 i 2.3.2 te dodatno uvodimo sljedeće logičke parametre:

- g_{f1} - slijed s_1 počinje procjepom.
- g_{b1} - slijed s_1 završava procjepom.
- g_{f2} - slijed s_2 počinje procjepom.
- g_{b2} - slijed s_2 završava procjepom.

U jednom trenutku ne mogu istovremeno biti istiniti parametri g_{f1} i g_{f2} te g_{b1} i g_{b2}. Navedeno se u pseudokodovima podrazumijeva i ne će se provjeravati.

```
1  NWG(s_1, s_2, m, n, d, e, sim(), g_f1, g_b1, g_f2, g_b2):
2  ...
3  for i ← 1 to m do
4      for j ← 1 to n do
5          ...
6          H(i, j) = max(e, f, h);
7          if g_f1 and j = 1 then H(i, j) = e;
8          if g_f2 and i = 1 then H(i, j) = f;
9          if g_b1 and j = n then H(i, j) = e;
10         if g_b2 and i = m then H(i, j) = f;
11         ...
12     end
13 end
14 ...
```

Slika 3.2: Pseudokod Needleman-Wunschova algoritma s mogućnosti kontrole rubnih procjepa gdje su prikazane promjene u odnosu na pseudokod dan slikom 2.4.

Kontrola procjepa obavlja se ignoriranjem optimalnih vrijednosti algoritma u početnom koraku za početne procjepe i završnom koraku za završne procjepe. Postavljanjem vrijednosti $H(i, j)$ na vrijednost e postavlja se i vrijednost $M(i, j)$ na vrijednost p_i, tj. na korak koji označava umetanje procjepa u slijed s_1. S druge strane postavljanjem vrijednosti $H(i, j)$ na vrijednost f postavlja se i vrijednost $M(i, j)$ na vrijednost p_d, tj. na korak koji označava umetanje procjepa u slijed s_2. Vremenska i memorijska složenost algoritma ostaju jednake kao i kod originalnog algoritma te iznose $O(mn)$.

3.1.2. Korak rješavanja Needleman-Wunschova algoritma s linearnom memorijskom složenošću

Hirschbergov algoritam koristi *divide and conquer* metodu rješavanja problema, tj. rekurzivno dijeli veći problem na manje probleme. Memorijska složenost Needleman-Wunschova algoritma je $O(mn)$ te, da bi se mogla primjeniti *divide and conquer* metoda, valja naći dio algoritma koji se može svesti na $O(n)$. U poglavlju 2.3.1 navedeno je da se svaki od opisanih algoritama, pa tako i Needleman-Wunschov algoritam, sastoji od koraka rješavanja i koraka rekonstrukcije. Korak rješavanja moguće je svesti na linearnu memorijsku složenost te će se on koristiti u Hirschbergovu algoritmu za podjelu posla.

```
1  NWS1(s_1, s_2, m, n, d, e, sim()):
2  H(0,0) = 0;
3  for i ← 1 to m do
4  |   H(i,0) = −(d + e(i − 1));
5  |   E(i,0) = − inf;
6  end
7  for j ← 1 to n do
8  |   H(0,j) = −(d + e(j − 1));
9  |   F(0,j) = − inf;
10 end
11 for i ← 1 to m do
12 |   for j ← 1 to n do
13 |   |   e_1 = H(i,j−1) − d;
14 |   |   e_2 = E(i,j−1) − e;
15 |   |   e = max(e_1, e_2);
16 |   |   f_1 = H(i−1,j) − d;
17 |   |   f_2 = F(i−1,j) − e;
18 |   |   f = max(f_1, f_2);
19 |   |   p_1 = sim(s_1(i−1), s_2(j−1));
20 |   |   h = H(i−1,j−1) + p_1;
21 |   |   H(i,j) = max(e, f, h);
22 |   end
23 end
24 s = H(m,n);
25 return s;
```

(a) Korak rješavanja Needleman-Wunschova algoritma s memorijskom složenošću $O(mn)$.

```
1  NWS2(s_1, s_2, m, n, d, e, sim()):
2  H_1(0) = 0;
3  F_1(0) = 0;
4  for i ← 1 to n do
5  |   H_1(i) = −(d + e(i − 1));
6  |   F_1(i) = − inf;
7  end
8  for i ← 1 to m do
9  |   if i == 1 then m_1 = 0;
10 |   else m_1 = −(d + e(i − 2));
11 |   h_1 = −(d + e(i − 1));
12 |   e_1 = − inf;
13 |   for j ← 1 to n do
14 |   |   e = max(h_1 − d, e_1 − e);
15 |   |   f = max(H_1(j) − d, F_1(j) − e);
16 |   |   h = m_1 + sim(s_1(i−1), s_2(j−1));
17 |   |   h_1 = max(e, f, h);
18 |   |   e_1 = e;
19 |   |   m_1 = H_1(j);
20 |   |   H_1(j) = h_1;
21 |   |   F_1(j) = f;
22 |   end
23 end
24 s = H_1(n);
25 return s;
```

(b) Korak rješavanja Needleman-Wunschova algoritma s memorijskom složenošću $O(n)$.

Slika 3.3: Usporedba rješavanja Needleman-Wunschova algoritma s memorijskim složenostima $O(mn)$ i $O(n)$.

Koristimo terminologiju navedenu u 2.3.1 i 2.3.2. Needleman-Wunschov algoritam pamti cijele matrice H, F, E i M. Matrica M potrebna je samo u koraku rekonstrukcije i neće se koristiti u Hirschbergovu algoritmu zbog čega je ignoriramo. Iz formula 2.6, 2.7 i 2.8 zaključujemo da su za izračunavanje elementa na poziciji $H_{i,j}$ potrebni elementi $E_{i,j-1}$, $F_{i-1,j}$, $H_{i,j-1}$, $H_{i-1,j}$ i $H_{i-1,j-1}$. Bez smanjenja općenitosti, pretpostavimo da se matrica H rješava po stupcima zatim po retcima. Tada, zbog ovisnosti rješavanja matrice H, za rješavanje pojedinog elementa moramo pamtiti elemente prethodnog retka matrice F i H te posljednje elemente matrica H i E. Navedene elemente možemo dinamički osvježavati. Uz navedena opažanja, korak rješavanja Needleman-Wunschova algoritma pamti $2n + 2$ elementa matrica, što je memorijska složenost od $O(n)$. Na slici 3.3 prikazana je usporedna implementacija originalnog algoritma i algoritma s manjom memorijskom složenošću. Implementacija nije značajno složenija

dok je memorijska složenost s kvadratne složenosti svedena na linearnu složenost.

Uvedimo u terminologiju pojmove H_1 i F_1 iz algoritma sa slike 3.3b. Nakon izvođenja prikazanog algoritma, u vektoru H_1 bit će pohranjen zadnji red matrice H, dok će u vektoru F_1 biti pohranjen zadnji red matrice F. Također, kao funkciju NWS2 uvodimo funkciju iz algoritma sa slike 3.3b. Ulazni parametri NWS2 funkcije jednaki su terminlogiji navedenoj u 2.3.1.

3.1.3. Korak rješavanja Needleman-Wunschova algoritma prilagođen Hirschbergovu algoritmu

Za korištenje koraka rješavanja Needleman-Wunschova algoritma u koracima Hirschbergova algoritma potrebno je uvesti dodatne modifikacije. Koristimo terminologiju navedenu u 2.3.1, 2.3.2, 3.1.1 i 3.1.2.

```
1  NWH(s_1, s_2, m, n, d, e, sim(), g_f1, g_f2):
2  g_1, g_2 = 0;
3  if g_f1 then g_1 = d − e;
4  if g_f2 then g_2 = d − e;
5  ...
6  for i ← 1 to n do
7  |   H_1(i) = −(d + e(i − 1)) + g_2;
8  |   ...
9  end
10 for i ← 1 to m do
11 |   if i == 1 then m_1 = 0;
12 |   else m_1 = −(d + e(i − 2)) + g_1;
13 |   h_1 = −(d + e(i − 1)) + g_1;
14 |   ...
15 end
16 ...
17 return H_1, F_1;
```

Slika 3.4: Korak rješavanja Needleman-Wunschova prilagođen Hirschbergovu algoritmu gdje su prikazane promjene u odnosu na pseudokod dan slikom 3.3.

Sljedećim logičkim parametrima mijenjamo interpretaciju:

- g_{f1} - smatraj početni procjep slijeda s_1 unaprijed otvorenim.
- g_{f2} - smatraj početni procjep slijeda s_2 unaprijed otvorenim.

Parametri g_{f1} i g_{f2} ne mogu istovremeno biti istiniti. Navedeno znači da će početni procjepi umjesto afine ocjene procjepa $-(d+e(n-1))$ koristiti ocjenu $-en$. Funkcija potrebna za korak Hirschbergova algoritma mora vraćati zadnje retke matrica H i F.

Uvedimo u terminologiju funkciju s navedenim svojstvima kao NWH. Funkcija NWH bazirana je na funkciji NWS2 prikazanoj na slici 3.3 uz prethodno objašnjene dodatne parametre g_{f1} i g_{f2}. Vremenska i memorijska složenost ostaju iste kao i kod funkcije NWS2 te iznose $O(mn)$ i $O(n)$.

3.1.4. Hirschbergov algoritam

Koristimo terminologiju navedenu u 2.3.1, 2.3.2 i 3.1.3. Algoritam dijeli matricu H horizontalno na dvije polovice te radi na principu spajanja polovica puteva rekonstrukcija u jednu cjelinu. Kako bi se dijelovi rekonstrukcije pravilno spojili, potrebno je donju polovicu matrice H rješavati u suprotnom smjeru. Rješavanjem Needleman-Wunschova algoritma, tj. funkcije NWS, na gornjoj i donjoj polovici matrice rješavanja dobivamo zadnje retke matrica H i F obje podmatrice. Nadalje, valja pronaći maksimalnu sumu odgovarajućih elemenata dobivenih redaka. Elementi kojima odgovara najveća navedena suma dijelovi su puta rekonstrukcije rješenja. Pozicija elemenata najveće sume dana je izrazom:

$$c = arg_{max}(\max(H_{1u}(i) + H_{1d}(n-i), F_{1u}(i) + F_{1d}(n-i) + d - e)), i \in 0..n, \quad (3.1)$$

gdje su H_{1u} i F_{1u} zadnji retci matrica H i F gornje podmatrice, dok su H_{1d} i F_{1d} zadnji retci matrica H i F donje podmatrice. Retci H_{1d} i F_{1d} su u obrnutom poretku zbog rješavanja algoritma nad okrenutim sljedovima te su stoga njihovi indeksi ispravljeni prema navedenom zapažanju. Ocjene dobivene iz redaka matrice F ispravljene su vrijednošću $d-e$. Navedeni problem vidljiv je na slici 3.5. Parametri algoritma su $d = 5$ i $e = 2$, te je funkcija $\text{sim}(x, y) = 1$ ako $x = y$, -3 inače. Na slici vidimo da je ocjena poravnanja jednaka -8, umjesto očekivanih -5. Razlog tome je to što je Needleman-Wunschov algoritam na svakoj od polovica matrica kaznio otvaranje novog procjepa, tj. otvaranje procjepa je kažnjeno dva puta. Kako bi ispravili navedeno, rezultatu se dodaje vrijednost $d - e$.

A	A	G	G	|	G	A	A
A	A	–	–	|	–	A	A
1	1	-5	-2	|	-5	1	1

$\sum = -8$

Slika 3.5: Problem procjepa u koraku Hirschbergova algoritma.

Ispravljanje vrijednosti rezultata uzrokuju novi problem. Kako su uzete vrijednosti ispravljene, postoji mogućnost da one nisu optimalne u rješavanoj odgovarajućoj podma-

trici, tj. moguće je da je globalno optimalno poravnanje rezultat dvaju suboptimalnih poravnanja. Kako se polovice matrice rješavaju Needleman-Wunschovim algoritmom koji garantira optimalno rješenje, algoritam se mora promijeniti na način da dozvoljava navedeno prepravljanje rezultata, tj. da se uvede mogućnost da se početni procjepi Needleman-Wunschova algoritma ne kažnjavaju parametrom otvaranja procjepa. Navedena funkcionalnost ostvaruje se parametrima g_{f1} i g_{f2} funkcije NWH objašnjene u poglavlju 3.1.3. Iz razloga što Hirschbergov algoritam dijeli matricu uvijek horizontalno, moguće je jedino dozvoljavanje procjepa vezanih uz slijed s_2, tj. parametar g_{f2} bit će uvijek jednak nuli. Zadane promjene potrebno je rekurzivno propagirati. Za navedeno služe parametri g_{f1} i g_{b1}. Iz istog razloga zbog kojeg se ne koristi parametar g_{f2}, ne će se koristiti parametri g_{f2} i g_{b2} funckije NWG. U terminologiju uvedimo funkciju HIRSCHBERG, koja uz parametre originalne funkcije NW prima i prethodno objašnjene dodatne parametre g_{f1} i g_{f2}. Postupak se nastavlja rekurzivno tako da se matrica H dijeli na podmatrice prema poziciji elemenata kojima odgovara najveća suma, tj. na pozicijama c i $c+1$ sve dok podjela podmatrica više nije moguća. Pseudokod Hirschbergova algoritma prikazan je na slici 3.6.

HIRSCHBERG$(s_1, s_2, m, n, d, e, \text{sim}(), g_{f1}, g_{b1})$:
if $n = 0$ **then return** $\underbrace{\{p_d, p_d, \ldots, p_d\}}_{m}$;
if $m = 0$ **then return** $\underbrace{\{p_i, p_i, \ldots, p_i\}}_{n}$;
if $m = 1$ **then** $R =$ **NWG**$(s_1, s_2, m, n, d, e, \text{sim}(), g_{f1}, g_{b1}, 0, 0)$, **return** $R.p$;
$r = m/2$;
$s_{1u} = s_1(0..r), s_{1d} = s_1(m-1..r+1)$;
$s_{2r} = s_2(n-1..0)$;
$H_{1u}, F_{1u} =$ **NWH**$(s_{1u}, s_2, r, n, d, e, \text{sim}(), g_{f1}, 0)$;
$H_{1d}, F_{1d} =$ **NWH**$(s_{1d}, s_{2r}, m-r-1, n, d, e, \text{sim}(), g_{b1}, 0)$;
$s = -\inf, c = -1, g = 0$;
for $i \leftarrow 0$ **to** n **do**
    $h = H_{1u}(i) + H_{1d}(n-i)$;
    $a = F_{1u}(i) + F_{1d}(n-i) + d - e$;
    if $h > s$ **then** $s = h, c = i, g = 0$;
    if $a > s$ **then** $s = a, c = i, g = 1$;
end
$s_{1u} = s_1(0..r), s_{1d} = s_1(r+1..m-1)$;
$s_{2u} = s_2(0..c), s_{2d} = s_2(c+1..n-1)$;
$p_0 =$**HIRSCHBERG**$(s_{1u}, s_{2u}, r, c, d, e, \text{sim}(), g_{f1}, g)$;
$p_1 =$**HIRSCHBERG**$(s_{1d}, s_{2d}, m-r-1, n-c-1, d, e, \text{sim}(), g, g_{b1})$;
return $p_0 \cup p_1$;

Slika 3.6: Pseudokod Hirschbergova algoritma.

3.1.5. Memorijska i vremenska složenost

Memorijsku složenost algoritma lako je pokazati analizirajući pseudokod na slici 3.6. Iz pseudokoda je vidljivo da se zauzima memorija za stvaranje novih sljedova, što je moguće učinkovito implementirati u $2(n + m)$. Ova ocjena proizlazi iz činjenice da su sljedovi nepromjenjivi stoga nije potrebno stvarati njihove kopije za podsljedove. Dvostruka potrošnja je potrebna jer se čuvaju i reverzni sljedovi. Memorijska složenost ovisi o funkcijama NWG i NWH. U poglavlju 3.1.3 pokazano je da je memorijska složenost funkcije NWH $O(n)$. U poglavlju 3.1.1 pokazano je da je memorijska složenost funkcije NWG $O(mn)$, no pri korištenju funkcije NWG u implemntaciji Hirschbergova algoritma uvijek vrijedi $m = 1$. Stoga, složenost funkcije NWG u navedenom korištenju je $O(n)$. Nadalje, koraci algoritma su međusobno memorijski nevezani. Uz sva navedena zapažanja, lako je zaključiti da je memorijska složenost Hirschbergova algoritma linearna, tj. $O(n)$.

Vremenska složenost Hirschbergova algoritma ovisi o funkcijama NWH i NWG, čije su vremenske složenosti jednake i iznose $O(mn)$, što je pokazano u poglavljima 3.1.1 i 3.1.3. Za dokazivanje vremenske složenosti potrebno je razmotriti slučaj algoritma koji troši najviše vremena. To je upravo slučaj kad se u svakom koraku matrica dijeli točno na četvrtine. Drugim riječima, u prvom koraku dimenzije podmatrice su upola manje nego u nultom koraku, u trećem upola manje nego u drugom, itd.. Označimo s m' i n' veličine podmatrica. Možemo generalizirati:

$$n'_i = \frac{n'_{i-1}}{2}, n'_0 = n, m'_i = \frac{m'_{i-1}}{2}, m'_0 = \frac{m}{2} \tag{3.2}$$

U nultom koraku postoje dvije podmatrice, u prvom četiri, itd.. Ukupnu vremensku složenost tada možemo raspisati:

$$\begin{aligned} t &= 2n'_0m'_0 + 4n'_1m'_1 + 8n'_2m'_2 + \dots \\ &= 2n\frac{m}{2} + 4\frac{n}{2}\frac{m}{4} + 8\frac{n}{4}\frac{m}{8} + 16\frac{n}{8}\frac{m}{16} + \dots \\ &= mn + \frac{mn}{2} + \frac{mn}{4} + \frac{mn}{8} + \dots \\ &= mn\sum_{i=0}^{\inf}\frac{1}{2^i} \\ &= 2mn \end{aligned} \tag{3.3}$$

Iz raspisa vidimo da Hirschbergov algoritam troši duplo više vremena nego originalna verzija Needleman-Wunschova algoritma, no i dalje sadržava složenost $O(mn)$.

3.2. Blokovski Hirschbergov algoritam

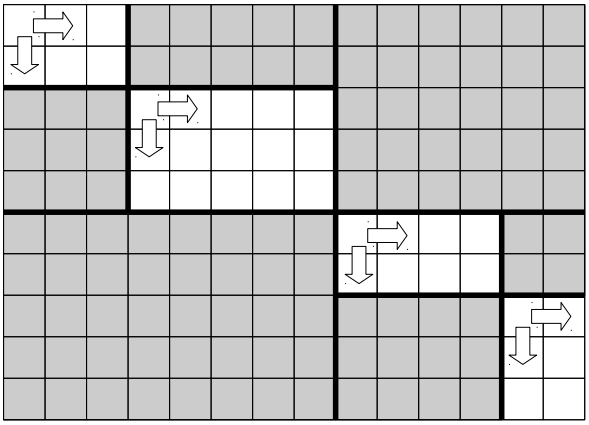
Osnovna ideja blokovskog Hirschbergov algoritma je da Hirschbergovim algoritmom dijeli matricu na podmatrice tako dugo dok veličina podmatrica bude dovoljno mala da je Needleman-Wunschov algoritam na navedenoj podmatrici izvediv. Razlog tome možemo pronaći u činjenici da je Hirschbergov algoritam upola sporiji od Needleman-Wunschova algoritma dok Needleman-Wunschov algoritam troši kvadratnu količinu memorije. Da bi ostvarili navedeno ponašanje, uvodi se novi parametar e_{max}. Za parametar e_{max} vrijedi:

$$0 < e_{max} \leq mn \tag{3.4}$$

Navedenim parametrom ograničava se maksimalni broj elemenata, tj. veličina podmatrice koja se šalje na rješavanje Needleman-Wunschovom algoritmu. Iz pseudokoda na slici 3.7 primjećujemo da je blokovski Hirschbergov algoritam implementacijski veoma sličan originalnom Hirschbergovu algoritmu. Za razliku od originalne verzije, ova verzija algoritma prekida rekurziju kada su podmatrice dovoljno male da se mogu obraditi s memorijskom složenošću $O(mn)$. Navedenim se upravlja dodatnim parametrom e_{max} koji određuje maksimalan broj elemenata podmatrice koje može obraditi funkcija NW. Na slici 3.8 prikazan je posljednji korak blokovskog Hirschbergova algoritma. Navedeni korak je nastavak algoritma na slici 3.1b, no umjesto da nastavi podjelu kao originalni Hirschbergov algoritam, algoritam zaustavlja podjelu i šalje podmatrice na rješavanje pomoću funkcije NW objašnjene u poglavlju 3.1.4. Navedene podmatrice prikazane su bijelom bojom. Podmatrice sive boje nije potrebno izračunavati. Primjetimo da za vrijednost $e_{max} = mn$ blokovski Hirschbergov algoritam postaje Needleman-Wunschov algoritam, dok za vrijednosti $e_{max} = 1$ postaje originalni Hirschbergov algoritam.

```
1 BHIRSCHBERG(s1, s2, m, n, d, e, sim(), gf1, gb1, emax):
2 if n = 0 then return {pd, pd, ..., pd} (m) ;
3 if m = 0 then return {pi, pi, ..., pi} (n) ;
4 if mn ≤ emax then R = NWG(s1, s2, m, n, d, e, sim(), gf1, gb1, 0, 0), return R.p ;
5 ...
6 p0 =BHIRSCHBERG(s1u, s2u, r, c, d, e, sim(), gf1, g, emax);
7 p1 =BHIRSCHBERG(s1d, s2d, m − r − 1, n − c − 1, d, e, sim(), g, gb1, emax);
8 ...
```

Slika 3.7: Pseudokod blokovskog Hirschbergova algoritma gdje su prikazane promjene u odnosu na pseudokod dan slikom 3.6.



Slika 3.8: Posljednji korak blokovskog Hirschbergova algoritma.

3.2.1. Memorijska i vremenska složenost

Prilikom analize memorijske složenosti vrijede sve primjedbe navedene 3.1.5, osim činjenice da se funkcija NWG ne koristi više samo u slučaju kada je $m = 1$. Najveća podmatrica na kojom se može pozvati funkcija NWG ima broj elemenata jednak e_{max}. Uz memorijsku složenost funkcije NWG od $O(mn)$, gdje je mn upravo broj elemenata potrebne matrice, dolazimo do zaključka da je složenost funkcije NWG pri korištenju s blokovskim Hirschbergovim algoritmom $O(e_{max})$. Jer je ostatak složenosti jednak kao i u originalnom algoritmu, dolazimo do zaključka da je ukupna memorijska složenost algoritma $O(n + e_{max})$.

Određivanje vremenske složenosti algoritma mnogo je složenije. Koristimo sličan raspis kao 3.3. Dodatno, za veličine ulaznih vektora u svakom koraku možemo pisati:

$$m_k = \frac{m}{2^k}, n_k = \frac{n}{2^k} \tag{3.5}$$

Potrošnju vremena po koracima algoritma raspisujemo kao:

$$\begin{aligned} t &= 2m'_0n'_0 + 4m'_1n'_1 + \ldots + 2^k m'_{k-1}n'_{k-1} + 2^k m_k n_k \\ &= 2\frac{m}{2}n + 4\frac{m}{4}\frac{n}{2} + \ldots + 2^k\frac{m}{2^k}\frac{n}{2^{k-1}} + 2^k\frac{m}{2^k}\frac{n}{2^k} \\ &= mn + \frac{mn}{2} + \ldots + \frac{mn}{2^{k-1}} + \frac{mn}{2^k} \\ &= mn\sum_{i=0}^{k}\frac{1}{2^i} \\ &= 2mn(1 - \frac{1}{2^{k+1}}) \end{aligned} \tag{3.6}$$

U koraku k funkcija NWH poziva se 2^{k+1} puta, dok se funkcija NWG poziva 2^k puta. Također, u zadnjem koraku, umjesto složenosti funkcije NWH, uvrštavamo složenost funkcije NWG. Navedene promjene u odnosu na raspis 3.3 vidljive su ograničenoj sumi te zadnjem članu sume. Parametar k interpretiramo kao broj koraka potreban da bi broj elemenata podmatrice pao ispod vrijednosti e_{max}. Za najmanji broj koraka k u kojem prekidamo rekurziju možemo pisati:

$$\begin{aligned} e_{max} &\geq \frac{m}{2^k}\frac{n}{2^k} \\ k &\geq \log_2 \sqrt{\frac{mn}{e_{max}}} \end{aligned} \tag{3.7}$$

Bez gubitka općenitosti, jer je k cjelobrojan, možemo pisati:

$$k = \left\lceil \log_2 \sqrt{\frac{mn}{e_{max}}} \right\rceil \tag{3.8}$$

Uvrštavanjem formule 3.8 u formulu 3.6 te sređivanjem jednadžbe dobijemo:

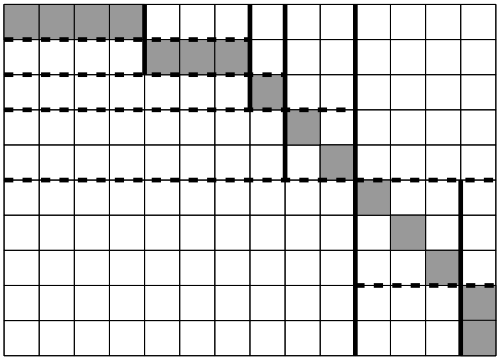
$$t = 2mn\left(1 - 2^{-\left(1+\left\lceil \log_2 \sqrt{\frac{mn}{e_{max}}} \right\rceil\right)}\right) \tag{3.9}$$

Vremensku složenost algoritma tada možemo ocijeniti kao $O\big(2mn - \sqrt{mne_{max}}\big)$.

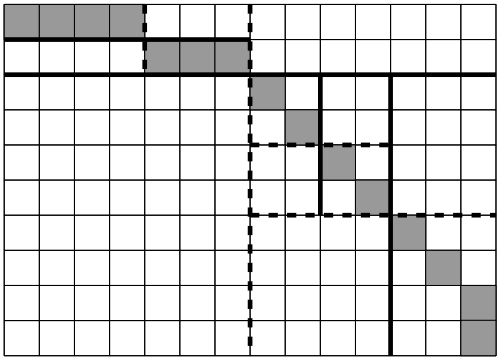
3.3. Blokovski Hirschbergov algoritam s dinamičkom podjelom

Blokovski Hirschbergov algoritam s dinamičkom podjelom optimizacija je načina podjele matrica Hirschbergova algoritma. Navedena optimizacija govori da se, umjesto konstantnog dijeljenja matrice horizontalno, matrica uvijek dijeli po najvećoj dimenziji. Navedeno sprječava protezanje uskih podmatrica kroz iteracije te može smanjiti broj podjela. Na slici 3.9 prikazana je usporedba broja koraka blokovskog Hirschbergova algoritma s dinamičkom podjelom i originalnog blokovskog Hirschbergova algoritma. Parametar e_{max} je jednak 12 te su podjele označene isprekidanim linijama. Primjećujemo da algoritam bez dinamičke podjele radi jednu podjelu više. Također, ovo svojstvo bit će veoma korisno pri korištenju narednih optimizacijskih algoritama. Iako se čini da predložena optimizacija uvelike komplicira implementaciju, uz jednostavnu dosjetku implementacija je veoma jednostavna. Dodatno se koriste ignorirani

parametri g_{f2} i g_{b2} jer se matrica ne dijeli više isključivo horizontalno. Ako je $m < n$, sve što treba napraviti je zamijeniti vrijednosti svih varijabli vezanih za s_1 i s_2 na početku algoritma te nakon pronalaženja svih potrebnih vrijednosti opet zamijeniti. Ako je $n < m$, algoritam je jednak blokovskom Hirschbergovom algoritmu. Uvedimo u terminologiju funkciju DHIRSCHBERG, koja je jednaka funkciji BHIRSCHBERG uz prethodno objašnjene dodatne parametre.



(a) Podjele matrica kod blokovskog Hirschbergova algoritma.



(b) Podjele matrica kod blokovskog Hirschbergova algoritma s dinamičkom podjelom.

Slika 3.9: Usporedba broja koraka blokovskog Hirschbergova algoritma s dinamičkom podjelom i originalnog blokovskog Hirschbergova algoritma.

```
1  DHIRSCHBERG(s_1, s_2, m, n, d, e, sim(), g_f1, g_b1, g_f2, g_b2, e_max):
2  ...
3  if mn ≤ e_max then R = NWG(s_1, s_2, m, n, d, e, sim(), g_f1, g_b1, g_f2, g_b2), return R.p ;
4  if m < n then swap((s_1, s_2), (m, n), (g_f1, g_f2), (g_b1, g_b2)) ;
5  ...
6  H_1u, F_1u = NWH(s_1u, s_2, r, n, d, e, sim(), g_f1, g_f2);
7  H_1d, F_1d = NWH(s_1d, s_2r, m − r − 1, n, d, e, sim(), g_b1, g_b2);
8  ...
9  for i ← 0 to n do
10 |  ...
11 end
12 g_1 = g, g_2 = 0;
13 if n < m then swap((s_1, s_2), (m, n), (g_f1, g_f2), (g_b1, g_b2), (r, c), (g_1, g_2)) ;
14 ...
15 p_0 =DHIRSCHBERG(s_1u, s_2u, r, c, d, e, sim(), g_f1, g_1, g_f2, g_2, e_max);
16 p_1 =DHIRSCHBERG(s_1d, s_2d, m − r − 1, n − c − 1, d, e, sim(), g_1, g_b1, g_2, g_b2, e_max);
17 ...
```

Slika 3.10: Pseudokod blokovskog Hirschbergova algoritma s dinamičkom podjelom gdje su prikazane promjene u odnosu na pseudokod dan slikom 3.7.

3.3.1. Memorijska i vremenska složenost

Važno svojstvo Hirschbergova algoritma s dinamičkom podjelom je smanjenje memorijske potrošnje funkcije NWH. Memorijska složenost ove funkcije je $O(n)$. Ako u svakom koraku osiguramo da se matrica dijeli po većoj dimenziji odgovarajućom zamjenom parametara, tada znamo da vrijedi $m > n$, tj. funkcija NWH će uvijek trošiti minimalno moguće memorije. Uzevši u obzir navedeno i analizu memorijske složenosti originalnog blokovskog Hirschbergova algoritma danu u poglavlju 3.2.1, zaključujemo da je memorijska složenost navedenog algoritma $O(min(m, n) + e_{max})$.

Vremenska složenost navedenog algoritma jednaka je kao i kod algoritma BHIRSCHBERG. Ako promotrimo raspis 3.6, primjećujemo da je kod analize složenosti uzet slučaj podjele matrice točno po polovici u obje dimenzije. Stoga zaključujemo da u analizi vremenske složenosti navedeni algoritam nema utjecaja, tj. njegova vremenska složenost jednaka je originalnom blokovskom Hirschbergovu algoritmu.

3.4. Ukkonenov algoritam

Ukkonenov algoritam, nazvan prema njegovu autoru i prvi puta objavljen u [12], optimizacijski je algoritam dizajniran za provjeravanje je li *edit distance* [13] dva slijeda unutar određenih granica. Navedeni algoritam primjenjiv je na algoritme poravnanja predstavljene u poglavlju 2. Važnost ovog algoritma leži u njegovoj složenosti, koja je korelirana s ocjenom sličnosti ulaznih sljedova, te može značajno ubrzati algoritme poravnanja. U nastavku je objašnjena primjena Ukkonenova algoritma na Needleman-Wunschov algoritam.

Ukkonenov algoritam baziran je na ograničavanju izračunavanja dijagonala matrice. Važna pretpostavka algoritma je da radi s unaprijed poznatom ocjenom poravnanja ili pak njenom gornjom granicom. Znajući ocjenu poravnanja, algoritam može sa sigurnošću odrediti između kojih dijagonala se nalazi put rekonstrukcije. Koristimo terminologiju navedenu u poglavljima 2.3.1 i 2.3.2 te dodatno uvodimo pojmove:

- t - maksimalna razlika dva slijeda definirana kao maksimalna suma različitih elemenata te procjepa u poravnanju dva slijeda.
- u - margina u odnosu na glavnu dijagonalu matrice unutar koje se nalazi optimalno poravnanje.
- Δ - maksimalna ocjena sličnosti dvaju elemenata, tj. maksimalna vrijednost funkcije sim.

Parametar t računamo direktno iz unaprijed poznate ocjene poravnanja kao:

$$t = \max(m, n) - \frac{s}{\Delta} \tag{3.10}$$

Riječima, parametar t je razlika duljine duljeg slijeda i minimalnog broja poravnatih elemenata sljedova koji se dobije iz omjera ocjene poravnanja i maksimalne sličnosti dvaju elemanata. Parametar t se također interpretira kao maksimalna deformacija puta poravnanja. Znajući maksimalnu razliku dvaju sljedova t, logično bi bilo pretpostaviti da se poravnanje nalazi između dijagonala $-\frac{t}{2}$ i $\frac{t}{2}$, uz činjenicu da su negativnim brojevima označene dijagonale matrice ispod glavne dijagonale, dok su pozitivnim brojevima označene dijagonale iznad glavne dijagonale, no navedeno ne vrijedi. Do problema dolazi zbog asimetričnosti matrice, tj. asimetričnost matrice se mora uračunati u marginu rješavanja. Definirajmo u kao:

$$u = \frac{t - |(m, n)|}{2} \tag{3.11}$$

Drugim riječima, asimetričnost matrice se u ovom koraku zanemaruje. Iz istog razloga definiraju se dva nova pojma:

- u_l - margina ispod glavne dijagonale matrice unutar koje se nalazi optimalan put poravnanja.
- u_r - margina iznad glavne dijagonale matrice unutar koje se nalazi optimalan put poravnanja.

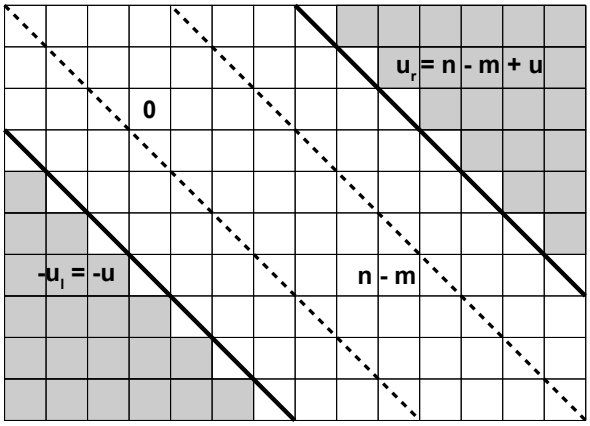
Zbog asimetričnosti matrice, margine u_l i u_r su također asimetrične i definiraju se kao:

$$u_l = \begin{cases} u & m < n \\ u + m - n & \text{inače} \end{cases} \tag{3.12}$$

$$u_r = \begin{cases} u & m > n \\ u + n - m & \text{inače} \end{cases} \tag{3.13}$$

Iz sljedećeg zaključujemo da je potrebno rješavati dijagonale matrice gdje vrijedi $-u \leq j - i \leq u + n - m$, ako vrijedi $m \leq n$ te $n - m - u \leq j - i \leq u$ ako vrijedi $m > n$. Geometrijska interpretacija slučaja kada je $m \leq n$ vidljiva je na slici 3.11. Sivom bojom označeni su elementi matrice koji se zanemaruju. Na slici je prikazan i slučaj izračunavanja margina u_l i u_r gdje su unaprijed poznate vrijednosti $t = 12$, $m = 10$ te $n = 14$. Primjenom izraza 3.11, 3.12 i 3.13 dobivamo vrijednosti $u = 4$, $u_l = 4$

te $u_r = 8$. Sve navedene vrijednosti se također mogu iščitati iz slike. Implementacija algoritma dana je pseudokom na slici 3.12. Uvedimo u terminologiju funkciju NWU. Uz standardne parametre definirane u 2.3.2, funkcija prima i dodatan parametar s, tj. ocjenu ili gornju granicu ocjene poravnanja. Zbog zanemarivanja elemenata svi unutarnji elementi matrica moraju se inicijalizirati na vrijednost $-\inf$, kako ne bi došlo do njihovog utjecaja na rezultat.



Slika 3.11: Područje rješavanja Ukkonenova algoritma.

```
1  NWU(s_1, s_2, m, n, d, e, sim(), s):
2  ...
3  t = max(m, n) − s / max(sim());
4  u_l, u_r = (t − |m − n|) / 2;
5  if n ≥ m then u_r += n − m;
6  else u_l += m − n ;
7  H(1..m, 1..n) = E(1..m, 1..n) = F(1..m, 1..n) = −inf;
8  for i ← 1 to m do
9  |   j_s = max(1, i − u_l), j_e = min(n, i + u_r);
10 |   for j ← j_s to j_e do
11 |   |   ...
12 |   end
13 end
14 ...
```

Slika 3.12: Pseudokod Ukkonenova algoritma u primjeni na Needleman-Wunschov algoritam gdje su prikazane promjene u odnosu na pseudokod dan slikom 2.4.

3.4.1. Memorijska i vremenska složenost

Algoritam dan pseudokodom 3.12 ima memorijsku složenost jednaku kao i originalan Needleman-Wunshov algoritam, tj. $O(mn)$. Navedeni pseudokod predstavljen je zbog

svoje jednostavnosti. Uz pametnije baratanje indeksima matrice te pamćenjem samo bitnih elemenata, navedeni algoritam izvediv je u $O(mt)$ memorije te se ta ocjena računa kao memorijska složenost navedenog algoritma.

Ako analiziramo pseudokod dan slikom 3.12, vidimo da se u svakom retku izračunava t elemenata te zaključujemo da je vremenska složenost algoritma $O(mt)$.

3.5. Algoritam dinamičkog odbacivanja elemenata

Algoritam dinamičkog odbacivanja elemenata optimizacija je Smith-Watermanova algoritma. Algoritam je predstavljen u [14]. Cilj ovog algoritma je smanjiti broj izračuna potrebnih da se dođe do optimalnog rješenja. Koristimo terminologiju navedenu u 2.3.1 i 2.3.3. Bez smanjenja općenitosti, pretpostavimo da se matrica H rješava redak po redak te u retku stupac po stupac. Cilj algoritma je pronaći interval u svakom retku unutar kojeg leži put optimalnog rješenja. Uvedimo pojmove:

- k_s - početak intervala rješavanja u svakom retku.
- k_e - kraj intervala rejšavanja u svakom retku.

Drugim riječima, u retku i rješavamo elemente $H(i,j), j \in [k_s, k_e]$. Odbacivi elementi su tada $H(i,j), j \in [0, k_s - 1] \cup [k_e + 1, n]$. Odbacivi elementi su svi elementi koji u najboljem slučaju ne mogu dostići trenutno najbolju ocjenu poravnanja. Uvedimo pojmove:

- λ - maksimalna vrijednost funkcije sim, tj. $\max(\text{sim}(x,y))$.
- $H_{max}(i,j)$ - maksimalna vrijednost matrice H, tj. maksimalna ocjena poravnanja koju element $H(i,j)$ može postići.
- s' - najveća vrijednost trenutno izračunatih elemenata matrice H.
- λ_i - najveća moguća vrijednost elemenata matrice H u retku i.

Odbacivi elementi su sve elementi za koje vrijedi:

$$H_{max}(i,j) < s' \tag{3.14}$$

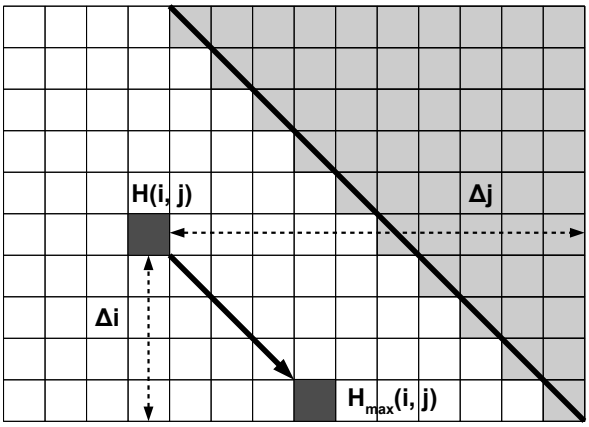
Do vrijednosti $H_{max}(i,j)$ dolazi se pretpostavkom da su sljedovi s_1 i s_2 od pozicija i, j identični, tj. da vrijedi $s_1(i..m) = s_2(j..n)$. Uvedimo dodatne pojmove:

- Δi - udaljenost do kraja slijeda s_1.
- Δj - udaljenost do kraja slijeda s_2.

Možemo pisati:

$$\Delta i = m - i, \Delta j = n - j \tag{3.15}$$

Geometrijska interpretacija navedenih vrijednosti prikazana je na slici 3.13. Na slici su označeni pojmovi $H_{max}, \Delta i$ i Δj. Dodatno, bijelom bojom označeni su Δi elementi, dok su sivom bojom označeni Δj elementi.



Slika 3.13: Geometrijska interpretacija pojmova algoritma dinamičkog odbacivanja elemenata.

Za slučaj najboljeg poravnanja dalje možemo pretpostaviti da je svaki od parova elemenata $s_1(i..m)$ i $s_2(j..n)$ ocijenjen najvećom mogućom ocjenom λ. Uz danu pretpostvaku $H_{max}(i, j)$ raspisujemo:

$$H_{max}(i, j) = \lambda \min(\Delta i, \Delta j) \tag{3.16}$$

Iz iste prepostavke možemo zaključiti:

$$\lambda_i = \lambda i \tag{3.17}$$

Uzevši u obzir odnose vrijednosti i i j, $H_{max}(i, j)$ rastavljamo na slučajeve:

$$H_{max}(i, j) = \begin{cases} \lambda \Delta i & \Delta i < \Delta j \\ \lambda \Delta j & \text{inače} \end{cases} \tag{3.18}$$

Uzevši u obzir formulu 3.18, uvodimo pojmove:

- Δi elementi - elementi matrice H za koje vrijedi $\Delta i < \Delta j$.
- Δj elementi - elementi matrice H za koje vrijedi $\Delta i \geq \Delta j$.

Zbog različitih svojstava Δi i Δj elemenata, algoritam dinamičkog odbacivanja elemenata dijeli se na:

- Δi dinamičko odbacivanje elemenata - algoritam dinamičkog odbacivanja elemenata koji radi na području Δi elemenata.
- Δj dinamičko odbacivanje elemenata - algoritam dinamičkog odbacivanja elemenata koji radi na području Δj elemenata.

3.5.1. Δi dinamičko odbacivanje elemenata

Kako je prethodno navedeno, cilj algoritma dinamičkog odbacivanja elemenata je pronaći interval $[k_s, k_e]$ u svakom retku unutar kojeg leži put optimalnog rješenja. Δi dinamičko odbacivanje elementa pronalazi parametar k_s. Proučimo neka svojstva Δi elemenata. Zbog svojstava Smith-Watermanova algoritma navedenih u 2.3.3 te formule 3.18, možemo zaključiti da ako su elementi $H(i-1,j), j \in [0, k_s - 1]$ odbacivi tada su odbacivi i elementi $H(i,j), j \in [0, k_s - 1]$. Tada zaključujemo da je parametar k_s u retku i je veći ili jednak parametru k_s u retku $i-1$. Da bi Δi element bio odbaciv za i dodatno mora vrijediti $i \geq \frac{m}{2}$. Dokaz:

$$\begin{aligned} H_{max}(i,j) &\leq \lambda_i \\ H(i,j) + \lambda \min(\Delta i, \Delta j) &\leq \lambda i \\ H(i,j) + \lambda \Delta i &\leq \lambda i \\ H(i,j) + \lambda(m-i) &\leq \lambda i \\ H(i,j) + \lambda m &\leq 2\lambda i \\ \frac{H(i,j)}{2\lambda} + \frac{m}{2} &\leq i \\ \frac{m}{2} &\leq i \end{aligned} \tag{3.19}$$

Za vrijednosti $H(i,j)$ i λ vrijedi $H(i,j) \geq 0$ i $\lambda > 0$.

3.5.2. Δj dinamičko odbacivanje elemenata

Kako je prethodno navedeno, cilj algoritma dinamičkog odbacivanja elementa je pronaći interval $[k_s, k_e]$ u svakom retku unutar kojeg leži put optimalnog rješenja. Δj dinamičko odbacivanje elemenata pronalazi parametar k_e. Proučimo neka svojstva Δj elemenata. Zbog svojstava Smith-Watermanova algoritma navedenih u 2.3.3 te formule 3.18, možemo zaključiti da ako su elementi $H(i-1,j), j \in [k_e + 1, n]$ od-

bacivi i element $H(i,x), x \in [k_e+1, n]$ je odbaciv tada su odbacivi i svi elementi $H(i,j), j \in [x+1, n]$, tj. k_e u retku i je manji ili jednak parametru k_e u retku $i-1$. U suprotnom slučaju moramo tražimo prvi odbacivi element $H(i,j), j \in [k_e+2, n]$. U tom slučaju za parametar k_e vrijedi $k_e = j$. Da bi Δj element bio odbaciv za j dodatno mora vrijediti $j \geq n - i$. Dokaz:

$$\begin{aligned} H_{max}(i,j) &\leq \lambda_i \\ H(i,j) + \lambda \min(\Delta i, \Delta j) &\leq \lambda i \\ H(i,j) + \lambda \Delta j &\leq \lambda i \\ H(i,j) + \lambda(n-j) &\leq \lambda i \\ \frac{H(i,j)}{\lambda} + n - j &\leq i \\ n - j &\leq i \\ n - i &\leq j \end{aligned} \tag{3.20}$$

Za vrijednosti $H(i,j)$ i λ vrijedi $H(i,j) \geq 0$ i $\lambda > 0$.

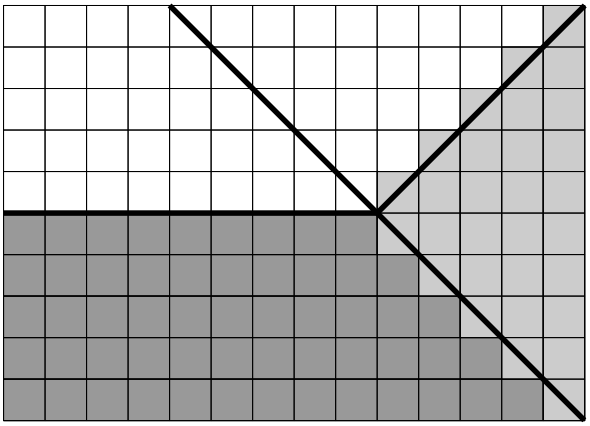
3.5.3. Algoritam dinamičkog odbacivanja elemenata

Algoritam dinamičkog odbacivanja elemenata unija je Δi i Δj dinamičkog odbacivanja elemenata. Uzevši u obzir dodatna svojstva Δi i Δj odbacivanja navedena u 3.5.1 i 3.5.2, računanje elemenata intervala $[k_s, k_e]$ možemo pisati kao:

$$\begin{aligned} k_s &= \begin{cases} \Delta i \text{ odbacivanje} & i \geq \frac{m}{2} \wedge \Delta i < \Delta j \\ 0 & \text{inače} \end{cases} \\ k_e &= \begin{cases} \Delta j \text{ odbacivanje} & j \geq n - i \wedge \Delta i \geq \Delta j \\ n & \text{inače} \end{cases} \end{aligned} \tag{3.21}$$

Područje djelovanja algoritma vidljivo je na slici 3.14. Računanjem navedenog odbacuju se svi elementi $H(i,j), j \in [0, k_s - 1] \cup [k_e + 1, n]$. Smith-Watermanov algoritam uz primjenu algoritma dinamičkog odbacivanja elemenata zadržava svoju optimalnost, kao i deterministčnost. Uvedimo u terminologiju funkciju SWDO. Funkcija prima algoritme jednake funkciji predstavljenoj pseudokodom na slici 2.7 te je njen pseudokod dan slikom 3.15. Zbog odbacivanja elemenata moramo inicijalizirati sve unutarnje elemente matrica na $-\inf$, kako ne bi došlo do njihovog utjecaja na rezultat. Na slici 3.14 prikazano je područje djelovanja algoritma dinamičkog odbacivanja

elemenata. Tamnosivom bojom označeni su elementi Δi odbacivanja za koje vrijedi $i \geq \frac{m}{2} \wedge \Delta i < \Delta j$, dok su svijetlosivom bojom označeni elementi Δj odbacivanja za koje vrijedi $j \geq n - i \wedge \Delta i \geq \Delta j$.



Slika 3.14: Područje djelovanja algoritma dinamičkog odbacivanja elemenata.

```
1  SWDO(s_1, s_2, m, n, d, e, sim()):
2  ...
3  H(1..m, 1..n) = E(1..m, 1..n) = F(1..m, 1..n) = -inf;
4  k_s = 0, k_e = n;
5  for i ← 1 to m do
6  |   j = k_s;
7  |   while i ≥ m/2 and Δi < Δj and H(i,j) + λΔi < s do  k_s = j + 1, j = j + 1 ;
8  |   j = n;
9  |   while Δi ≥ Δj and j ≥ n − i and H(i,j) + λΔj < s do  k_e = j − 1, j = j − 1 ;
10 |   for j ← k_s to k_e do
11 |   |   ...
12 |   end
13 end
14 ...
```

Slika 3.15: Pseudokod algoritma dinamičkog odbacivanja elemenata gdje su prikazane promjene u odnosu na pseudokod dan slikom 2.7.

3.5.4. Memorijska i vremenska složenost

Memorijska složenost algoritma jednaka je kao i memorijska složenost Smith-Watermanova algoritma te iznosi $O(mn)$. Vremensku složenost algoritma možemo pisati kao:

$$t = m(k_e - k_s) \tag{3.22}$$

Vremenska ovisnosti ovisi o razlici $k_e - k_s$. Ta će razlika biti manja čim su sljedovi s_1 i s_2 sličniji. U najgorem slučaju ta će razlika iznositi n tokom cijelog izvršavanja. Zaključujemo da je vremenska složenost $O(mn)$.

3.6. Blokovski Ukkonen-Hirschbergov algoritam s dinamičkom podjelom

Blokovski Ukkonen-Hirschbergov algoritam s dinamičkom podjelom je primjena Ukkonenova algoritma u svim koracima blokovskog Hirschbergova algoritma s dinamičkom podjelom. Navedeni algoritam je predstavljen ovim radom.

3.6.1. Needleman-Wunsch-Ukkonenov algoritam s mogućnosti kontrole rubnih procjepa

Needleman-Wunsch-Ukkonenov algoritam s mogućnosti kontrole rubnih procjepa primjena je Ukkonenova algoritma na Needleman-Wunschov algoritam s mogućnosti kontrole rubnih procjepa. Koristimo terminologiju navedenu u 2.3.1, 2.3.2, 3.1.1 i 3.4. Uvedimo u terminologiju funkciju NWGU. Funkcija prima iste parametre kao i funkcija NWG navedena u 3.1.1 uz dodatni parametar s, unaprijed poznatu ocjenu poravnanja. Algoritam zadržava memorijsku i vremensku složenost Ukkonenova algoritma, tj. obje složenosti iznose $O(mt)$.

```
1  NWGU(s_1, s_2, m, n, d, e, sim(), g_f1, g_b1, g_f2, g_b2, s):
2  ...
3  t = max(m, n) − s/max(sim());
4  u_l, u_r = (t−|m−n|)/2;
5  if n ≥ m then u_r+ = n − m;
6  else u_l+ = m − n ;
7  H(1..m, 1..n) = E(1..m, 1..n) = F(1..m, 1..n) = − inf;
8  for i ← 1 to m do
9  |   j_s = max(1, i − u_l), j_e = min(n, i + u_r);
10 |   for j ← j_s to j_e do
11 |   |   ...
12 |   end
13 end
14 ...
```

Slika 3.16: Pseudokod Needleman-Wunschov-Ukkonenova algoritma s mogućnosti kontrole rubnih procjepa gdje su prikazane promjene u odnosu na pseudokod dan slikom 3.4.

3.6.2. Korak rješavanja Needleman-Wunschova algoritma prilagođen blokovskom Ukkonen-Hirschbergovom algoritmu s dinamičkom podjelom

Korak rješavanja Needleman-Wunschova algoritma prilagođen blokovskom Ukkonen-Hirschbergovom algoritmu s dinamičkom podjelom primjena je Ukkonenova algoritma na Needleman-Wunschov algoritam prilagođen Hirschbergovom algoritmu. Koristimo terminologiju navedenu u 2.3.1, 2.3.2, 3.1.3 i 3.4. Uvedimo u terminologiju funkciju NWHU koja prima iste parametre kao i funkcija NWH navedena u 3.1.1 uz dodatni parametre u_l i u_r, unaprijed izračunate margine Ukkonenova algoritma. Algoritam zadržava vremensku složenost Ukkonenova algoritma koja iznosi $O(mt)$. Primjena Ukkonenova algoritma smanjuje memorijsku složenost algoritma na $O(t)$.

```
1  NWHU(s_1, s_2, m, n, d, e, sim(), g_f1, g_f2, u_l, u_r):
2  ...
3  t = max(m, n) − s/max(sim());
4  u_l, u_r = (t−|m−n|)/2;
5  if n ≥ m then u_r+ = n − m;
6  else u_l+ = m − n ;
7  for i ← 1 to m do
8  |   ...
9  |   if i − 1 > u_l then m_1 = − inf;
10 |   if i > u_l then h_1 = − inf;
11 |   if i + u_r ≤ n then H_1(i + u_r) = − inf, F_1(i + u_r) = − inf;
12 |   j_s = max(1, i − u_l), j_e = min(n, i + u_r);
13 |   for j ← j_s to j_e do
14 |   |   ...
15 |   end
16 end
17 ...
18 H_1(0..n − u_l) = F_1(0..n − u_l) = − inf;
19 return H_1, F_1;
```

Slika 3.17: Korak rješavanja Needleman-Wunschova algoritma prilagođen blokovskom Ukkonen-Hirschbergovom algoritmu s dinamičkom podjelom gdje su prikazane promjene u odnosu na pseudokod dan slikom 3.4.

3.6.3. Blokovski Ukkonen-Hirschbergov algoritam s dinamičkom podjelom

Blokovski Ukkonen-Hirschbergov algoritam s dinamičkom podjelom iskorištava dobre strane svih algoritama čija je kompozicija. Korištenje Hirschbergova algoritma daje al-

goritmu mogućnost izvršavanja u linearnoj memorijskoj složenosti. Korištenje blokovske inačice Hirschbergova algoritma omogućuje kontrolu odnosa potrošnje memorije i vremena. Korištenje dinamičke podjele osigurava da su u svakom koraku dimenzije podmatrica što je više moguće jednake. Ovo svojstvo posebice odgovara Ukkonenovu algoritmu jer uvelike smanjuje asimetričnost matrica te samim time i memorijsku i vremensku složenost primjene Ukkonenova algoritma na navedene podmatrice. Važno svostvo navednog algoritma je njegova paralelizabilnost koja se ostvaraje na dvije razine. Unutar samog koraka algoritma paralelno se mogu izvršavati rješavanje gornje i donje podmatrice. Svaki idući rekurzivni korak može se izvršavati nezavisno.

```
1  UHIRSCHBERG(s_1, s_2, m, n, d, e, sim(), g_f1, g_b1, g_f2, g_b2, e_max, s):
2  ...
3  t = max(m, n) − s / max(sim());
4  if tn ≤ e_max then R = NWGU(s_1, s_2, m, n, d, e, sim(), g_f1, g_b1, g_f2, g_b2, s), return R.p ;
5  ...
6  u, u_r = (t − |m − n|) / 2;
7  u_l = u + m − n;
8  H_1u, F_1u = NWHU(s_1u, s_2, r, n, d, e, sim(), g_f1, g_f2, u_l, u_r);
9  H_1d, F_1d = NWHU(s_1d, s_2r, m − r − 1, n, d, e, sim(), g_b1, g_b2, u_l, u_r);
10 ...
11 s_u, s_d = 0;
12 for i ← 0 to n do
13 |   ...
14 |   if h > s then s_u = H_1u(i), s_d = H_1d(n − i);
15 |   if a > s then s_u = F_1u(i) + d − e, s_d = F_1d(n − i) + d − e;
16 end
17 ...
18 p_0 =UHIRSCHBERG(s_1u, s_2u, r, c, d, e, sim(), g_f1, g_1, g_f2, g_2, e_max, s_u);
19 p_1 =UHIRSCHBERG(s_1d, s_2d, m − r − 1, n − c − 1, d, e, sim(), g_1, g_b1, g_2, g_b2, e_max, s_d);
20 ...
```

Slika 3.18: Pseudokod blokovskog Ukkonen-Hirschbergova algoritma s dinamičkom podjelom gdje su prikazane promjene u odnosu na pseudokod dan slikom 3.10.

Pseudokod algoritma prikazan je na slici 3.18. Uvedimo u terminologiju funkciju UHIRSCHBERG, čiji je pseudokod prikazan slikom 3.18. Funkcija prima jednake parametre kao i funkcija DHIRSCHBERG, čiji je pseudokod prikazan slikom 3.10 uz dodatan parametar s, unaprijed poznatu ocjenu poravnanja. Ako je s nepoznat, ne će se primijeniti Ukkonenova optimizacija isključivo u prvom koraku algoritma. U idućim rekurzivnim podkoracima ocjena poravnanja je poznata te omogućuje korištenje Ukkonenova algoritma.

3.6.4. Memorijska i vremenska složenost

Prilikom analize memorijske složenosti vrijede sve primjedbe navedene u poglavlju 3.2.1. Najveća podmatrica na kojom se može pozvati funkcija NWGU ima broj elemenata jednak e_{max}. Uz memorijsku složenost funkcije NWGU od $O(mt)$, navedene u 3.6.1, gdje za sumu mt vrijedi $mt \leq mn$, dolazimo do zaključka da je složenost funkcije NWGU pri korištenju s blokovskim Ukkonen-Hirschbergovim algoritmom s dinamičkom podjelom jednaka $O(e_{max})$. Funkcija NWHU ima složenost $O(t)$, koja je navedena u 3.6.2 te se ona pri korištenju s navedenim algoritmom ne mijenja. Dolazimo do zaključka da je ukupna memorijska složenost algoritma $O(t + e_{max})$.

Analiza vremenske složenosti algoritma slična je analizi provedenoj u 3.2.1. Razlika je u tome što korak rješavanja podmatrice nema složenost $O(mn)$ već $O(mt)$. Prilikom analize možemo pretpostaviti da se vrijednost parametra t smanjuje u svakom koraku na pola, tj. za parametar t u koraku k vrijedi:

$$t_k = \frac{t}{2^k} \tag{3.23}$$

U raspisu složenosti zbog prethodno navedenog tada možemo zamjeniti parametar n parametrom t. Uvrštavanjem u formulu 3.9 dobivamo:

$$t = 2mt\left(1 - 2^{-\left(1+\left\lceil \log_2 \sqrt{\frac{mt}{e_{max}}} \right\rceil\right)}\right) \tag{3.24}$$

Vremensku složenost algoritma tada možemo ocijeniti kao $O\left(2mt - \sqrt{mte_{max}}\right)$.

4. *Compute Unified Device Architecture*

CUDA (*Compute Unified Device Architecture*) je opće primjenjiva arhitektura paralelizacije razvijena od strane NVIDIA-e [15]. Za razliku od klasičnih sustava, CUDA iskorištava procesore grafičkih kartica. Specifičnost sustava procesora na grafičkim karticama u odnosu na obične procesore predstavlja činjenica da sustav procesora na grafičkoj kartici sadrži velik broj slabijih procesora koji imaju mogućnost paralelnog izvođenja. NVIDIA je otvorila svijet paralelizacije široj javnosti, s obzirom na to da je cijena potrebne opreme realna.

4.1. Arhitektura

Grafička kartica koja podržava tehnologiju CUDA može izvršavati algoritme sa značajnim ubrzanjem u odnosu na vrijeme njihova izvođenja na običnom procesoru. Navedena arhitektura temelji se na velikom broju procesora koji istovremeno mogu izvršavati velik broj dretvi te učinkovito izvršavati aritmetičke operacije nad podacima spremljenim u jedinstveno organiziranoj memoriji [15].

U CUDA-i, dretve su grupirane u blokove, dok su blokovi grupirani u mrežu. Dretve unutar istog bloka mogu razmjenjivati informacije preko brze dijeljene memorije i mogu biti sinkronizirane. Dretve unutar različitih blokova dijele informacije preko spore globalne memorije i rijetko se sinkroniziraju. Sama CUDA nadogradnja je na programski jezik C, što omogućava lako programiranje jezgri [15]. Jezgre su funkcije programa koje se izvršavaju na grafičkoj kartici, tj. standardni program napisan na tehnologiji CUDA kombinira izvršavanje na standardnom procesoru i na procesoru grafičke jedinice.

4.2. Memorija

Organizacija memorije je ključna pri ubrzavanju s tehnologijom CUDA. Arhitektura memorije nešto je složenija, a različite vrste memorija razlikuju se po veličini i brzini

pristupa. Postoji više vrsta memorije, poredanih po brzini pristupa:

- memorija registara - interna memorija svake dretve, nevidljiva svim ostalim dretvama
- memorija konstanti - memorija koja služi samo za čitanje, dostupna svim dretvama
- dijeljena memorija - memorija vidljiva dretvama unutar jednog bloka
- teksture - globalna memorija koja služi samo za čitanje podataka.
- globalna memorija - memorija vidljiva svim dretvama, velika i najsporija

Ključ je uspjeha pri ubrzanju u korištenju što više memorije s vrha popisa, a što manje memorije s dna popisa [15]. Važno je ostajati u ograničenjima veličine određene memorije propisane arhitekturom. Npr. lako se koristi više od predviđeno dostupnih registara za pojedini CUDA blok, no to znači da će se zbog korištenja previše resursa odjednom moći izvršavati manje CUDA blokova.

5. Implementacija algoritama poravnanja sljedova na CUDA arhitekturi

5.1. Algoritmi poravnanja parova sljedova

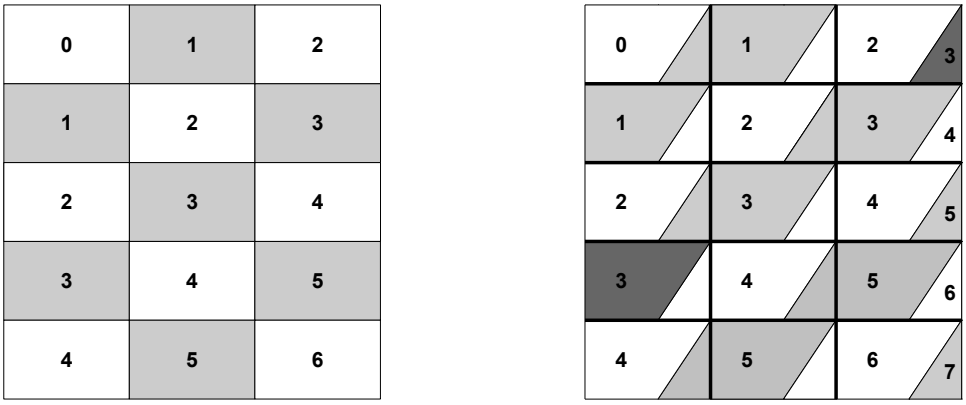
5.1.1. *Wavefront* metoda

Smith-Watermanov i Needleman-Wunschov algoritam te algoritam poluglobalnog poravnanja imaju zajedničko svojstvo neovisnosti elemenata na sporednoj dijagonali matrice rješavanja. Drugim rječima, svi elementi sporedne dijagonale navedenih algoritama mogu se rješavati istovremeno. Istovremeno rješavanje elemenata sporedne dijagonale naziva se *wavefront* metoda. Kako je broj elemenata sporedne dijagonale velik, *wavefront* metoda pogodna je za implementaciju na grafičkim karticama s CUDA arhitekturom.

Kako bi se obavila paralelizacija, potrebno je osmisliti način grupiranja posla. *Wavefront* metoda za CUDA arhitekturu imat će dvije razine paralelizacije. Prva razina paralelizacije je grupiranje elemenata u blokove po sporednoj dijagonali, tj. elementi na sporednoj dijagonali grupiraju se u veće pravokutne blokove rješavanja. Koristimo termonologiju navedenu u 2.3.1. Ako su dimenzije matrice rješavanja $m \times n$, tada želimo grupirati elemente tako da grupe podmatrica budu dimenzija $M \times N$. Nazovimo matricu dimenzija $M \times N$ matrica grupiranja. Svaki element matrice grupiranja je tada podmatrica matrice rješavanja. Za svaku od podmatrica želimo da ima jednak broj elemenata. Problem nastaje zbog nedjeljivosti m sa M te n sa N. Tada sljedove s_1 i s_2 nadopunjavamo neutralnim elementima kako bi navedena djeljivost vrijedila. Neutralni element je takav element za kojeg vrijedi:

$$\text{sim}(x, y) \leq 0 \tag{5.1}$$

Neutralan element ne može utjecati na podizanje ocjene rješenja poravnanja. Tada svaki od elemenata matrice grupiranja sadrži $\frac{mn}{MN}$ elemenata. Zatim grupiramo elemente matrice grupiranja u sporedne dijagonale. Takve dijagonale nazivamo vanjske dijagonale. Označimo vanjske dijagonale s $D_i, i \in [0, M+N-2]$. Pojedini elementi vanjskih dijagonala $D_i, i \in [0, N-2] \cup [M, M+N-2]$ leže van matrice grupiranja. Takvi elementi se preskaču te se za navedene dijagonale kaže da ne postižu potpunu paralelizaciju. Podjela na vanjske dijagonale prikazana je na slici 5.1a. Za parametre M i N vrijedi $M = 5$ i $N = 3$. Dijagonale D_0, D_1, D_5 i D_6 vidljive na lijevoj slici ne postižu potpunu paralelzaciju.



(a) Elementi matrice grupiranja podijeljeni u pravokutnike.

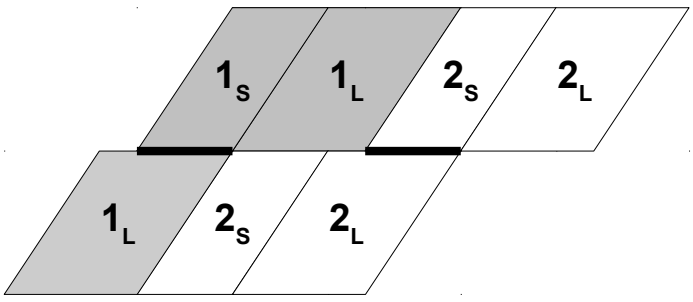
(b) Elementi matrice grupiranja podijeljeni u paralelograme.

Slika 5.1: Primjer matrica grupiranja.

Druga razina paralelizacije je podjela posla unutar elemenata matrice grupiranja. Željene dimenzije podijeljene matrice grupiranja su $M_1 \times N$. Drugim riječima podjela posla unutar elementa matrice grupiranja biti će po retcima. Matrice dobivene navedenom podjelom zovu se unutarnje podmatrice. Ovakvu podjelu možemo shvatiti jednakom kao i kod prve podjele, osim što unutarnje podmatrice sadrže $\frac{M}{M_1}$ elemenata. Unutarnje podmatrice također grupiramo u sporedne dijagonale. Navedene dijagonale zovu se unutarnje dijagonale. Označimo unutarnje dijagonale s $I_i, i \in [0, M_1+N-2]$. Pojedni elementi vanjskih dijagonala $I_i, i \in [0, N-2] \cup [M_1, M_1+N-2]$ leže van matrice grupiranja. Za razliku od elemenata vanjske dijagonale izvan matrice grupiranja, elementi unutarnje matrice izvan elementa matrice grupiranja mogu se iskoristiti tako da većina elemenata unutarnje dijagonale postiže potpunu paralelizaciju. Elementi unutarnje dijagonale koji ne postižu potpunu paralelizaciju, također su elementi elemenata vanjske dijagonale koji ne postižu potpunu paralelizaciju. Navedena promjena je mijenjanje oblika podmatrica matrice grupiranja. Broj elemenata podmatrica ostaje

jednak i iznosi $\frac{mn}{MN}$, no pravokutni oblik mijenja se u oblik paralelograma. Označimo sa $D_{i,j}, j \in [0, N-1]$ elemente vanjske dijagonale D_i. Promjena u paralelogram za dijagonale D za koje vrijedi $D_{i,j}, j \in [1, N-1]$ je trivijalna, no elementi unutarnje dijagonale $D_{i,0}$ rješavaju preostale elemente unutarnje dijagonale $D_{i-1,N-1}$. Navedeni postupak naziva se delegiranje elemenata. Delegiranje elemenata zbog same svoje definicije uzrokuje povećanje broja vanjskih dijagonala za jedan. Na slici 5.1b prikazane su navedene promjene. Za parametre M i N vrijedi $M = 5$ i $N = 3$. Primjećujemo da jedino dijagonale D_3 i D_4 postižu punu paralelizaciju, no značajno više elemenata unutarnje dijagonale postiže punu paralelizaciju. Tamnosivom bojom prikazano je delegiranje elemenata.

Navedena podjela posla je pogodna implementaciji na CUDA arhitekturi. Vanjske dijagonale rješavat će CUDA blokovi dok će unutarnje dijagonale rješavati CUDA dretve. Označimo s B broj CUDA blokova. Cilj je rješavati jednu cijelu vanjsku dijagonalu odjednom. Za vrijednost N tada vrijedi $B = \frac{n}{N}$, tj. $N = \frac{n}{B}$. Indeksi blokova jednaki su indeksima vanjske dijagonale, tj. blok B_i rješava element vanjske dijagonale D_i. Podjela unutarnjih dijagonala napravljena je na način da svaka dijagonala rješava $\frac{M}{M_1}$ redaka. Empirijskim isprobavnjem, kojem je podloga CUDA zasićenost i korištenje ugrađenih četveroelementnih vektorskih tipova, pokazano je da se najbrža podjela unutarnjih dijagonala događa kada svaka dretva rješava četiri retka, tj. vrijedi $\frac{M}{M_1} = 4$. Označimo broj CUDA dretvi s T. Označimo s (i_0, j_0) lijevi gornji kut paralelgorama elementa vanjske dijagonale. Dretva T_i tada rješava elemente $(i_0 + i, x - i), x \in [0, N-1]$. Za vrijednost M vrijedi $M = 4T$. Broj vanjskih dijagonala tada je jednak $|D| = \frac{m}{4T} + B$.



Slika 5.2: Problem sinkronizacije CUDA blokova *wavefront* metode.

Prilikom rješavanja vanjskih dijagonala javlja se CUDA sinkronizacijski problem. Zbog ovisnosti elemenata vanjske dijagonale, potrebna je njihova međusobna sinkronizacija. Svaka aplikacija implementirana na CUDA arhitekturi mora osigurati neo-

visnost izvođenja blokova. Na slici 5.2 širokom crnom linijom označeni su elementi međuovisnosti između CUDA blokova. Kako bi se međuovisnost izbjegla, svaka vanjska dijagonala rješava se u dva koraka, dugom i kratkom koraku. Rješavanje vanjske dijagonale u dvije faze također se preslikava i na rješavanje unutarnjih dijagonala svakog elementa vanjske dijagonale. U kratkom koraku svaka od dretvi T rješava elemente $(i_0 + i, x - i), x \in [0, T - 1]$, dok u dugom koraku rješava elemente $(i_0 + i, x - i), x \in [T, N - 1]$. Na slici 5.2 vidljiva je podjela elemenata vanjske dijagonale na kratki i dugi korak.

$H_{(i-1,\,j-1)}$	$H_{(i-1,\,j)}$ $F_{(i-1,\,j)}$
$H_{(i,\,j-1)}$ $E_{(i,\,j-1)}$	$H_{(i,\,j)}$ $E_{(i,\,j)}$
$H_{(i+1,\,j-1)}$ $E_{(i+1,\,j-1)}$	$H_{(i+1,\,j)}$ $E_{(i+1,\,j)}$
$H_{(i+2,\,j-1)}$ $E_{(i+2,\,j-1)}$	$H_{(i+2,\,j)}$ $E_{(i+2,\,j)}$
$H_{(i+3,\,j-1)}$ $E_{(i+3,\,j-1)}$	$H_{(i+3,\,j)}$ $E_{(i+3,\,j)}$

(a) Odgovarajući elementi K-susjedstva u matrici rješavanja algoritama poravnanja sljedova.

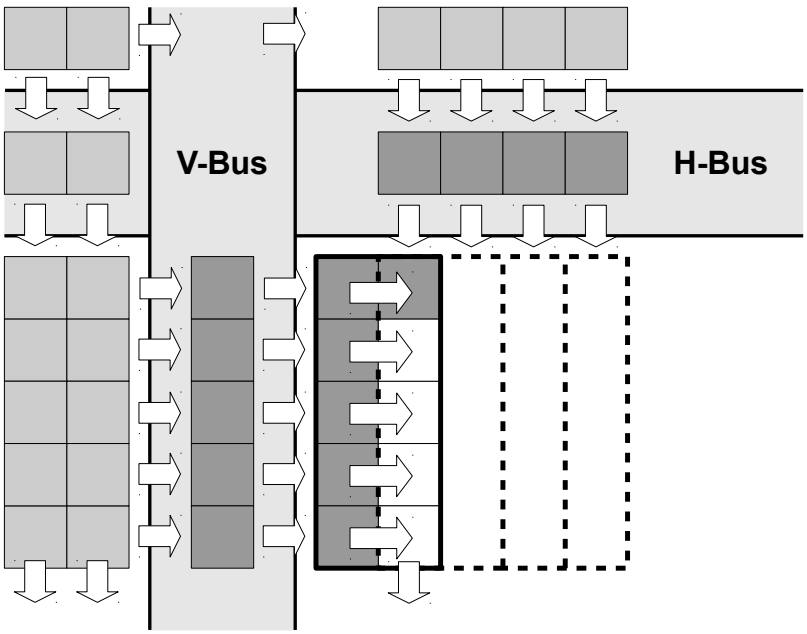
K_d	K_{up}
$K_l.x$	$K_r.x$
$K_l.y$	$K_r.y$
$K_l.z$	$K_r.z$
$K_l.w$	$K_r.w$

(b) Nazivi elemenata K susjedstva korišteni pri implementaciji *wavefont* metode na CUDA arhitekturi.

Slika 5.3: K-susjedstvo *wavefont* metode.

Kao što je prethodno spomenuto, svaka dretva rješava odjednom četiri retka. Definirajmo elemente potrebne za rješevanje četiri retka kao K-susjedstvo. Svako K-susjedstvo sastoji se od deset elemenata. Drugim riječima to je podmatrica dimenzija 5×2. Elementi koji se rješava sadržani su u vektoru K_r, elementi lijevo od vektora K_r nazivaju se K_l, dijagonalni element je K_d, dok je gornji element nazvan K_{up}. Ovi elementi će poslužiti za razmjenu podataka među K-susjedstvima. Potrebno je osmisliti sustav razmjene elemenata među K susjedstvima. Za navedeno će poslužit dva sustava, horizontalna i vertikalna sabirnica, ili kraće H_{bus} i V_{bus}. Na slici 5.3 vidljiva je geometrijska interpretacija K-susjedstva te usporedba sa formalno definiranim elementima matrice rješavanja. Svjetlosivom bojom označeni su elementi vertikalne sabirnice, dok je tamnijom sivom označen element horizontalne sabirnice.

Iako nazvana vertikalna sabirnica, navedena sabirnica prenosi elemente s lijeva na desno. Ako promotrimo sliku 5.1b, možemo uvidjeti da je u jednom trenutku potrebno prenositi elemente slijeva nadesno samo unutar jedne eksterne dijagonale. Iz navedenog zaključujemo da je potrebna veličina vertikalne sabirnice $4BT$. Kako je svaka dretva zadužena za četiri retka, zaključujemo da je elemente preko same vertikalne sabirnice potrebno prenositi samo između koraka rješavanja vanjskih dijagonala. Drugim riječima, vertikalna sabirnica prenosi elemente K_l i K_d. Pretpostavimo da je element K_{up} dostupan u svakom koraku. Na početku se učitaju elementi K_l i K_d iz vertikalne sabirnice te se izračunaju elementi K_r. Sljedeći korak izračuna postiže se pomicanjem elemenata K_r u elemente K_l te pomicanjem elemenata H matrice elementa K_{up} u element K_d te se ponovno izračunaju elementi K_r. Na kraju kratkog i dugog koraka elementi K_r te element matrice H elementa K_{up} spremaju se u vertikalnu sabirnicu.



Slika 5.4: Sabirnice *wavefront* metode.

Horizontalna sabirnica prenosi elemente između svaka dva retka elemenata matrice. Za razliku od vertikalne sabirnice, horizontalna sabirnica djeluje nad cijelim područjem matrice te je njezina veličina n. Spremanje i učitavanje iz horizontalne sabirnice obavlja se pri svakom rješavanju K-susjedstva te je upravo zbog izmjene elemenata horizontalne sabirnice bila potrebna podjela na dugi i kratki korak rješavanja vanjske dijagonale. Horizontalna sabirnica u sebi sadrži elemente K_{up} te se oni učitavaju na početku rješavanja svakog K-susjedstva. Prilikom svakog rješavanja K-

susjedstva poznata je vrijednost F matrice elemenata K_r. Navedena vrijednost matrice F, koja odgovara elementu $K_r.w$, te element H matrice elementa $K_r.w$ na kraju svakog rješavanja K-susjedstva spremaju se u horizontalnu sabirnicu.

Na slici 5.4 vidljiv je sustav sabirnica *wavefront* metode. Na slici je vidljiva razmjena i protok elemenata prilikom rješavanja algoritma. Svjetlosivom bojom označeni su riješeni elementi. Tamnosivom bojom označeni su elementi koji se prenose sabirnicama te je protok dodatno označen strelicama. Bijelom bojom označeni su neriješeni elementi. Isprekidanim pravokutnicama označen je protok K-susjedstva.

Efikasnost implementacije *wavefront* metode na CUDA arhitekturi uvelike ovisi o memorijskom dizajnu algoritma. Ulazni sljedovi s_1 i s_2 učitani su u globalnu memoriju te se čitaju preko tekstura. Vertikalna sabirnica u potpunosti je pohranjena u globalnoj memoriji. Horizontalnu sabirnicu memorijski je moguće podjeliti u dva dijela, globalnu sabirnicu i sabirnicu pohranjenu u dijeljenoj memoriji bloka. Zbog samog dizajna algoritma dretva T_0 će čitati vrijednosti horizontalne sabirnice iz globalne memorije, dok će dretva T_{T-1} zapisivati vrijednosti u horizontalnu sabirnicu smještenu u globalnoj memoriji. Sva ostala čitanja i pisanja horizontalne sabirnice odvijat će se preko dijeljene memorije bloka. Čitanje horizontalne sabirnice iz globalne memorije odvija se preko teksture. Vrijednosti K-susjedstva pohranjene su u registrima. Svi ostali potrebni podaci smješteni su u konstantnoj memoriji. Ovakav dizajn pogoduje memorijskoj složenosti CUDA algoritma jer jedino ograničenje predstavlja potrošnja globalne memorije. Potrošnja globalne memorije horizontalne sabirnice je po element matrica H i F za svaki njezin element. Iz razloga što je dovoljno prikazivati vrijednosti matrice pomoću podataka veličine četiri bajta, memorijska potrošnja horizontalne sabirnice iznosi $8n$ bajtova. Za svaki redak vertikalne sabirnice potrebno je spremiti element H matrice elementa K_{up} te elemente H i E vektora K_r, tj. ukupno devet vrijednosti. Memorijska potrošnja vertikalne sabirnice tada iznosi $36BT$. Ako uračunamo i ulazne sljedove, ukupna potrošnja globalne memorije CUDA algoritma iznosi $36BT + 8n + n + m$. Ako se zanemari konstantna potrošnja memorije, zaključujemo da varijabilna potrošnja CUDA algoritma iznosi $9n + m$.

Pri procjeni teoretskog ubrzanja CUDA implementacije *wavefront* algoritma, pretpostavit ćemo da sve vanjske dijagonale postižu punu paralelizaciju. U navedenom slučaju istovremeno se izračunava $4BT$ elemenata matrice. Stvarno ubrzanje algoritma mnogo je manje od navedenog zbog samog dizajna CUDA arhitekture te značajno ovisi o grafičkom procesoru na kojem se izvodi.

```
1  CUDAWFI(D_k, d, t):
2  r, c = coo (D_k, t, blockIdx.x, threadIdx.x);
3  if c < 0 then
4  |  delegate(r, c);
5  end
6  K_l, K_d = V_bus(r, c);
7  s'_1 = s_1(r..r + 3);
8  for i ← 0 to d do
9  |  if r ≥ 0 and r < m then
10 |  |  K_u = H_bus(c, threadIdx.x);
11 |  |  solve(K, s'_1, s_2(c));
12 |  |  H_bus(c, threadIdx.x) = K_r.w, F;
13 |  |  K_l = K_r;
14 |  |  K_d = K_up;
15 |  end
16 |  c = c + 1;
17 |  if c = n then
18 |  |  undelegate(r, c);
19 |  |  init(K, r);
20 |  |  s'_1 = s_1(r..r + 3);
21 |  end
22 end
23 V_bus(r, c) = K_r, K_up;
```

(a) Pseudokod jezgre CUDA implementacije *wavefront* metode.

```
1  CUDAWFE:
2  for i ← 0 to M + B − 1 do
3  |  CUDAWFI«<B, T»>(i, T, short);
4  |  CUDAWFI«<B, T»>(i, N − T, long);
5  end
```

(b) Pseudokod vanjskog pozivanja jezgre CUDA implementacije *wavefront* metode.

Slika 5.5: Pseudokod CUDA implementacije *wavefront* metode.

Na slici 5.5 dan je pseudokod opisane implementacije *wavefront* metode na CUDA arhitekturi. Uvedimo u terminologiju sljedeće funkcije:

- *coo* - funkcija vraća koordinate unutarnje dijagonale koju dretva rješava. Ova funkcija ovisi o indeksu bloka, dretve, dijagonale te tipu koraka koji se rješava.
- *delegate* - funkcija prepravlja redak i stupac dretve, tj. obavlja delegiranje elemenata. Funkcija se aktivira samo u blokovima B_0 kratkog koraka.
- *undelegate* - funkcija je inverz funkcije *delagate*.
- *init* - funkcija inicijalizira K-susjedstvo na početne uvjete korištenog algoritma poravnanja.
- V_{bus} - funkcija dohvaća ili sprema podatke vertikalne sabirnice. Kako je vertikalna sabirnica duljine visine jedne vanjske dijagonale, tj. indeksi vertikalne sabirnice dinamički mijenjaju svoj pripadajući redak, vertikalna sabirnica se dinamički mora inicijalizirati. Parametar r, indeks retka, koristi se za određivanje odgovarajućeg retka vertikalne sabirnice. Parametar c se koristi za prije

navedeno dinamičko inicijaliziranje. Ako je $c = 0$, tada se umjesto elemenata vertikalne sabirnice vraćaju izračunati rubni uvjeti algoritma poravnanja.

– H_{bus} - funkcija dohvaća ili sprema podatke horizontalne sabirnice. Za razliku od vertiklane sabirnice elementi horizontalne sabirnice mogu se inicijalizarati prije pokretanja prve vanjske dijagonale. Uz očekivani parametar c, indeks stupca koji odgovara indeksu u horizontalnoj sabirnici, funkcija prima parametar *threadIdx.x*, indeks dretve. Ako je indeks dretve jednak nuli, tada funkcija preko teksture čita horizontalnu sabirnicu iz globalne memorije. Ako je indeks dretve jednak $T - 1$ tada se sadržaj horizontalne sabirnice zapisuje u globalnu memoriju. U svim ostalim slučajevima funkcija radi s dijeljenom memorijom.

– *solve* - funkcija izvršava algoritam poravnanja nad jednim K-susjedstvom. Ovisno o algoritmu poravnanja, navedena funkcija također se brine o pamćenju najbolje ocjene poravnanja te njene pozicije.

5.1.2. Metoda dopunjavanja elemenata *wavefront* metode s kontrolom rubnih procjepa prilagođena ekstrakciji zadnjih redaka matrica H i F

Pri korištenju CUDA *wavefront* implementacije algoritama poravnanja često je korisno dohvaćanje zadnjeg retka matrica H i F te kontrola rubnih procjepa. Navedeno je potrebno kako bi se implementirale CUDA verzije algoritama navedene u poglavlju 3.

Logično je ekstrakciju zadnjih redaka matrice H i F obaviti preko horizontalne sabirnice. Štoviše, problem je trivijalan ako je m djeljivo sa M jer horizontalna sabirnica sadrži upravo tražene podatke. Ako navedeno ne vrijedi, problem je zahtjevniji. Problem nastaje u dopunjavanju elemenata, konkretno elemenata slijeda s_1. Navedeni problem prikazan je na slici 5.6. Ako s_1 nadopunimo elementima sa stražnje strane, tada u horizontalnoj sabirnici više neće biti sadržan zadnji redak matrice H i F, kao što je vidljivo na slici 5.6a. Stoga se elementi slijeda s_1 dopunjavaju neutralnim elementima sa prednje strane, kao što je vidljivo na slici 5.6b.

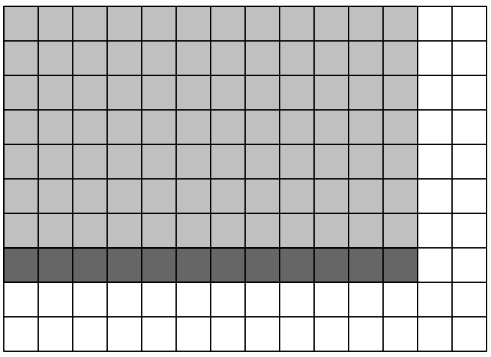
Navedeni postupak stvara problem inicijalizacije rubnih uvjeta. Rubni uvjeti se inicijaliziraju na način da ispravljaju činjenicu da je matrica rješavanja dopunjena s prednje strane. Navedeno utječe na inicijalizaciju horizontalne i vertikalne sabirnice.

Izmijenjena inicijalizacija početnih uvjeta matrice dana je izrazima:

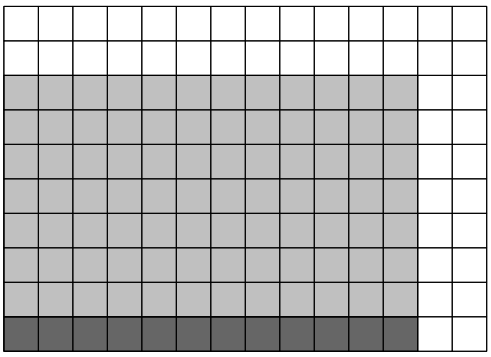
$$H'_{i,0} = \begin{cases} -\inf & i < \Delta m \\ H_{i-\Delta m,0} + (d-e) & g_{f1} \\ H_{i-\Delta m,0} & \text{inače} \end{cases} \tag{5.2}$$

$$E'_{i,0} = \begin{cases} 0 & g_{f1} \wedge i = \Delta m \wedge \Delta m > 0 \\ -\inf & \text{inače} \end{cases} \tag{5.3}$$

$$H'_{0,j} = \begin{cases} H_{0,j} & \Delta m = 0 \\ H_{0,j} + (d-e) & \Delta m = 0 \wedge g_{f2} \\ -\inf & \text{inače} \end{cases} \tag{5.4}$$



(a) Dopunjavanje elemenata slijeda s_1 sa stražnje strane.



(b) Dopunjavanje elemenata slijeda s_1 s prednje strane.

Slika 5.6: Usporedba dopunjavanja elemenata slijeda s_1 sa prednje i stražnje strane.

Inicijalizacijom početnih uvjeta na navedeni način osigurava se nepromijenjenost izvođenja algoritma usprkos nadopunjavanju slijeda s_1 neutralnim elementima s prednje strane. Navedeni postupak i njegova ispravnost pokazani su slikom 5.7. Parametri postupka su $d = 5$ i $e = 2$, funkcija $\text{sim}(x, y) = 1$ ako $x = y$, -3 inače, te $g_{f1} = 1$ i $g_{f0} = 0$. Kao neutralni element odabran je X. Primjećujemo jednakost matrica rješavanja u području neneutralnih elemenata.

				A		
	0	-∞	-∞	-2	-∞	-∞
A	-5	-∞	-∞	-1	-7	-10
C	-7	-∞	-∞	-4	-4	-12
G	-9	-∞	-∞	-6	-6	-14
X	-11	-∞	-∞	-8	-8	-16

(a) Algoritam poravnanja izveden u slučaju nadopunjavanja elemenata slijeda s_1 sa stražnje strane te bez ispravljanja početnih uvjeta.

				A		
	-∞	-∞	-∞	-∞	-∞	-∞
X	0	-∞	0	-2	-∞	-2
A	-5	-∞	-∞	1	-7	-10
C	-7	-∞	-∞	-4	-4	-12
G	-9	-∞	-∞	-6	-6	-14

(b) Algoritam poravnanja izveden u slučaju nadopunjavanja elemenata slijeda s_1 s prednje strane te s ispravljanjem početnih uvjeta.

Slika 5.7: Metoda dopunjavanja elemenata *wavefront* metode s kontrolom rubnih procjepa prilagođena ekstrakciji zadnjih redaka matrica H i F.

5.1.3. Algoritam dinamičkog odbacivanja elemenata prilagođen *wavefront* metodi

Algoritam dinamičkog odbacivanja elemenata objašnjen je u poglavlju 3.5 te se koristi terminologija definirana istim poglavljem. Primjena algoritma dinamičkog odbacivanja elemenata u njegovoj izvornoj formi na CUDA implementaciju *wavefront* metode je neodgovarajuća zato što bi za svaki element posebno trebalo provjeravati je li unutar intervala $[k_s, k_e]$. Navedeno je loše jer, osim dodatnih provjera, može izazvati divergenciju toka izvođenja među dretvama, što je na CUDA arhitekturi iznimno loše. Kako bi se navedeno izbjeglo, umjesto određivanja intervala rješavanja nad elementima, interval rješavanja definirat će se nad CUDA blokovima.

Definirajmo interval $[b_s, b_e]$, slično intervalu $[k_s, k_e]$, kao interval rješavanja blokova. Interval $[b_s, b_e]$ određuje koji će se blokovi vanjske dijagonale izvoditi u potpunosti, a koji će odmah prekinuti s izvođenjem. Kako bi ostvarili navedenu funkcionalnost, na kraju svakog dugog koraka svaki od blokova zapisuje u globalnu memoriju maksimalnu ocjenu poravnanja nađenu u tom bloku. Algoritam određivanja intervala $[b_s, b_e]$ odvija se u potpunosti na standardnom procesoru.

Svaki od blokova B može prenijeti najbolju ocjenu iz dva bloka prethodne vanjske dijagonale, bloka iznad sebe B_u i bloka lijevo od sebe B_l. Kako je pozicija najbolje ocjene bloka nepoznata pri određivanju vrijednosti Δi i Δj, koriste se maksimalne

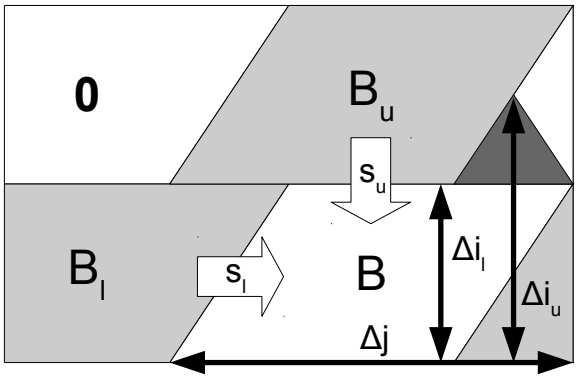
moguće vrijednosti. Budući da su blokovi oblika paralelograma, dijelimo određivanje vrijednosti Δi i Δj na dva dijela, jedan koji uzima ocjenu bloka B_l i drugi koji uzima ocjenu bloka B_u. Pretpostavimo da je gornja lijeva koordinata paralelograma koji predstavlja blok jednaka (i_0, j_0). Za vrijednost Δj tada vrijedi:

$$\Delta j = n' - \max(0, j_0 - T) \tag{5.5}$$

Pri računanju vrijednosti H_{max} uzimaju se različite vrijednosti parametra Δi za blokove B_l i B_u. Razlog tome leži u obliku blokova. Ako se maksimalna ocjena nalazi na donjoj polovici desnog ruba paralelograma bloka B_u, optimalan put može proći kroz gornji desni šiljak promatranog bloka te se on tada ne smije odbacti. Navedeno problematično područje označeno je tamnosivom bojom na slici 5.8. Umjesto parametra Δi uvodimo parametre Δi_l i Δi_u za koje vrijedi:

$$\Delta i_l = m' - i_0, \Delta i_u = \Delta i_l + 4\frac{T}{2} \tag{5.6}$$

Geometrijsko objašnjenje navedenih vrijednosti dano je slikom 5.8. Vidljivo je da se parametri Δi i Δj maksimiziraju za promatrani blok. Tamnosivom bojom označeno je problematično područje kroz koje može prolaziti optimalno poravnanje zbog čega se ono mora uračunati.



Slika 5.8: Geometrijska interpretacija parametara algoritma dinamičkog odbacivanja elemenata prilagođenog *wavefront* metodi.

Za vrijednost H_{max} svakog bloka tada vrijedi:

$$H_{max} = \begin{cases} \max(s_l, s_u) + \lambda\Delta j & \Delta j \text{ odbacivanje} \\ \max(s_l + \lambda\Delta i_l, s_u + \lambda\Delta i_u) & \Delta i \text{ odbacivanje} \end{cases} \tag{5.7}$$

Iz navedenih izraza izračunava se interval $[b_s, b_e]$ nakon izračunavanja svake vanjske dijagonale. Vrijednosti vertikalne i horizontalne sabirnice koje odgovaraju odbačenim blokovima postavljaju se na $-\inf$ kako ne bi utjecale na optimalnost rješenja.

5.1.4. Ukkonenov algoritam prilagođen *wavefront* metodi

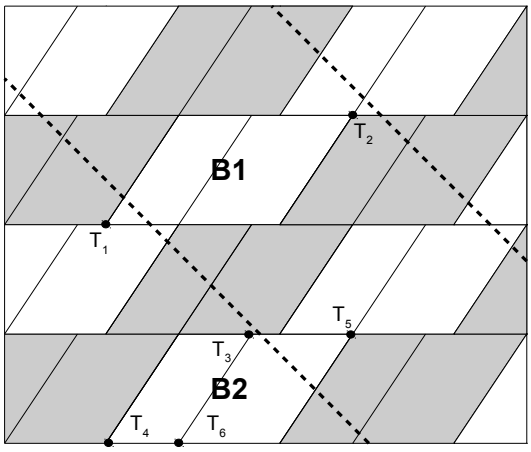
Ukkonenov algoritam objašnjen je u poglavlju 3.4 te se koristi terminologija definirana istim poglavljem. Primjena Ukkonenova algoritma u njegovoj izvornoj formi na CUDA implementaciju *wavefront* metode je neodgovarajuća iz razloga što bi za svaki element posebno trebalo provjeravati je li unutar intervala $[u_l, u_r]$. Navedeno je loše jer, osim dodatnih provjera, može izazvati divergenciju toka izvođenja među dretvama, što je na CUDA arhitekturi iznimno loše. Kako bi se to izbjeglo, umjesto odbacivanja elemenata, odbacivanje će se vršiti nad CUDA blokovima.

Pretpostavimo da je gornja lijeva koordinata paralelograma koji predstavlja blok jednaka (i_0, j_0). Provjera potrebe računanja blokova vrši se na početku izvođenja oba koraka jedne vanjske dijagonale neovisno. Dio bloka koji pripada kratkom koraku može se odbaciti neovisno o dijelu bloka koji pripada dugom koraku. Da bi se kratki dio bloka mogao odbaciti mora vrijediti:

$$(i_0 + T) - j_0 > u_l \vee (j_0 + 4T) - (i_0 - T) > u_r \tag{5.8}$$

Da bi se dugi dio bloka mogao odbaciti mora vrijediti:

$$(i_0 + N) - j_0 > u_l \vee (j_0 + 4T) - i_0 > u_r \tag{5.9}$$



Slika 5.9: Primjer Ukkonenova algoritma prilagođenog *wavefront* metodi.

Primjer odbacivanja dan je slikom 5.9. Provjerom točaka T_1 i T_2 zaključujemo da se ni dugi ni kratki dio bloka B_1 ne može odbaciti. Provjerom točaka T_3 i T_4 zaključujemo da se odbacuje kratki dio bloka B_2. Provjerom točaka T_5 i T_6 zaključujemo da se ne odbacuje dugi dio bloka B_2. Vrijednosti vertikalne i horizontalne sabirnice koje odgovaraju odbačenim dijelovima blokova postavljaju se na $-\inf$ kako ne bi utjecale na optimalnost rješenja.

5.2. Algoritmi poravnanja parova skupova sljedova

Dosad se u radu govorilo o poravnanju dva slijeda. Čest zahtjev za algoritme poravnanja sljedova je poravnanje jednog ili više sljedova sa bazom sljedova. Nazovimo jedan ili više sljedova koji se poravnavaju s bazom sljedova subjektni sljedovi. Baze sljedova često su vrlo velike. Iz tog razloga se ne rade poravnanja subjektnih sljedova s cijelom bazom već se samo ocjenjuje poravnanje. U sljedećem koraku se odabiru parovi koji se tada poravnavaju. Ovo poglavlje bavit će se CUDA implementacijom ocjenjivanja poravnanja subjektnih sljedova s bazom sljedova. Paralelizacija je usmjerena na poravnanje jednog subjektnog slijeda s bazom. Ako postoji više subjektnih sljedova tada se uspoređuju s bazom jedan po jedan. Zbog empirijski pokazanog različitog ponašanja poravnanja subjektnog slijeda s dugim i kratkim sljedovima u bazi, predstavljaju se dva algoritma: algoritam poravnanja kratkih sljedova te algoritam poravnanja dugih sljedova. Prije objašnjenja navedenih algoritama valja objasniti optimizaciju profila subjektnog slijeda.

5.2.1. Profil subjektnog slijeda

U pravilu, pri poravnavanju skupova sljedova, subjektni sljedovi te sljedovi iz baze su kratki, tj. rijetko premašuju duljinu od 50000 elemenata. Ulazni skup elemenata u sljedovima je veoma malen (najčešće engleska abeceda). Navedene činjenice se mogu iskoristiti kako bi se značajno smanjilo potrebno računanje u algoritmima poravnanja. Pretpostavimo da je maksimalan broj različitih elemenata ulaznih sljedova 26 (slova engleske abecede). Ako je duljina subjektnog slijeda 100000, tablica veličine 100000 $\times$ 26 troši manje od 4MB memorije. Navedeno svojstvo iskoristit će se kako bi za cjeli subjektni slijed napravili tablicu koja sadrži ocjene sa svakim mogućim ulaznim znakom. Takva tablica zove se profil subjektnog slijeda.

Vrijednosti koje vraća funkcija sim su male. Stoga možemo iskoristiti CUDA tip podatka *char4* kako bi se odjednom pročitale četiri vrijednosti iz tablice. Duljina su-

bjektnog niza m tada mora biti cjelobrojno djeljiva s četiri. Ako to nije slučaj, nadopunjavamo subjektni niz neutralnim elementima do duljine m' koja je cjelobrojno djeljiva s četiri. Označimo s |sim| broj raličitih elemenata subjektnih sljedova i sljedova u bazi. Tablica profila tada je dimenzija $\frac{m'}{4} \times |\text{sim}|$. S navedenom tablicom jednim čitanjem iz globalne memorije dobivamo ocjene sličnosti četiri elementa, što je četiri puta manje od čitanja svake ocjene zasebno iz tablice sličnosti. Zbog svoje veličine, tablica profila spremljena je u globalnoj memoriji. Kako je tablica profila konstantna, učitava se preko dvodimenzionalne teksture. Nazovimo navedenu teksturu QP. Ocjene iz tablice profila tada se dohvaćaju:

$$QP(r, x), \tag{5.10}$$

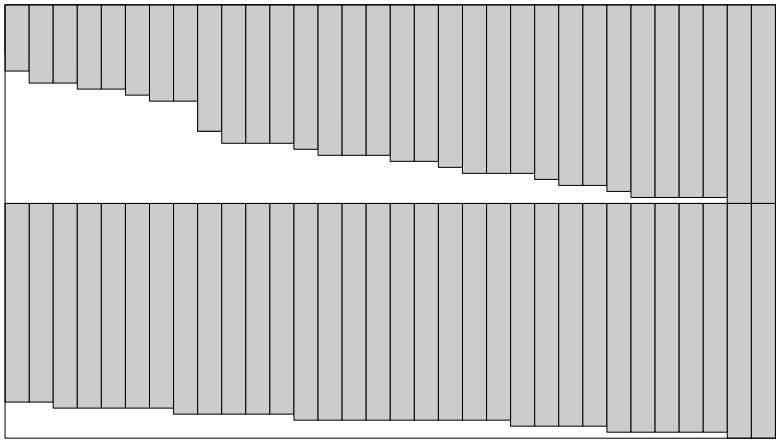
gdje je r indeks retka dok je x element trenutnog slijeda iz baze. Ovom naredbom dohvaćaju se ocjene sličnosti elemenata retka s indeksima od $4r$ do $4r+3$ s elementom x slijeda iz baze.

5.2.2. Ocjenjivanje poravnanja baze kratkih sljedova

Cilj ocjenjivanja poravnanja kratkih sljedova je brzo poravnati jedan subjektni slijed s bazom kratkih sljedova. Definirajmo kratki slijed kao slijed čija duljina ne premašuje vrijednosti n_{max}. Označimo bazu sljedova s DB te broj sljedova u bazi db. Cilj algoritma je izračunati ocjenu poravnanja subjektnog slijeda sa svim sljedovima u bazi sljedova DB. Budući da su nizovi kratki, poravnanje subjektnog slijeda sa sljedovima iz baze nije učinkovito rješavati s više od jedne CUDA dretve, tj. za jedno poravnanje bit će zadužena jedna CUDA dretva.

Kako bi CUDA dretve jednog bloka izvršavale što sličniji posao, čime se izbjegava CUDA serijalizacija koda i nepotrebno trošenje vremena, sljedovi baze se sortiraju. Zbog efikasnosti memorijskog pristupa, sljedovi se u memoriji ne spremaju slijedno već stupčasto, tj. slijedno su u memoriji spremljeni svi elementi sljedova s indeksom nula, pa sa indeksom jedan, itd.. Navedenim se postiže slijedno čitanje CUDA dretvi iz memorije, tj. CUDA može serijalizirati pozive čitanja čime se štedi vrijeme. Kako bi se iskoristila navedena svojstva,s baza sljedova se u memoriju sprema u posebnom matričnom obliku. Označimo sortirane sljedove baze DB s $DB_i, i \in [0, db-1]$ te njihove duljine s db_i gdje je db_i duljina slijeda DB_i. Podjela dimenzija matrice izvršava se na dvije razine. Izgradimo matricu dimenzija $DB_r \times 1$, gdje je $DB_r = \left\lceil \frac{db}{BT} \right\rceil$. U svaki redak takve matrice spremamo BT sljedova baze na način da se u redak i spremaju nizovi baze $DB_j, j \in [iBT, \min((i+1)BT, db)-1]$. Zadnji redak navedene matrice može sadržavati manje od BT sljedova. Označimo elemente matrice s $DBR_i, i \in [0, DB_r-1]$.

Svaki od elemata DBR_i je matrica dimenzija $\frac{db_{\min((i+1)BT,db)-1}}{4} \times BT$. Visina elementa matrice DBR_i određena je zadnjim, tj. najdužim elementom kojeg sadrži. Faktor dijeljenja s četiri pojavljuje se jer su sljedovi baze zapisani pomoću CUDA tipa podataka *char4*. Ako slijed DB_i nije cjelobrojno djeljiv s četiri, nadopunjava se neutralnim elementima do djeljivosti. Sljedovi unutar matrice DBR_i posloženi su stupčasto, tj. jedan stupac matrice DBR_i odgovara jednom slijedu iz baze DB. Svi elementi baze iz globalne memorije čitaju se pomoću dvodimenzionalne teksture. Geometrijska interpretacija navedene memorijske organizacije prikazana je na slici 5.10. Sivom bojom označeni su vertikalno posloženi slijedovi, dok je bijelom bojom označen neiskorišten memorijski prostor.



Slika 5.10: Geometrijska interpretacija baze kratkih sljedova.

Za rješavanje su još potrebne matrica duljina $lengths$ i matrica ocjena $scores$. Obje matrice su dimenzija $DB_r \times BT$. Element $lengths(i, j)$ matrice duljina odgovara duljini slijeda koji se nalazi u stupcu j matrice DBR_i. Element matrice ocjene $scores(i, j)$ matrice ocjena odgovara ocjeni slijeda koji se nalazi u stupcu j matrice DBR_i. Elementi matrice duljina i ocjene s koordinatama $(DBR_r - 1, \mod (db, BT)..BT - 1)$ su postavljeni na nulu te se ne koriste. Svaka matrica DBR_i rješava se nezavisnim pokretanjem CUDA jezgre, tj. CUDA jezgra se poziva DB_r puta. Iz tog razloga dovoljna je samo jedna matrica horizontalne sabirnice dimenzija $n_{max} \times BT$. Svaki stupac matrice horizontalne sabirnice odgovara horizontalnoj sabirnici potrebnoj za rješavanje jednog slijeda.

Kako bi se dodatno smanjio broj čitanja i pisanja u globalnu memoriju, za razliku od *wavefront* metode koja rješava četiri retka u jednom prolazu, rješava se osam elemenata u jednom prolazu. Ako subjektni slijed nije cjelobrojno djeljiv s osam, na-

dopunjava se neutralnim elementima do djeljivosti te se tada kreira njegov profil. K susjedstvo definirano je slično kao i kod *wavefront* metode, osim što elementi K_l i K_r imaju osam elemenata. Implementacijski, K susjedstvo je izvedeno s po dva umjesto jednog četveročlanog vektora. Jednim čitanjem iz globalne memorije čitaju se četiri elementa slijeda iz baze. Za svaki od četiri učitana elementa, s dva čitanja učitavamo po osam ocjena iz profila subjektnog slijeda, jednim čitanjem učitavamo vrijednost horizontalne sabirnice te jednim pisanjem zapisujemo vrijednost u sabirnicu. Ukupno sa 13 čitanja iz i 4 pisanja u globalnu memoriju rješavamo 32 elementa matrice. Pseudokod opisanog algoritma dan je slikom 5.11.

```
1  DBSHORT(r_B, r_b):
2  id = threadIdx.x + B blockIdx.x;
3  n' = lengths(r_B, id);
4  init(H_bus(0..n, id));
5  s = -inf;
6  for i ← 0 to m'/8 do
7      for j ← 0 to n'/4 do
8          c(0..3) = BASE(r_b + j, id);
9          init(K);
10         for k ← 0 to 4 do
11             K_up = H_bus(id, 4j + k);
12             qp(0..3) = QP(2i, c_k);
13             qp(4..7) = QP(2i + 1, c_k);
14             solve(K, qp);
15             update(K, s);
16             H_bus(id, 4j + k) = K_r(7), F;
17             K_l = K_r;
18             K_d = K_up;
19         end
20     end
21 end
22 scores(r_B, id) = s;
```

(a) CUDA jezgra algoritma.

```
1  DBSHORTCPU:
2  for i ← 0 to DB_r do
3      r_b = db_min((i+1)BT,db)-1;
4      DBSHORT«<B, T»>(i, r_b);
5  end
```

(b) Pozivanje CUDA jezgre algoritma.

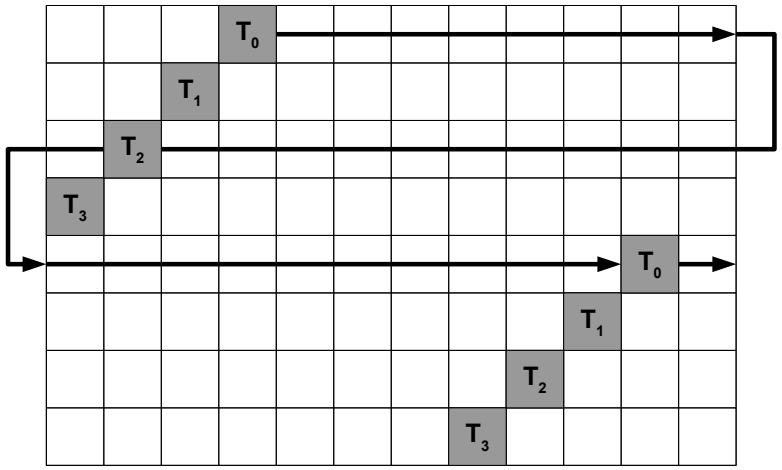
Slika 5.11: Pseudokod CUDA implementacije ocijenjivanje poravnanja baze kratkih sljedova.

Memorijska potrošnja navedenog algoritma uvelike ovisi o raspodjeli duljina sljedova u bazi DB. Gornja granica potrošnje baze DB je $n_{max} \left\lceil \frac{db}{BT} \right\rceil$ bajtova. Upravo zbog navedene memorijske potrošnje navedeni algoritam nije pogodan za duge sljedove u bazi DB. Granica n_{max} postavlja se oko vrijednosti 3000. Od značajnije potrošnje još valja napomenuti potrošnju profila subjektnog slijeda od $26m$ bajtova te potrošnju horizontalne sabirnice od $8n_{max}BT$ bajtova.

5.2.3. Ocjenjivanje poravnanja baze dugih sljedova

Ocjenjivanje poravnanja baze dugih sljedova modifikacija je *wavefront* metode. Za razliku od *wavefront* metode, za poravnanje jednog slijeda sa subjektnim slijedom koristi se samo jedan CUDA blok. Ovo značajno pojednostavljuje implementaciju zbog nemogućnosti sinkronizacijskih problema među blokovima. Označimo bazu sljedova s DB te broj sljedova u bazi db.

Memorijska organizacija mnogo je jednostavnija nego organizacija pri ocjenjivanju poravnanja baze kratkih sljedova. Označimo sljedove u bazi DB s $DB_i, i \in [0, db-1]$ te njihove duljine s db_i gdje je db_i duljina sljeda DB_i. S N označimo vrijednost $\sum_0^{db-1} db_i$, tj. zbroj duljina svih sljedova u bazi. Baza DB je organizirana na način da su sljedovi DB_i memorijski slijedno zapisani u memoriji, tj. kao niz od N elemenata. Dodatno, potreban je niz duljina sljedova $lengths$ te niz odmaka sljedova od početka memorijskog zapisa $offsets$. Oba niza sastoje se od db elemenata. Dodatno, za rješavanje je potrebna horizontalna sabirnica duljine N.



Slika 5.12: Elementi rješavanja jedne dretve bloka pri poravnanju baze dugih sljedova.

Svaki CUDA blok B_i rješava sljedove $DB_{xB+i}, x \in R, xB + i < db$. Svi navedeni sljedovi rješavaju se jednim pokretanjem CUDA jezgre. K-susjedstvo rješavanja definirano je isto kao i kod *wavefront* metode, tj. svaka dretva rješava u jednom prolazu četiri retka. Iz tog razloga, ako subjektni slijed nije cjelobrojno djeljiv s četiri, nadopunjava se neutralnim elementima do djeljivosti te se tada kreira njegov profil. Označimo duljinu takvog subjektnog slijeda s m'. Dretva T_i rješava elemente matrice s koordinatama $(4(Tx+i)..4(Tx+i)+3, 0..n), x \in R, 4(Tx+i) < m'$. Geometrijska interpretacija navedenih koordinata prikazana je na slici 5.12. Dodatno, svaka dretva

T_i čeka prvih i koraka te na kraju čeka T_{T-1-i} koraka kako bi se elementi rješavali među dretvama po sporednoj dijagonali. Memorijska implementacija horizontalne sabirnice jednaka je kao i kod *wavefront* metode. Pseudokod opisanog algoritma dan je slikom 5.13. Memorijska potrošnja baze DB iznosi N bajtova, horizontalne sabirnice $8N$ bajtova te dodatno profil subjektnog slijeda troši $26m'$ bajtova.

DBLONG($lengths, offsets, scores$):
$i =$ blockIdx.x;
while $i < b$ **do**
  $off = offsets(i)$;
  $n' = lengths(i)$;
  $s = -\inf, c =$ -threadIdx.x, $r = 4$threadIdx.x;
  init(K);
  for $j \leftarrow 0$ **to** $\lceil \frac{m'}{4T} \rceil n' + T - 1$ **do**
    if $c > 0$ *and* $r < m'$ **then**
      $K_u = H_{bus}(c + off,$ threadIdx.x$)$;
      $qp(0..3) = QP(r, BASE(off + c))$;
      $solve(K, qp)$;
      $update(K, s)$;
      $H_{bus}(c + off,$ threadIdx.x$) = K_r.w, F$;
    end
    $c = c + 1$;
    if $c = n'$ **then**
      init(K);
      $c = 0, r = r +$ 4threadIdx.x;
    end
  end
  sreduce(s);
  if *threadIdx.x = 0* **then** $scores(i) = s$;
  $i = i + B$;
end

Slika 5.13: Pseudokod CUDA implementacije ocijenjivanje poravnanja baze dugih sljedova.

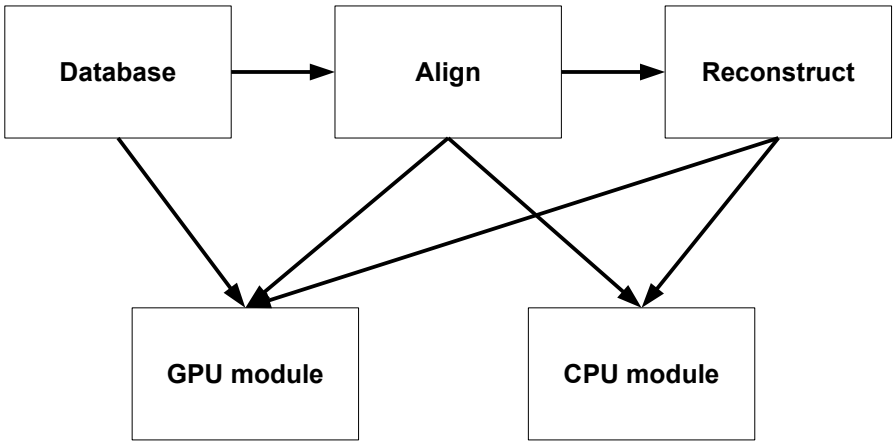
6. Biblioteka SW#

Namjena biblioteke SW# je pružiti jednostavan i efikasan način za korištenje determinističkih algoritama u realnim vremenskim ograničenjima uz nisku memorijsku potrošnju. Biblioteka nudi funkcije za poravnavanje parova sljedova, parova skupova sljedova te razne pomoćne funkcije korisne pri korištenju algoritama poravnavanja sljedova. Izvorni kod biblioteke pisan je u jezicima C, C++ i CUDA. Biblioteka je javno dostupna na Windows, Mac OS i Linux platformama. Osnovna namjena biblioteke je korištenje s drugim alatima, no uz biblioteku u SW# paketu dolazi nekoliko modula koji iskorištavaju njenu funkcionalnost.

6.1. SW# jezgra

SW# jezgra osnova je svih ostalih modula dostupnih u SW# paketu. Ona sadrži sve ključne funkcionalnosti te je namijenjena isključivo za korištenje u drugim modulima ili aplikacijama. Sama jezgra sastoji se od pet glavnih podmodula:

- CPU podmodul - modul sadrži osnovne funkcije poravnanja sljedova.
- GPU podmodul - modul sadrži sve funkcije koje su implementirane za CUDA arhitekturu.
- *Reconstruct* podmodul - modul objedinjuje CPU i GPU funkcije za rekonstrukciju kroz vlastito sučelje te je vremenski i memorijski učinkovit.
- *Align* podmodul - modul objedinjuje CPU i GPU funkcije za poravnavanje parova sljedova kroz vlastito sučelje te je vremenski i memorijski učinkovit. Ovaj podmodul iskorištava *reconstruct* podmodul.
- *Database* podmodul - modul objedinjuje CPU i GPU funkcije za poravnanje parova skupova sljedova kroz vlastito sučelje te je vremenski i memorijski učinkovit. Ovaj podmodul iskorištava *align* podmodul.



Slika 6.1: Ovisnost podmodula SW# jezgre.

U nastavku su za svaki podmodul navedene ključne funkcije te je ukratko objašnjena njihova implementacija.

6.1.1. GPU podmodul

HW_END_DATA_GPU($s_1, s_2, m, n, d, e, \text{sim}()$):
return e_1, e_2, s

Funkcija koristi *wavefont* metodu objašnjenu u poglavlju 5.1.1. Početni uvjeti te funkcija *solve wavefont* metode definirani su formalno u poglavlju 2.3.4. Funkcija *solve* ne brine o poziciji najbolje ocjene poravnanja. Iz definicije algoritma poluglobalnog poravnanja zaključujemo da uvjek vrijedi $e_1 = m$, tj. rezultat se uvjek nalazi u zadnjem retku matrice. Funkcija koristi metodu dopunjavanja elemenata *wavefront* metode s kontrolom rubnih procjepa prilagođenu ekstrakciji zadnjih redaka matrica H i F te pronalazi maksimalnu vrijednost u zadnjem retku matrice H koja je ujedno i rješenje. Mogućnost kontrole rubnih procjepa se ne koristi. Povratne vrijednosti funkcije su ocjena najboljeg poravnanja algoritma poluglobalnog poravnanja te pozicija elementa kojem odgovara navedena ocjena.

NW_FIND_SCORE_GPU($s_1, s_2, m, n, d, e, \text{sim}(), g_{f1}, g_{f2}, s$):
return e_1, e_2

Funkcija koristi *wavefont* metodu objašnjenu u poglavlju 5.1.1. Početni uvjeti te funkcija *solve wavefont* metode definirani su formalno u poglavlju 2.3.2. Dodatno, funkcija *solve* uspoređuje vrijednost matrice H svakog elementa s predanim parametrom s te

pamti poziciju ukoliko su navedene vrijednosti jednake. Nakon izvođenja svakog koraka vanjske dijagonale *wavefont* metode provjerava se je li tražena vrijednost nađena te se u tom slučaju ne izvode sljedeći koraci vanjskih dijagonala. Zbog poznate ocjene poravnanja s, funkcija koristi Ukkonenov algoritam prilagođen *wavefront* metodi. Povratna vrijednosti funkcije je pozicija elementa s vrijednošću matrice H jednakom s. Ako ocjena poravnanja s ni je pronađena, funkcija dojavljuje pogrešku.

NW_LINEAR_DATA_GPU($s_1, s_2, m, n, d, e, \text{sim}(), g_{f1}, g_{f2}, u_l, u_r$):
return H_1, F_1

Funkcija koristi *wavefont* metodu objašnjenu u poglavlju 5.1.1. Početni uvjeti te funkcija *solve wavefont* metode definirani su formalno u poglavlju 2.3.2. Funkcija koristi metodu dopunjavanja elemenata *wavefront* metode s kontrolom rubnih procjepa prilagođenu ekstrakciji zadnjih redaka matrica H i F te Ukkonenov algoritam prilagođen *wavefront* metodi. Povratna vrijednosti funkcije su zadnji redak matrice H i F.

SW_END_DATA_GPU($s_1, s_2, m, n, d, e, \text{sim}()$):
return e_1, e_2, s, H_1, F_1

Funkcija koristi *wavefont* metodu objašnjenu u poglavlju 5.1.1. Početni uvjeti te funkcija *solve wavefont* metode definirani su formalno u poglavlju 2.3.3. Funkcija koristi metodu dopunjavanja elemenata *wavefront* metode s kontrolom rubnih procjepa prilagođenu ekstrakciji zadnjih redaka matrica H i F te algoritam dinamičkog odbacivanja elemenata prilagođen *wavefront* metodi. Mogućnost kontrole rubnih procjepa se ne koristi. Povratna vrijednosti funkcije su zadnji redak matrice H i F, ocjena najboljeg poravnanja Smith-Watermanova algoritma poravnanja te pozicija elementa kojem odgovara navedena ocjena.

SCORE_DATABASE_GPU($s_1[], s_2[], m[], n[], d, e, \text{sim}(), \text{type}$):
return $s[[]]$

Funkcija implementira algoritme objašnjene u poglavljima 5.2.2 i 5.2.3. Sljedovi s_2 dijele se na kratke i duge te se za svaki od sljedova s_1 pozivaju funkcije ocjenjivanja kratke i duge baze. Funkcija *solve* i početni uvjeti ovise o predanom parametru *type*. Moguće funkcije postavljanja početnih uvjeta te rješavanja opisane su u poglavljima 2.3.2, 2.3.3 i 2.3.4. Funkcija vraća ocjenu poravnanja za svaki par s_1, s_2 iz ulaznih skupova sljedova u obliku matrice gdje retci predstavljaju sljedove iz skupa s_1 dok stupci predstavljaju sljedove iz skupa s_2.

6.1.2. CPU podmodul

ALIGN_PAIR_CPU($s_1, s_2, m, n, d, e, \text{sim}(), \text{type}$):
return R

Funkcija ovisno o predanom parametru *type* izvodi jednu od funkcija NW, SW ili HW. Navedene funkcije definirane su u poglavljima 2.3.2, 2.3.3 i 2.3.4. Povratna vrijednost funkcije je poravnanje sljedova s_1 i s_2.

NW_RECN_CPU($s_1, s_2, m, n, d, e, \text{sim}(), g_{f1}, g_{f2}, g_{b1}, g_{b2}, s$):
return p, s'

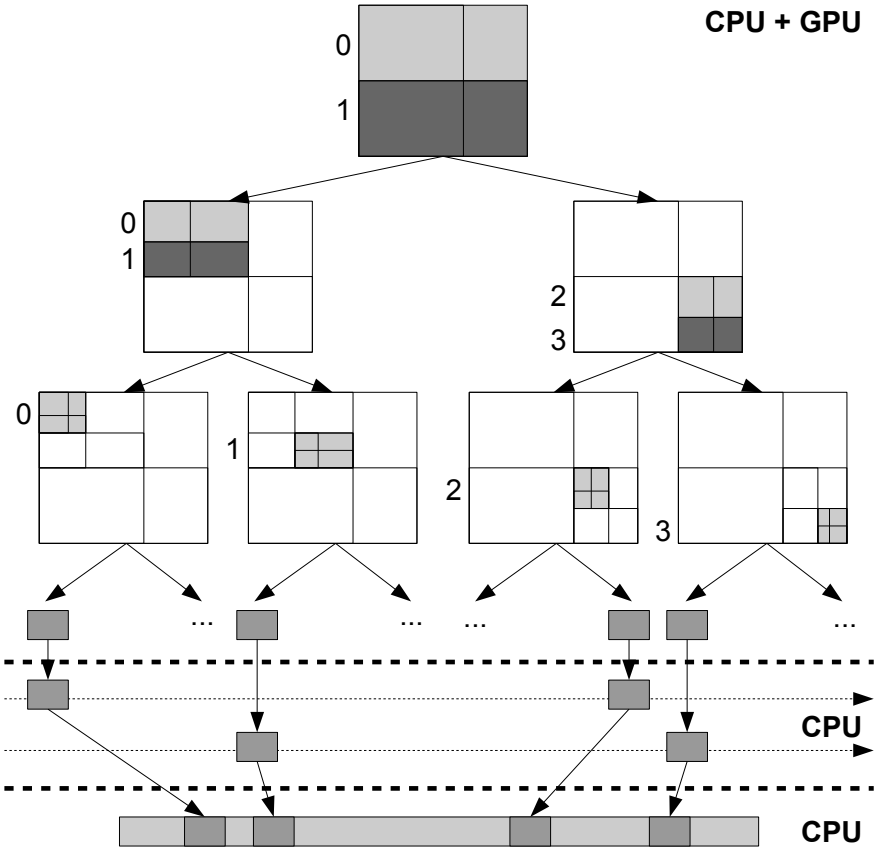
Funkcija je jednaka funkciji NWGU definiranoj u poglavlju 3.6.1. Ako je prethodno nepoznata ocjena poravnanja, parametar s postavlja se na vrijednost $-\inf$. U navedenom slučaju funkcija ne koristi Ukkonenov algoritam. Osim puta poravnanja, dodatno se vraća i poravnanjem dobivena ocjena s'. Navedeno je korisno ako je parametar s nepoznat, u suprotnom slučaju mora vrijediti $s' = s$.

6.1.3. *Reconstruct* podmodul

NW_RECN($s_1, s_2, m, n, d, e, \text{sim}(), g_{f1}, g_{f2}, g_{b1}, g_{b2}, s$):
return p, s'

Funkcija je jednaka funkciji UHIRSCHBERG opisanoj u poglavlju 3.6.3 uz manje implementacijske promjene. Umjesto funkcije NWGU koristi se funkcija NW_RECN_CPU, dok se umjesto funkcije NWHU koristi funkcija NW_LINEAR_DATA_GPU. Pozivi funkcije NW_RECN_CPU se ne rade izravno, već postoji red poslova u koji se dodaju potrebni podaci. Na redu poslova čekaju CPU dretve koje obavljaju poravnanja podmatrica. Na kraju postupka navedena poravnanja podmatrica se sortiraju i spajaju u poravnanje cijele matrice. Funkcije rješavanja gornje i donje podmatrice se obavljaju paralelno ako su u trenutnom podkoraku dostupne barem dvije CUDA kartice. Također podkoraci Hirschbergova algoritma se izvode paralelno ako u trenutnom podkoraku postoje barem dvije dostupne CUDA kartice. Skica navedenog postupka dana je slikom 6.2. Na navedenoj slici pretpostavljeno je da postoje četiri CUDA kartice te da na redu poslova poravnanja podmatrica rade dvije CPU dretve. U svakom koraku različitim bojama su označene podmatrice koje se izvode paralelno. Pokraj svake podmatrice naveden je indeks CUDA kartice na kojoj se rješavanje podmatrice izvodi. Isprekidane crte predstavljaju CPU dretve. Na kraju postupka vidljivo je spajanje poravnanja podmatrica u poravnanje cijele matrice. Ako je prethodno nepoznata ocjena poravnanja, parametar s postavlja se na vrijednost

$-\inf$. U navedenom slučaju u prvom koraku Hirschbergova algoritma se ne koristi Ukkonenov algoritam. Nakon prvog koraka rezultat poravnanja s' je poznat te se u ostalim podkoracima tada koristi Ukkonenov algoritam. Osim puta poravnanja, dodatno se vraća i poravnanjem dobivena ocjena s'. Navedeno je korisno ako je parametar s nepoznat, u suprotnom slučaju mora vrijediti $s' = s$.



Slika 6.2: Skica postupka rekonstrukcije poravnanja dostupnog u biblioteci SW#.

6.1.4. *Align* podmodul

ALIGN_PAIR($s_1, s_2, m, n, d, e, \text{sim}(), \text{type}$):
return R

Ako je vrijednost mn dovoljno mala ili nije dostupna niti jedna CUDA kartica, poziva se funkcija ALIGN_PAIR_CPU. U suprotnom, ovisno o parametru *type* i broju dostupnih CUDA kartica, poziva se jedna od četiri funkcije:

– SW_ALIGN_SINGLE - funkcija se poziva ako je dostupna samo jedna CUDA kartica te parametar *type* zahtijeva lokalno poravnanje.

– SW_ALIGN_DUAL - funkcija se poziva ako je dostupno dvije ili više CUDA kartica te parametar *type* zahtijeva lokalno poravnanje.

– NW_ALIGN - funkcija se poziva ako parametar *type* zahtijeva globalno poravnanje.

– HW_ALIGN - funkcija se poziva ako parametar *type* zahtijeva poluglobalno poravnanje.

Funkcija SW_ALIGN_SINGLE implementira paraleliziranu inačicu Smith-Watermanova algoritma uz iskorištavanje jedne CUDA kartice pri koraku rješavanja. Funkcija prvo pronalazi ocjenu i kraj poravnanja te zatim traži početak poravnanja. Početak poravnanja pronalazi se izvođenjem Needleman-Wunschova algoritma u suprotnom smjeru na podmatrici omeđenoj pronađenom krajnjom točkom. Navedeni postupak naziva se svođenje lokalnog na globalno poravnanje. Uz poznatu početnu i krajnju točku izvodi se korak rekonstrukcije. Možemo reći da se navedeni postupak sastoji od tri koraka: koraka rješavanja, koraka traženja te koraka rekonstrukcije. Kao što je prethodno navedeno, korak rekonstrukcije koristi proizvoljno mnogo kartica. Navedena funkcija dana je pseudokodom na slici 6.3. Geometrijska interpretacija navedenog algoritma prikazana je na slikama 6.5a, 6.6a i 6.7a.

1 **SW_ALIGN_SINGLE**$(s_1, s_2, m, n, d, e, \text{sim}())$:
2 e_1, e_2, s = **SW_END_DATA**$(s_1, s_2, m, n, d, e, \text{sim}())$;
3 b_1, b_2 = **NW_FIND_SCORE**$(s_1(e_1..0), s_2(e_2..0), e_1, e_2, d, e, \text{sim}(), 0, 0, s)$;
4 p = **NW_RECN**$(s_1(b_1..e_1), s_2(b_2..e_2), e_1 - b_1, e_2 - b_2, d, e, \text{sim}(), 0, 0, 0, 0, s)$;
5 **return** $\{b_1, e_1, b_2, e_2, s, p\}$;

Slika 6.3: Pseudokod paralelizirane inačice Smith-Watermanova algoritma uz iskorištavanje jedne CUDA kartice.

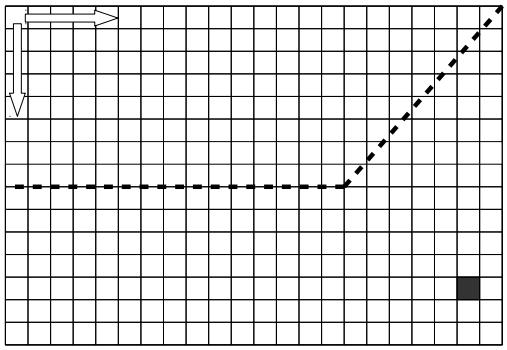
Funkcija SW_ALIGN_DUAL implementira paraleliziranu inačicu Smith-Watermanova algoritma uz iskorištavanje dvije CUDA kartice pri koraku rješavanja. Ideja pristupa je izvesti prvi korak Hirschbergova algoritma na način da se na svakoj kartici izvede jedna polovica matrice. Razlika je u tome što se podmatrice ne rješavaju Needleman-Wunschovim već Smith-Watermanovim algoritmom. Ovime se, osim zadnjih redaka matrica H i F, dobiva i maksimalna ocjena te njena pozicija za obje podmatrice. Usto računa se pozicija i maksimalna ocjena pomoću zadnjih redaka H i F dviju podmatrica. Nazovimo tako izračunatu ocjenu ocjena sredine. Kao ocjena poravnanja uzima se maksimalna od tri izračunate maksimalne ocjene. Dosad navedeni postupak je korak rješavanja algoritma te je njegova geometrijska interpretacija prikazana na slici 6.5b.

```
1  SW_ALIGN_DUAL(s_1, s_2, m, n, d, e, sim()):
2  r = m/2;
3  e_1u, e_2u, s_u, H_1u, F_1u = SW_END_DATA(s_1(0..r), s_2, r, n, d, e, sim());
4  e_1d, e_2d, s_d, H_1d, F_1d = SW_END_DATA(s_1(m..r), s_2(n..0), m − r, n, d, e, sim());
5  s_m = s_mu = s_md = −inf, c = −1, g = 0;
6  for i ← 0 to n do
7  |   h = H_1u(i) + H_1d(n − i);
8  |   a = F_1u(i) + F_1d(n − i) + d − e;
9  |   if h > s_m then s_m = h, c = i, g = 0, s_mu = H_1u(i), s_md = H_1d(n − i);
10 |   if a > s_m then s_m = a, c = i, g = 1, s_mu = F_1u(i) + d − e, s_md = F_1d(n − i) + d − e;
11 end
12 s = max(s_u, s_d, s_m);
13 if s = s_u then
14 |   e_1 = e_1u, e_2 = e_2u;
15 |   b_1, b_2 = NW_FIND_SCORE(s_1(e_1..0), s_2(e_2..0), e_1, e_2, d, e, sim(), 0, 0, s);
16 |   p = NW_RECN(s_1(b_1..e_1), s_2(b_2..e_2), e_1 − b_1, e_2 − b_2, d, e, sim(), 0, 0, 0, 0, s);
17 |   return {b_1, e_1, b_2, e_2, s, p};
18 end
19 if s = s_d then
20 |   b_1 = m − e_1d, b_2 = n − e_2u;
21 |   e_1, e_2 = NW_FIND_SCORE(s_1(b_1..m), s_2(b_2..n), e_1d, e_2d, d, e, sim(), 0, 0, s);
22 |   p = NW_RECN(s_1(b_1..e_1), s_2(b_2..e_2), e_1 − b_1, e_2 − b_2, d, e, sim(), 0, 0, 0, 0, s);
23 |   return {b_1, e_1, b_2, e_2, s, p};
24 end
25 b_1, b_2 = NW_FIND_SCORE(s_1(r..0), s_2(c..0), r, c, d, e, sim(), g, 0, s_mu);
26 e_1, e_2 = NW_FIND_SCORE(s_1(r..m), s_2(c..n), m − r, n − c, d, e, sim(), g, 0s_md);
27 p_1 = NW_RECN(s_1(b_1..r), s_2(b_2..c), r − b_1, c − b_2, d, e, sim(), 0, g, 0, 0, s);
28 p_2 = NW_RECN(s_1(r..e_1), s_2(c..e_2), e_1 − r, e_2 − c, d, e, sim(), g, 0, 0, 0, s);
29 return {b_1, e_1, b_2, e_2, s, p_1 ∪ p_2};
```

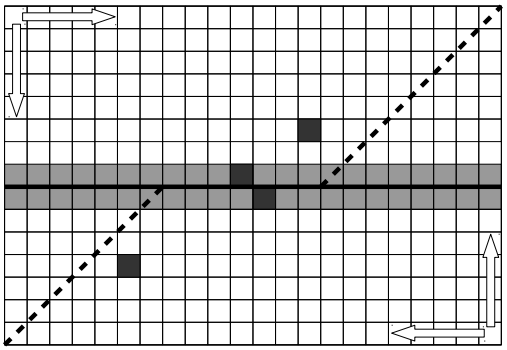
Slika 6.4: Pseudokod paralelizirane inačice Smith-Watermanova algoritma uz iskorištavanje dvije ili više CUDA kartica.

Koraci traženja i rekonstrukcije granaju se na tri slučaja: slučaj kada je ocjena poravnanja jednaka ocjeni gornje podmatrice, slučaj kada je ocjena poravnanja jednaka ocjeni donje podmatrice te kada je ocjena poravnanja jednaka ocjeni sredine. Ako je ocjena poravnanja jednaka ocjeni gornje podmatrice, tada su koraci traženja i rekonstrukcije jednaki kao i kod funkcije SW_ALIGN_SINGLE. Ako je ocjena poravnanja jednaka ocjeni donje podmatrice, tada je pozicija ocjene poravnanja jednaka poziciji početka poravnanja. Korak traženja izvodi se na donjoj podmatrici u standardnom smjeru od pozicije ocjene poravnanja te se nalazi kraj poravnanja. Kada su poznati početak i kraj poravnanja, korak rekonstrukcije jednak je kao i u prvom slučaju. Ako je ocjena poravnanja jednaka ocjeni sredine, tada se provedbom koraka Hirschbergova algoritma, osim pozicije maksimalne ocjene izračunate pomoću zadnjih redaka dvaju podmatrica,

dobivaju i ocjene redaka gornje i donje podmatrice koje tvore maksimalnu ocjenu. Paralelno se traže navedene ocjene na gornjoj i donjoj podmatrici. Na gornjoj podmatrici traži se ocjena u suprotnom smjeru, ograničena pozicijom elemenata zadnjeg reda gornje podmatrice koji tvori maksimalnu ocjenu. Na donjoj podmatrici traži se ocjena u normalnom smjeru, ograničena pozicijom elemenata zadnjeg reda donje podmatrice koji tvori maksimalnu ocjenu. Time se dobivaju pozicija početka i kraja poravnanja. Kako je poznata točka na sredini matrice, korak rekonstrukcije se također izvodi paralelno na gornjoj i donjoj podmatrici. Na gornjoj podmatrici korak rekonstrukcije provodi se na podmatrici ograničenoj početkom poravnanja i pozicijom elemenata zadnjeg reda gornje podmatrice koji tvori maksimalnu ocjenu. Na donjoj podmatrici korak rekonstrukcije provodi se na podmatrici ograničenoj pozicijom elemenata zadnjeg reda donje podmatrice koji tvori maksimalnu ocjenu te krajem poravnanja. Tako dobivena dva poravnanja se na kraju postupka spajaju u jedno. Geometrijske interpretacije koraka traženja i rekonstrukcije navedenog slučaja prikazane su na slikama 6.6b i 6.7b.



(a) Jednokartični korak rješavanja paralelne inačice Smith-Watermanova algoritma.

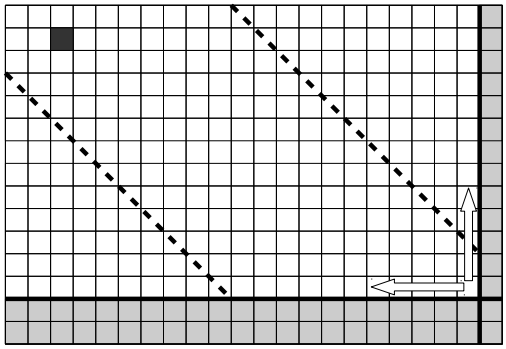


(b) Višekartični korak rješavanja paralelne inačice Smith-Watermanova algoritma.

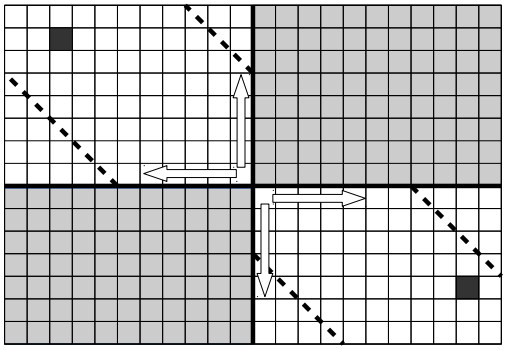
Slika 6.5: Usporedba koraka rješavanja jednokartične i višekartične paralelne inačice Smith-Watermanova algoritma.

Na slici 6.5 vidljiva je usporedba koraka rješavanja jednokartične i višekartične paralelne inačice Smith-Watermanova algoritma. Isprekidanim linijama odvojeno je područje djelovanja algoritma dinamičkog odbacivanja elemenata. Tamnosivom bojom označeni su elementi s najvećim vrijednostima. U slučaju višekartičnog rješavanja označeni su elementi za sva tri slučaja. Na slici 6.6 vidljiva je usporedba koraka traženja jednokartične i višekartične paralelne inačice Smith-Watermanova algoritma. Isprekidanim linijama označeno je područje rješavanja dobiveno Ukkonenovim algoritmom. Za višekartično rješavanje prikazan je slučaj kada je maksimalna ocjena jednaka ocjeni

sredine. Tamnosivom bojom označeni su elementi traženi u ovom koraku. Na slici 6.7 vidljiva je usporedba koraka rekonstrukcije jednokartične i višekartične paralelne inačice Smith-Watermanova algoritma. Isprekidanim linijama označeno je područje rješavanja dobiveno Ukkonenovim algoritmom. Za višekartično rješavanje prikazan je slučaj kada je maksimalna ocjena jednaka ocjeni sredine. Višekartično rješavanje preskače prvi korak Hirschbergova algoritma jer je prvi korak Hirschbergova algoritma obavljen u koraku rješavanja.

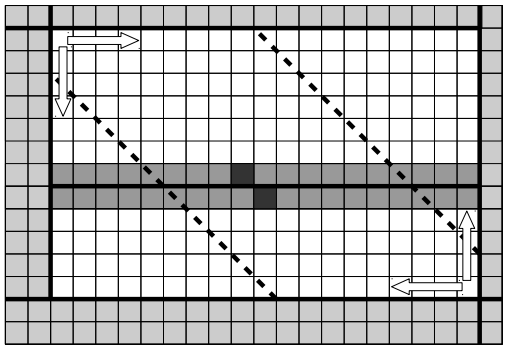


(a) Jednokartični korak traženja paralelne inačice Smith-Watermanova algoritma.

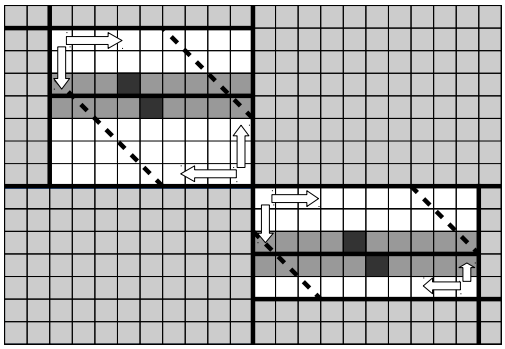


(b) Višekartični korak traženja paralelne inačice Smith-Watermanova algoritma.

Slika 6.6: Usporedba koraka traženja jednokartične i višekartične paralelne inačice Smith-Watermanova algoritma.



(a) Jednokartični korak rekonstrukcije paralelne inačice Smith-Watermanova algoritma.



(b) Višekartični korak rekonstrukcije paralelne inačice Smith-Watermanova algoritma.

Slika 6.7: Usporedba koraka rekonstrukcije jednokartične i višekartične paralelne inačice Smith-Watermanova algoritma.

Funkcija NW_ALIGN se u potpunosti obavlja pomoću funkcije NW_RECN, jer su prema definiciji Needleman-Wunschova algoritma svi elementi oba slijeda dio poravnanja. Iz istog razloga funkcija iskorištava proizvoljan broj CUDA kartica. Pseudokod funkcije dan je slikom 6.8.

```
1 NW_ALIGN(s_1, s_2, m, n, d, e, sim()):
2 p, s' = NW_RECN(s_1, s_2, m, n, d, e, sim(), 0, 0, 0, 0, −inf);
3 return {0, m, 0, n, s', p};
```

Slika 6.8: Pseudokod paralelizirane inačice Needleman-Wunschova algoritma.

Funkcija HW_ALIGN jednaka je funkciji SW_ALIGN_SINGLE, osim što se u koraku rješavanja umjesto funkcije SW_END_DATA koristi funkcija HW_END_DATA. Koraci traženja i rekontrukcije jednaki su kod obje funkcije. Kao i kod funkcije SW_ALIGN_SINGLE, koraci rješavanja i traženja iskorištavaju jednu CUDA karticu, dok se u koraku rekonstrukcije iskorištava proizvoljno mnogo CUDA kartica. Pseudokod funkcije dan je slikom 6.9.

```
1 HW_ALIGN(s_1, s_2, m, n, d, e, sim()):
2 e_1, e_2, s = HW_END_DATA(s_1, s_2, m, n, d, e, sim());
3 ...
4 return {b_1, e_1, b_2, e_2, s, p};
```

Slika 6.9: Pseudokod paralelizirane inačice algoritma poluglobalnog poravnanja gdje su prikazane promjene u odnosu na pseudokod dan slikom 6.3.

6.1.5. *Database* podmodul

ALIGN_DATABASE($s_1[], s_2[], m[], n[], d, e, \text{sim}(), \text{type}, E_{max}, R_{max}$):
return $R[[]]$

Funkcija najprije poziva funkciju SCORE_DATABASE_GPU kako bi dobila ocjene poravnanja za svaki par s_1 i s_2 iz ulaznih skupova sljedova. Zatim se za svaki slijed iz skupa s_1 sortiraju ocjene poravnanja koristeći E vrijednost definiranu u poglavlju 2.4 uz korištenje parametara definiranih u [9]. Odbacuju se svi parovi čija E vrijednost prelazi E_{max} te se poziva ALIGN_PAIR funkcija nad najviše R_{max} parova za svaki slijed iz skupa s_1. Na navedeni način se za svaki od sljedova iz skupa s_1 vraća najviše R_{max} poravnanja.

6.2. SW# moduli

Uz samu SW# biblioteku, u paketu dolazi i nekoliko modula koji iskorištavaju njene funkcionalnosti. Dostupni moduli su:

- SW#n - modul se koristi za poravnanje jednog para dugih sljedova nukleotida koristeći Smith-Watermanov algoritam. Iako modul radi i na vrlo kratkim nukleotidima, optimiran je za duge nukleotide. Ovaj modul koristi konstantni model ocjenjivanja supstitucija.
- SW#nc - modul se koristi za poravnanje jednog para dugih sljedova nukleotida koristeći Smith-Watermanov algoritam. Za razliku od SW#n modula, ovaj modul pretražuje i komplementarni DNA slijed. Modul vraća poravnanje slijeda s većom ocjenom.
- SW#p - modul se koristi za poravnanje jednog para proteina koristeći Smith-Watermanov algoritam. Ovaj modul koristi tablični model ocjenjivanja supstitucija. Dostupne tablice supstitucija su BLOSUM_45, BLOSUM_50, BLOSUM_62, BLOSUM_80, BLOSUM_90, PAM_30, PAM_70 i PAM_250.
- SW#db - modul se koristi za poravnanje jednog proteina s bazom proteina koristeći Smith-Watermanov algoritam. Model ocjenjivanja poravnanja jednak je kao i kod SW#p modula. Poravnanja se međusobno ocjenjuju koristeći E vrijednost definiranu u poglavlju 2.4 uz korištenje parametara definiranih u [9]. Iako je u biblitoteci podržano i poravnanje nukleotida s bazom nukleotida, ovaj modul to ne nudi zbog relativno rijetke primjene.
- SW#dbmpi - modul je jednak modulu SW#db osim što je prilagođen izvođenju pomoću MPI (eng. *Message Passing Interface*) [16] tehnologije.
- SW#out - modul se koristi za naknadnu obradu rezultata poravnanja parova sljedova.

7. Rezultati

7.1. Alati i metode potrebni za evaluaciju rezultata

7.1.1. Vrijeme izvođenja

Sva mjerenja provedena su na jednoj od tri dostupne kongiruacije. Definirajmo navedene konfiguracije kao $conf_1$, $conf_2$ i $conf_3$. Konfiguracija $conf_1$ ima sljedeće karakteristike:

- Intel® Core™2 Quad CPU Q6600 @ 2.40GHz
- NVIDIA™Corporation GF110 [GeForce GTX 570]
- 8GB RAM
- NVIDIA™CUDA 5.0 SDK
- Archlinux

Konfiguracija $conf_2$ ima sljedeće karakteristike

- Intel® Core™i7-3770 CPU @ 3.40GHz
- 2 $\times$ NVIDIA™Corporation GK104 [GeForce GTX 690]
- 16GB RAM
- NVIDIA™CUDA 5.0 SDK
- Ubuntu 12.04.2 LTS

Valja naglasiti da, iako konfiguracija sadrži fizički dvije grafičke kartice, one se programski ponašaju kao četiri. U nastavku će se zbog jednostavnosti pretpostaviti da i fizički postoje četiri kartice. Konfiguracija $conf_3$ sastoji se od četiri mrežno povezana identična stroja sa sljedećim karakteristikama:

- Intel® Xeon® CPU E5-2667 0 @ 2.90GHz
- 4 $\times$ NVIDIA™Corporation Tesla K20Xm
- 16GB RAM

– NVIDIA^{TM}CUDA 5.0 SDK

– CentOS

Sva mjerenja brzine izvođenja mjerena su alatom *time*, koji je dostupan kao standardni dio *linux* ljuske.

7.1.2. Binarna klasifikacija

Pri ocjenjivanju algoritama poravnanja sekvenci često se koristi binarna klasifikacija. Binarna klasifikacija općenit je postupak ocjenjivanja algoritama. Pretpostavimo da postoji određen broj testnih primjera podijeljenih u dvije skupine, pozitivne primjere i negativne primjere. Za svaki od primjera provodi se željeni algoritam te se rezultat svodi na razvrstavanje primjera u pozitivnu ili negativnu skupinu. U sljedećem koraku uspoređuje se poznato stanje primjera i stanje dobiveno testiranjem. Pri uspoređivanju događa se jedan od četiri slučaja:

– TP (eng. *True Positive*) - pozitvan primjer testiranjem je označen pozitivno.

– FP (eng. *False Positive*) - negativan primjer testiranjem je označen pozitivno.

– TN (eng. *True Negative*) - negativan primjer testiranjem je označen negativno.

– FN (eng. *False Negative*) - pozitvan primjer testiranjem je označen negativno.

Navedeni slučajevi također su prikazani su tablicom 7.1. Iz navedena četiri parametra izvode se ocjene točnosti, preciznosti, osjetljivosti te specifičnosti algoritma.

Tablica 7.1: Mogući ishodi binarne klasifikacije.

	Pozitivan primjer	Negativan primjer
Pozitivan test	TP	FP
Negativan test	FN	TN

Točnost (eng. *accuracy*) algoritma definira se kao mjera bliskosti poznatih stanja primjera i stanja primjera dobivenih testiranjem. Formalno, točnost je definirana izrazom:

$$\text{točnost} = \frac{TP + TN}{TP + FP + TN + FN} \tag{7.1}$$

Preciznost (eng. *precision*) algoritma definira se kao mjera bliskosti između više ponovljenih mjerenja u nepromijenjenoj okolini. Formalno, preciznost je definirana izrazom:

$$\text{preciznost} = \frac{TP}{TP + FP} \tag{7.2}$$

Osjetljivost (eng. *sensitivity*) algoritma definira se kao mjera koja iskazuje svojstvo algoritma da prepozna pozitivne primjere. Formalno, osjetljivost je definirana izrazom:

$$\text{osjetljivost} = \frac{TP}{TP + FN} \tag{7.3}$$

Specifičnost (eng. *specificity*) algoritma definira se kao mjera koja iskazuje svojstvo algoritma da prepozna negativne primjere. Formalno, specifičnost je definirana izrazom:

$$\text{specifičnost} = \frac{TN}{TN + FP} \tag{7.4}$$

Uz standardne ocjene binarne klasifikacije, često se koristi i mjera pokrivenosti. Uz slučaj da algoritam ocijeni primjer kao pozitivan ili negativan može se dogoditi da algoritam uopće ne ocijeni primjer. Pokrivenost (eng. *coverage*) algoritma definira se kao mjera broja ocijenjenih primjera. Formalno, pokrivenost je definirana izrazom:

$$\text{pokrivenost} = \frac{TP + TN + FP + FN}{\text{ukupni broj primjera}} \tag{7.5}$$

7.1.3. Ostali alati poravnanja sljedova

- CUDAlign [14] - implementacija poravnanja parova sljedova pomoću Smith-Watermanova algoritma na CUDA arhitekturi.
- SSW [17] - implementacija Smith-Watermanova algoritma koja koristi procesorske SIMD (eng. *Single Instruction, Multiple Data*) instrukcije.
- BLAST [7] (eng. *Basic Local Alignment Search Tool*) - najpoznatija heuristička metoda poravnanja sljedova.
- PSIBLAST [18] (eng. *Position-Specific Iterative BLAST*) - algoritam koristi BLAST algoritam te iterativno pokušava poboljšati njegovo rješenje.
- GPUBLAST [19] - implementacija BLAST algoritma na CUDA arhitekturi.
- SSEARCH [20] - implementacija Smith-Watermanova algoritma dostupna u Fasta programskom paketu.
- CUDASW++ [21] - implementacija poravnanja jednog slijeda s bazom sljedova pomoću Smith-Watermanova algoritma na CUDA arhitekturi.
- blat [8] (eng. *The Blast-Like Alignment Tool*) - algoritam sličan BLASTu čija je glavna odlika brzina čime se žrtvuje točnost algoritma.

– SW-FPGA [22] - implementacija Smith-Watermanova algoritma na FPGA (eng. *Field-Programmable Gate Array*) čipovima.

7.2. Algoritmi poravnavanja parova sljedova

Najvažnija odlika alata za poravnanje parova sljedova jest njihova brzina. U nastavku su komparativna mjerenja brzina raznih alata za poravnanje parova sljedova. Za mjerenje brzine biblioteke SW# koristi se modul SW#n, tj. modul za poravnavanje nukleotida koristeći Smith-Watermanov algoritam. Tablica 7.2 prikazuje parove nukleotida, preuzete iz GenBank baze nukleotida [23], korištene za mjerenja.

Tablica 7.2: Parovi sljedova korišteni za testiranje SW#n modula.

Veličina	Duljina slijeda	Serijski broj	Ime
162K×172K	162,114	NC_000898.1	Human herpesvirus 6B
	171,823	NC_007605.1	Human herpesvirus 4 type 1
542K×536K	542,868	NC_003064.2	Agrobacterium tumefaciens str. C58
	536,135	NC_000914.1	Rhizobium sp. NGR234 plasmid pNGR234a
1044K×1073K	1,044,459	CP000051.1	Chlamydia trachomatis A/HAR-13
	1,072,950	AE002160.2	Chlamydia muridarum Nigg
3147K×3283K	3,147,090	BA000035.2	Corynebacterium efficiens YS-314
	3,282,708	BX927147.1	Corynebacterium glutamicum ATCC 13032
5227K×5229K	5,227,293	AE016879.1	Bacillus anthracis str. Ames
	5,228,663	AE017225.1	Bacillus anthracis str. Sterne
7146K×5227K	7,145,576	NC_005027.1	Rhodopirellula baltica SH 1
	5,227,293	NC_003997.3	Bacillus anthracis str. Ames
23013K×24544K	23,011,544	NT_033779.4	Drosophila melanogaster chromosome 2L
	24,543,557	NT_037436.3	Drosophila melanogaster chromosome 3L
59374K×23953K	59,373,566	NC_000024.9	Homo sapiens chromosome Y, GRCh37.p10
	23,952,694	NC_006492.2	Pan troglodytes chromosome Y
32799K×46944K	32,799,110	BA000046.3	Pan troglodytes DNA, chromosome 22
	46,944,323	NC_000021.7	Homo sapiens chromosome 21

Kako bi izmjerili učinkovitost CUDA inačice algoritama poravnanja, dobivena vremena izvođenja uspoređena su s dvije CPU inačice Smith-Watermanova algoritma, starijom inačicom SW#n modula [24] bez korištenja CUDA kartica te s alatom SSW. Valja napomenuti da alat SSW nema memorijski efikasan korak rekonstrukcije te ne može obraditi dulja poravnanja. Iz tog razloga mjerena je samo brzina koraka rješavanja alata SSW. Sva mjerenja prizana su tablicom 7.3. Možemo primjetiti da ubrzanje algoritma seže i do preko 400 puta. Pri poravnanju kraćih parova sljedova dolazi do izražaja inicijalizacija CUDA arhitekture, koja uzima oko jedne sekunde ovisno o konfiguraciji. Mjerenje starije inačice SW#n modula obavljeno je na konfiguraciji $conf_1$ te. alata SSW na konfiguraciji $conf_3$. Vrijeme izvođenja uspoređeno je i s CUDA ina-

čicom Smith-Watermanova algoritma prikazanom u radu [14]. Vlastita mjerenja nisu obavljena jer navedeni alat ni je javno dostupan. SW#n modul mjeren je na konfiguracijama $conf_1$ i $conf_2$. Možemo usporediti vremena alata CUDAlign, koja su dobivena mjerenjima na kartici GTX560, s vremenima izvođenja izmjerenim na GTX570 kartici. Ove kartice su veoma slične, no svejedno valja uzeti u obzir da je kartica GTX570 nešto brža. Usporednim promatranjem tablica 7.3 i 7.4. Primjećujemo da su vremena veoma slična, osim u slučajevima duljih poravnanja. Alat CUDAlign je u tim slučajevima duplo brži, no troši značajnu količinu memorije, npr. za zadnji par u tablici 7.3, alat troši dodatnih 50GB prostora na disku. Također valja primjetiti da navedeni alat nije prilagođen sustavima s više CUDA kartica za razliku od SW#n modula.

Tablica 7.3: Usporedba vremena izvođenja različitih algoritama poravnavanja parova sljedova.

Veličina	SW# 0.1 Intel® Q6600	SSW Intel® E5-2667	CUDAlign GTX560	SW# GTX570	SW# GTX690 ×1	SW# GTX690 ×2
162K×172K	660s	7.51s	2.1s	2.13s	2.84s	2.60s
542K×536K	9090s	77.42s	11.8s	10.43s	9.14s	5.89s
1044K×1073K		9869s	43.4s	56.32s	44.92s	23.85s
3147K×3283K			367s	301s	236s	120s
5227K×5229K			558s	697s	576s	320s
7146K×5227K			1321s	1078s	844s	426s
23013K×24544K			19757s	15925s	12676s	6417s
59374K×23953K			47123s	40934s	31747s	16422s
32799K×46944K			30369s	60059s	45244s	25122s

Tablica 7.4: Detalji poravnanja dobivenih SW#n modulom.

Veličina	Ocjena	Početak	Kraj	Duljina
162K×172K	18	(41040, 44335)	(41057, 44352)	18
542K×536K	48	(455042, 308466)	(455133, 308557)	92
1044K×1073K	88353	(259025, 606895)	(722724, 1072949)	472249
3147K×3283K	4226	(2977390, 2675374)	(2991492, 2689487)	14582
5227K×5229K	5220960	(0, 1)	(5077641, 4655866)	5229192
7146K×5227K	172	(5077083, 4655314)	(5077641, 4655866)	565
23013K×24544K	9063	(14642625, 11492211)	(14651730, 11501312)	9107
59374K×23953K	1307541	(14367608, 17336758)	(16467140, 19576119)	2280483
32799K×46944K	27206434	(0, 13841680)	(32718230, 46919079)	33583686

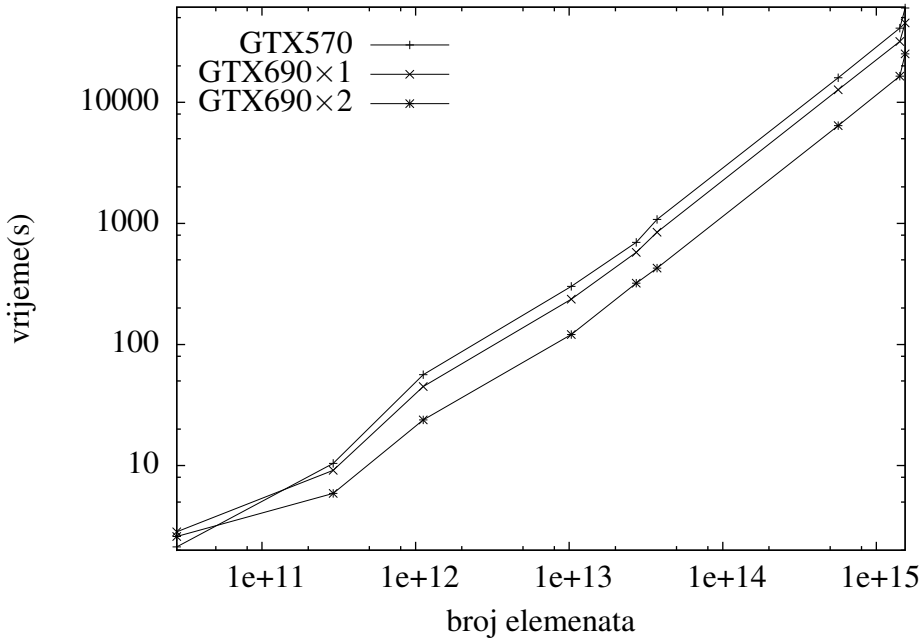
Provjera točnosti rezultata poravnanja napravljena je usporedbom detalja poravnanja dobivenih SW#n modulom s detaljima poravnanja prikazanim u [14]. Oba alata dala su iste rezlutate. Navedeni rezultati prikazani su tablicom 7.4.

Kako bi ocijenili korisnost algoritma dinamičkog odbacivanja elemenata te Ukkonenova algoritma kroz pojedine korake SW#n modula, obavljena su mjerenja brzine s navedenim algoritmima i bez njih. Mjerenja su obavljena koristeći konfiguraciju $conf_1$. Rezultati su prikazani u tablici 7.5. Oba navedena algoritma uvelike ovise o poravnanju. Što su sljedovi koji se poravavaju sličniji, to oba navedena algoritma više pridonose ubrzanju. Algoritam dinamičkog odbacivanja elemenata ubrza korak rješavanja i do 2 puta. Ukkonenov algoritam ubrza korak traženja i do 8 puta te korak rekonstrukcije i do 7 puta. Ukupno, oba navedena algoritma doprinose ubrzanju i do 4 puta na navedenim parovima nukleotida.

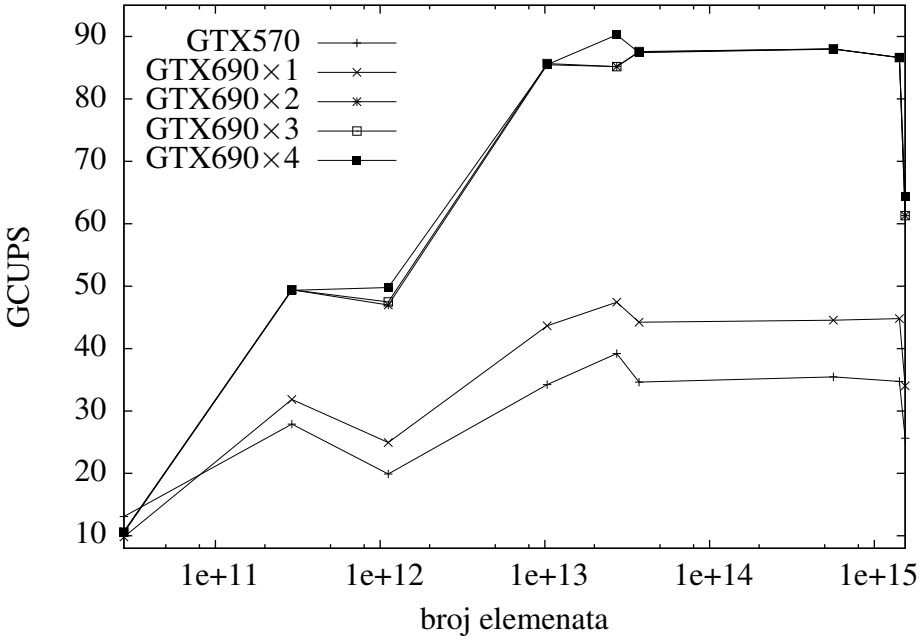
Tablica 7.5: Usporedba vremena izvođenja s uključenim optimizacijskim algoritmima i bez njih u pojedinom koraku SW#n modula.

Veličina	SW# korak 1 Dinamičko odbacivanje		SW# korak 2 Ukkonen		SW# korak 3 Ukkonen		Ukupno	
	✓	✗	✓	✗	✓	✗	✓	✗
162K×172K	1.57s	1.57s	0.01s	0.01s	0.01s	0.01s	2.13s	2.12s
542K×536K	9.84s	9.69s	0.01s	0.01s	0.01s	0.01s	10.43s	10.27s
1044K×1073K	31.21s	34.00s	11.37s	13.87s	13.17s	19.24s	56.32s	67.69s
3147K×3283K	300s	301s	0.17s	0.17s	0.20s	0.20s	301s	302s
5227K×5229K	399s	789s	93.74s	761s	203s	1442s	697s	2994s
7146K×5227K	1077s	1077s	0.06s	0.06s	0.01s	0.01s	1078s	1078s
23013K×24544K	15922s	15950s	0.46s	0.46s	0.02s	0.12s	15925s	15951s
59374K×23953K	40046s	40129s	758s	1049s	126s	272s	40934s	41497s
32799K×46944K	24435s	43551s	19843s	42512s	15778s	53829s	60059s	139895s

Kako bi se prikazala skalabilnost SW#n modula na višekartičnim CUDA sustavima, provedena su mjerenja na konfiguraciji $conf_2$ koristeći od jedne do četiri kartice. Rezultati su grafički prizani na slici 7.1. Na istoj slici također je označena brzina izvođenja na konfiguraciji $conf_1$. Primjećujemo značajno ubrzanje od gotovo dva puta u slučaju korištenja jedne i dvije grafičke kartice. Budući da se više od dvije grafičke kartice koriste samo u koraku rekonstrukcije, ubrzanje koristeći tri ili četiri kartice nije značajno te navedene brzine nisu označene na slici. Navedeno vrijedi jer korak rekonstrukcije troši u najgorem slučaju oko četvrtine vremena, što je vidljivo iz tablice 7.5. Uz vrijeme izvođenja često se kao mjera brzine izvođenja koristi mjera GCUPS (eng. *Giga Cell Updates Per Second*) [14] [25] koja označava broj rješenih milijardi elemenata u jednoj sekundi. Dobiveni rezultati grafički su prikazani slikom 7.2. Primjećujemo da SW# modul s dvije CUDA kartice postiže razinu od 85 GCUPSa što je značajno više od 50 GCUPSA navedenih u radu [14].



Slika 7.1: Usporedba vremena izvođenja SW#n modula na različitim CUDA karticama.



Slika 7.2: Usporedba GCUPSa SW#n modula na različitim CUDA karticama.

Uz usporedbu brzina izvođenja SW#n i SW#p modula s CPU i CUDA inačicama alata, napravljena je usporedba i s SW-FPGA alatom koji se izvršava na FPGA arhitekturi. Mjereno je trajanje izvođenja CUDA jezgre koraka rješavanja te je uspoređeno s brzinom izvođenja na FPGA modulu. Rezultati su priazani tablicom 7.6. Primjeću-

jemo sumjerljivost dobivenih rezultata. S porastom veličine ulaznih sljedova smanjuje je i razlika u brzini. Također valja napomenuti da navedeni modul izvršava samo korak rješavanja Smith-Watermanova algoritma te ne može ocjenjivati sljedove veće od sljedova navedenih u tablici 7.6. Korišteni sljedovi pri mjerenju automatski su generirani.

Tablica 7.6: Usporedba vremena izvođenja CUDA jezgre koraka rješavanja SW#n modula i alata SW-FPGA.

Veličina	Nukleotidi		Proteini	
	FPGA	SW#n	FPGA	SW#p
256×256	0.0002s	0.001s	0.0003s	0.001s
512×512	0.0004s	0.003s	0.0004s	0.003s
1024×1024	0.0005s	0.004s	0.0005s	0.005s
2048×2048	0.0009s	0.006s	0.0008s	0.006s
4096×4096	0.0018s	0.009s	0.0018s	0.010s
8192×8192	0.0047s	0.014s	0.0047s	0.017s
16384×16384	0.014s	0.026s	0.014s	0.031s
32768×32768	0.049s	0.058s	0.049s	0.073s
65536×65536	0.181s	0.144s	0.181s	0.205s

7.3. Algoritmi poravnavanja parova skupova sljedova

7.3.1. Poravnavanje jednog slijeda sa skupom sljedova

Tablica 7.7: Sljedovi korišteni za testiranje SW#db modula.

Duljina	Serijski broj	Ime
110	O74807	Uncharacterized protein C2D10.16 OS=Schizosaccharomyces pombe
195	P19930	Hydrogenase 1 maturation protease OS=Escherichia coli
299	B8E1A7	Ribosomal protein L11 methyltransferase OS=Dictyoglomus turgidum
390	Q3ZAI3	DNA polymerase IV OS=Dehalococcoides ethenogenes
513	P18080	5-aminolevulinate synthase, erythroid-specific, mitochondrial
607	O84416	Lipid-A-disaccharide synthase OS=Chlamydia trachomatis
727	A9BIH4	Phosphoribosylformylglycinamidine synthase 2 OS=Petrotoga mobilis
804	Q2LR26	Phenylalanine–tRNA ligase beta subunit OS=Syntrophus aciditrophicus
980	B4KLY7	Serrate RNA effector molecule homolog OS=Drosophila mojavensis
1485	Q5R7Y0	Brain-specific angiogenesis inhibitor 2 OS=Pongo abelii
5141	Q700K0	SCO-spondin OS=Rattus norvegicus
6733	P0C6V8	Replicase polyprotein 1ab OS=Breda
7094	P0C6W9	Replicase polyprotein 1ab OS=Bovine coronavirus
8081	O01761	Muscle M-line assembly protein unc-89 OS=Caenorhabditis elegans
10746	Q6GGX3	Extracellular matrix-binding protein ebh OS=Staphylococcus aureus
18141	Q9I7U4	Titin OS=Drosophila melanogaster
22152	Q8WXI7	Mucin-16 OS=Homo sapiens
36805	Q3ASY8	Parallel beta-helix repeat OS=Chlorobium chlorochromatii

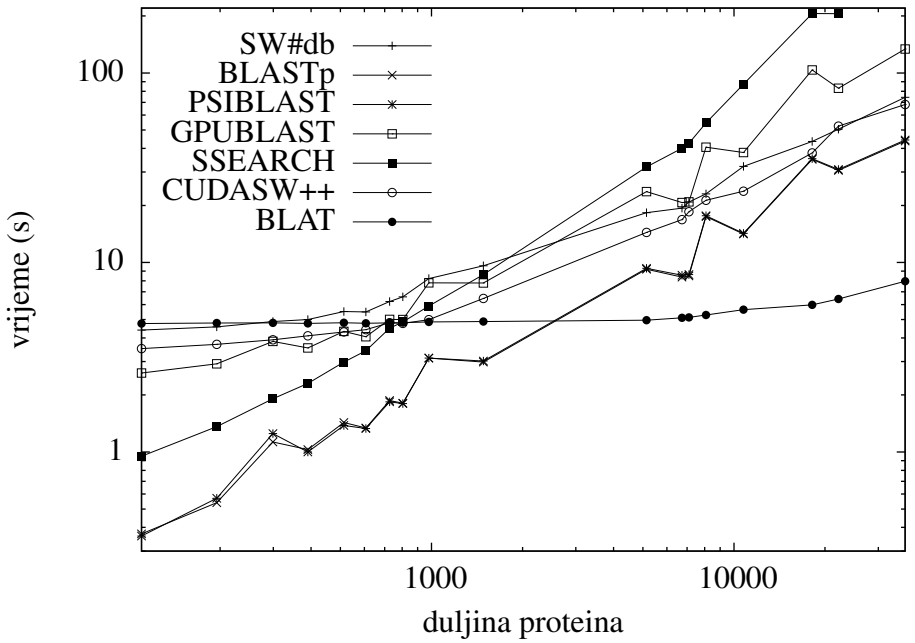
Najvažnija odlika alata za poravnavanje jednog slijeda sa skupom sljedova jest odnos njegove brzine i točnosti. Točnost će biti mjerena pri poravnanju parova skupova sljedova dok su komparativna mjerenja brzina raznih alata za poravnanje jednog slijeda sa skupom sljedova dana u nastavku. Za mjerenje brzine biblioteke SW# koristi se modul SW#db, tj. modul za poravnavanje proteina koristeći Smith-Watermanov algoritam. Tablica 7.2 prikazuje popis korištenih proteina, preuzetih iz Uniprot baze proteina [26], korištenih za mjerenja. Navedeni proteini su u svim testovima poravnavani s cijelom Uniprot bazom potvrđenih proteina.

Tablica 7.8: Usporedba vremena izvođenja različitih algoritama poravnavanja jednog slijeda sa skupom sljedova.

Duljina	SW#db		BLASTp		PSIBLAST		GPUBLAST		SSEARCH		CUDASW++		BLAT	
	t	n	t	n	t	n	t	n	t	n	t	n	t	n
110	4.40s	64	0.37s	40	0.36s	40	2.61s	40	0.95s	42	3.51s	6	4.76s	1
195	4.57s	92	0.54s	24	0.57s	24	2.92s	24	1.36s	20	3.70s	13	4.79s	5
299	4.88s	500	1.13s	500	1.25s	500	3.83s	500	1.91s	500	3.91s	499	4.80s	139
390	5.00s	328	1.03s	257	1.00s	257	3.54s	257	2.30s	260	4.10s	253	4.77s	67
513	5.51s	500	1.43s	411	1.38s	411	4.33s	411	2.98s	425	4.30s	386	4.81s	38
607	5.48s	313	1.34s	262	1.33s	262	4.06s	262	3.42s	270	4.42s	257	4.78s	7
727	6.22s	500	1.87s	500	1.84s	500	5.01s	500	4.50s	500	4.81s	355	4.80s	278
804	6.59s	500	1.81s	377	1.80s	377	5.00s	377	4.86s	383	4.77s	351	4.82s	129
980	8.23s	500	3.13s	30	3.13s	30	7.81s	30	5.89s	500	4.99s	499	4.86s	24
1485	9.60s	500	2.98s	327	3.02s	328	7.80s	327	8.63s	363	6.46s	296	4.88s	7
5141	18.31s	500	9.21s	359	9.31s	359	23.66s	359	31.92s	500	14.40s	395	4.96s	33
6733	19.36s	456	8.37s	128	8.56s	130	20.71s	128	40.09s	140	16.84s	91	5.11s	5
7094	20.85s	500	8.49s	160	8.63s	163	20.87s	160	42.50s	165	18.53s	111	5.13s	62
8081	23.02s	500	17.47s	500	17.63s	500	40.57s	500	54.53s	500	21.28s	499	5.28s	13
10746	32.13s	500	14.14s	96	14.25s	96	38.03s	96	87.23s	500	23.77s	499	5.64s	18
18141	43.54s	500	35.01s	337	35.30s	337	103s	337	206s	500	37.84s	499	5.98s	5
22152	50.48s	500	30.63s	3	30.95s	3	83.00s	3	205s	500	52.46s	499	6.42s	1
36805	74.34s	500	43.75s	50	44.26s	50	133s	50			68.20s	359	7.96s	0

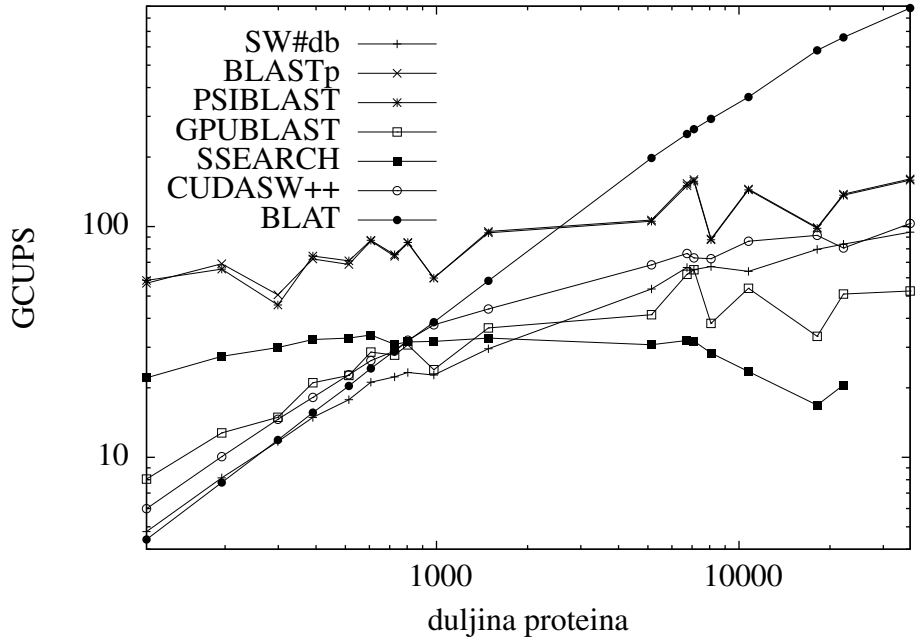
U tablici 7.8 prikazano je vrijeme izvođenja te broj pronađenih poravnanja raznih alata za poravnanje jednog slijeda sa skupom sljedova. Valja napomenuti da svi alati, osim CUDASW++a, vraćaju poravnanja te ocjenjuju poravnanja prema E vrijednosti definiranoj u poglavlju 2.4. Alat CUDASW++ ne vraća poravnanja te ocjenjuje poravnanja direktno kroz njihovu ocjenu. Valja napomenuti da alati BLASTp, PSIBLAST i GPUBLAST troše dodatnih oko minutu vremena za pripremu navedene baze sljedova, dok se baze sljedova kod ostalih alata pripremaju tokom izvođenja. Svi alati, osim alata BLAT koji ne koristi klasično ocjenjivanje, postavljeni su da koriste BLO-

SUM_62 tablicu supstitucija, da je kazna otvaranja procjepa jednaka 10 te da je kazna produljivanja procjepa jednaka 2. Također je za sve alate postavljen maksimalan broj poravnanja na 500. Za sve alate, osim alata CUDASW++, maksimalna E vrijednost postavljena je na 10. Dodatno, alati BLASTp i PSIBLAST postavljeni su da koriste četiri dretve. Sva mjerenja provedena su na konfiguraciji $conf_2$. Iako među sporijima, modul SW#db pronašao je daleko najveći broj poravnanja. Konkretno, našao je 7753 poravnanja, dok je alat BLASTp našao 4361. Sličan alat, CUDASW++, brži je oko 30% gledajući ukupno vrijeme izvršavanja te oko 5% gledajući samo vrijeme ocjenjivanja, jer ne troši dodatno vrijeme na poravnanja što je kod većih proteina značajna količina vremena. Zanimljivo je primijetiti da je alat GPUBLAST, iako iskorištava CUDA kartice, značajno sporiji od originalnog BLASTp algoritma koji koristi četiri dretve. Zanimljivo je da alat SSEARCH nađe mnogo manji broj poravnanja samo zbog drugačijeg računanja E vrijednosti.



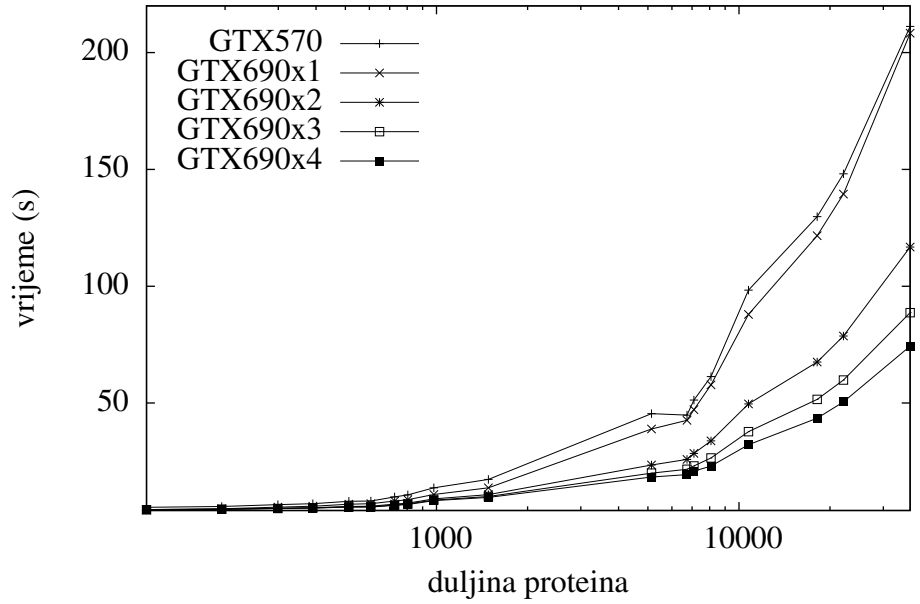
Slika 7.3: Usporedba vremena izvođenja različitih algoritama poravnavanja jednog slijeda sa skupom sljedova.

Grafički prikaz rezultata iz tablice 7.8 dan je slikom 7.3. Možemo primjetiti da je brzinom modul SW#db nešto sporiji, konkretno u prosjeku je peti od sedam mjerenih alata. No, valja primijetiti da nađe malo manje od dva puta više poravnanja od alata BLASTp i gotovo deset puta više poravnanja od najbržeg BLAT alata. Uz usporedbu vremena izvođenja, dana je usporedba postignutih GCUPSa. Navedena usporedba grafički je prizana na slici 7.4. SW#db modul postiže od 5 do 95 GCUPSa.



Slika 7.4: Usporedba GCUPSa različitih algoritama poravnavanja jednog slijeda sa skupom sljedova.

SW#db modul, kao i SW#n modul, prilagođen je izvođenju na višekartičnim CUDA sustavima. Kako bi se prikazala skalabilnost SW#db modula na višekartičnim CUDA sustavima provedena su mjerenja na konfiguraciji $conf_2$ koristeći od jedne do četiri kartice. Rezultati su grafički prizani na slici 7.5. Na istoj slici također je označena brzina izvođenja na konfiguraciji $conf_1$.



Slika 7.5: Usporedba vremena izvođenja SW#db modula na različitim CUDA karticama.

7.3.2. Poravnavanje parova skupova sljedova

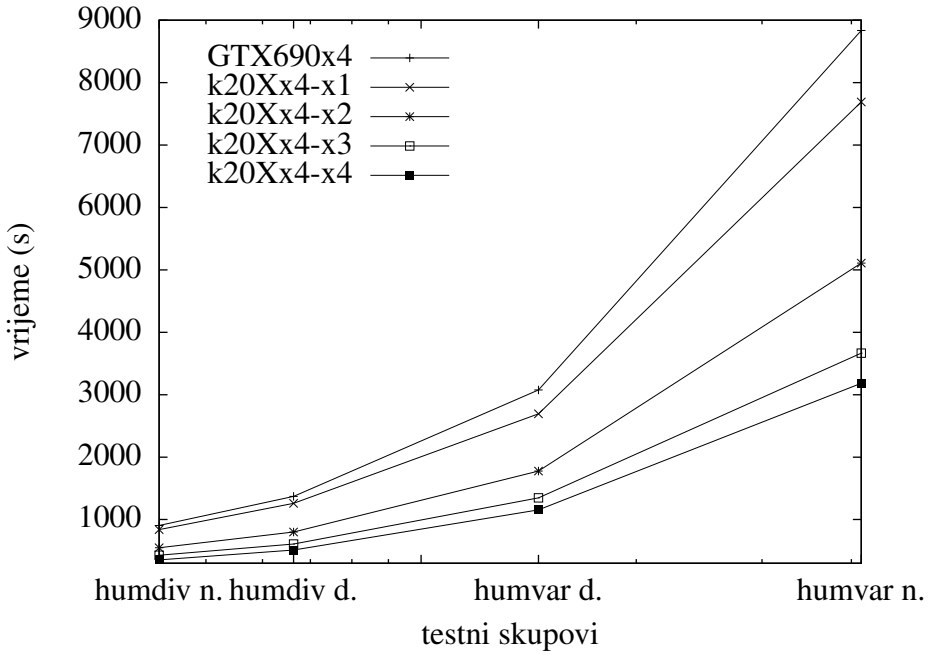
Najvažnija odlika alata za poravnavanje parova skupova sljedova jest odnos njegove brzine i točnosti. Komparativna mjerenja brzina i točnosti raznih alata za poravnanje parova skupova sljedova dana su u nastavku. Za mjerenje brzine biblioteke SW# koriste se moduli SW#db i SW#dbmpi, tj. moduli za poravnavanje proteina koristeći Smith-Watermanov algoritam. Kao testni skupovi koriste se humdiv i humvar baze proteina predstavljene radom [27] te Swiss-prot baza proteina definirana radom [28]. Svaka od humdiv i humvar baze proteina dijeli se na neutral i deleterious podskupove.

Tablica 7.9: Usporedba vremena izvođenja različitih algoritama poravnavanja parova skupova sljedova.

Algoritam	humdiv				humvar			
	neutral		deleterious		neutral		deleterious	
	t	GCUPS	t	GCUPS	t	GCUPS	t	GCUPS
SW#db	905s	47.53	1370s	48.39	8835s	46.85	3077s	47.57
BLASTp	474s	90.72	715s	92.75	4574s	90.49	1613s	90.73
PSIBLAST	594s	72.38	906s	73.14	5846s	70.81	2031s	72.06
GPUBLAST	1240s	34.68	1877s	35.32	11676s	35.45	4162s	35.16
BLAT	20s	2180	27s	2426	136s	3033	55s	2640

U tablici 7.8 prikazano je vrijeme izvođenja te vrijednost GCUPSa raznih alata za poravnanje parova skupova sljedova pri poravnanju neutral i deleterious podskupova humdiv i humvar baza proteina sa Swiss-prot bazom proteina. Navedena mjerenja provedena su na konfiguraciji $conf_2$. Iako je alat BLAT za dva reda veličine brži od ostalih, kasnije će se pokazati njegova točnost izuzetno mala. Postavke algoritama jednake su kao i kod testiranja opisanog u poglavlju 7.3.1. Opet je zanimljivo primijetiti da je GPUBLAST sporiji od BLASTp inačice s četiri pokrenute dretve. Valja napomenuti da je modul SW#db izvođen na četiri CUDA kartice (fizički dvije kartice). Primjećujemo da je alat BLASTp u prosjeku oko 50% brži od modula SW#db dok je alat PSIBLAST u prosjeku oko 40% brži. Zanimljivo je primjetiti visok broj GCUPSa za deterministički algoritam koji postiže modul SW#db.

SW#dbmpi modul, kao i SW#n modul i SW#db modul, prilagođen je izvođenju na višekartičnim CUDA sustavima te dodatno na MPI sustavima. Kako bi se prikazala skalabilnost SW#dbmpi modula na MPI sustavima provedena su mjerenja na konfiguraciji $conf_3$ koristeći od jednog do četiri MPI čvora. Rezultati mjerenja grafički su prikazani na slici 7.6. Dodatno, na istoj slici je prikazano vrijeme izvođenja SW#db modula na konfiguraciji $conf_2$.



Slika 7.6: Usporedba vremena izvođenja SW#dbmpi modula na različitim CUDA karticama s različitim brojem MPI čvorova.

Ocjenjivanje točnosti alata za poravnanje parova skupova sljedova provedeno je po uzoru na mjerenja provedena u radu [29]. Kako je navedeno, humdiv i humvar skupovi sastoje se od neutral i deleterious podskupova. Nad neutral i deleterious podskupovima provodit će se binarna klasifikacija, tj. primjeri će biti označeni kao neutral ili deleterious. Za dobivanje oznake primjera iz izlaza alata za poravnanje parova skupova sljedova koristi se alat SIFT (eng. *Sorting Intolerant From Tolerant*) [27]. Brojevi ispravnih/pogrešnih pozitiva/negativa definira se kao:

- TP - neutral primjer označen kao neutral.
- FP - deleterious primjer označen kao neutral.
- TN - deleterious primjer označen kao deleterious.
- FN - neutral primjer označen kao deleterious.

Vrijednosti dobivene testiranjem prikazane su u tablici 7.10

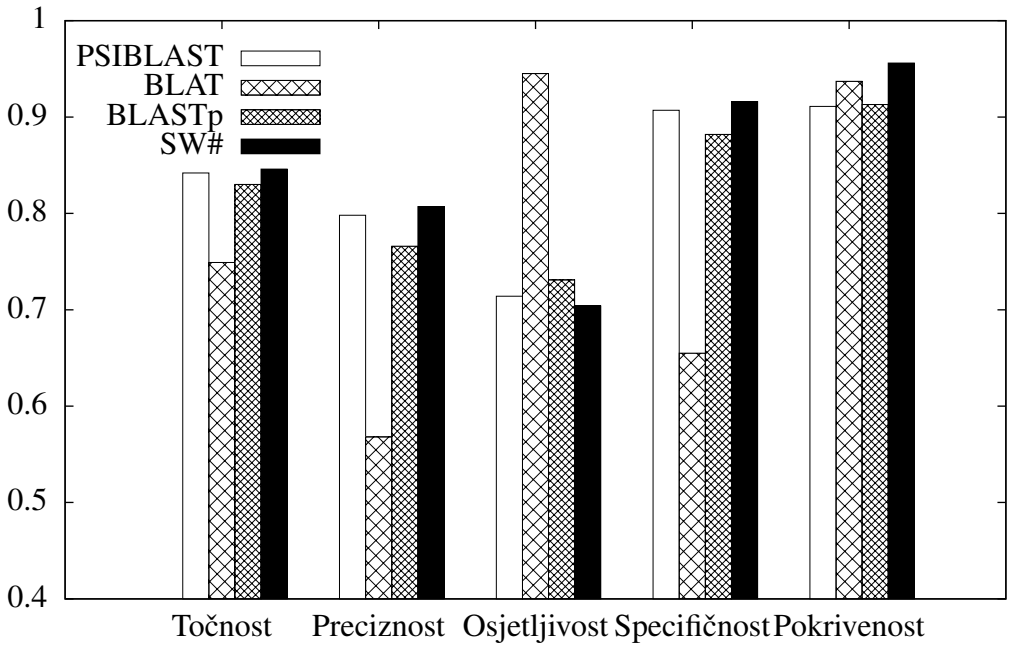
Tablica 7.10: Brojevi ispravnih/pogrešnih pozitiva/negativa dobivene testiranjem humdiv i humvar skupova.

Algoritam	humdiv				humvar			
	TP	FP	TN	FN	TP	FP	TN	FN
BLAT	2613	1984	3764	153	10246	3681	3004	619
BLASTp	2095	639	4789	773	8560	2311	5142	3101
SW#db	2032	486	5308	854	8185	1877	5577	3386

Vrijednosti binarne klasifikacije dobivene testiranjem humdiv skupa prikazane su u tablici 7.11 te su grafički prikazane na slici 7.7. Dodatno, vrijednosti PSIBLAST alata preuzete su iz rada [27]. Zbog većeg broja pronađenih poravnanja, modul SW#db ima nižu osjetljivost od ostalih alata, dok alat BLAT zbog veoma malog broja pronađenih poravnanja ima jako visoku osjetljivost. Prema svim ostalim pokazateljima SW#db modul je bolji od ostalih navedenih algoritama.

Tablica 7.11: Vrijednosti binarne klasifikacije dobivene testiranjem humdiv skupa.

Algoritam	Točnost	Preciznost	Osjetljivost	Specifičnost	Pokrivenost
PSIBLAST	0.842	0.798	0.714	0.907	0.911
BLAT	0.749	0.568	0.945	0.655	0.937
BLASTp	0.830	0.766	0.731	0.882	0.913
SW#	0.846	0.807	0.704	0.916	0.956

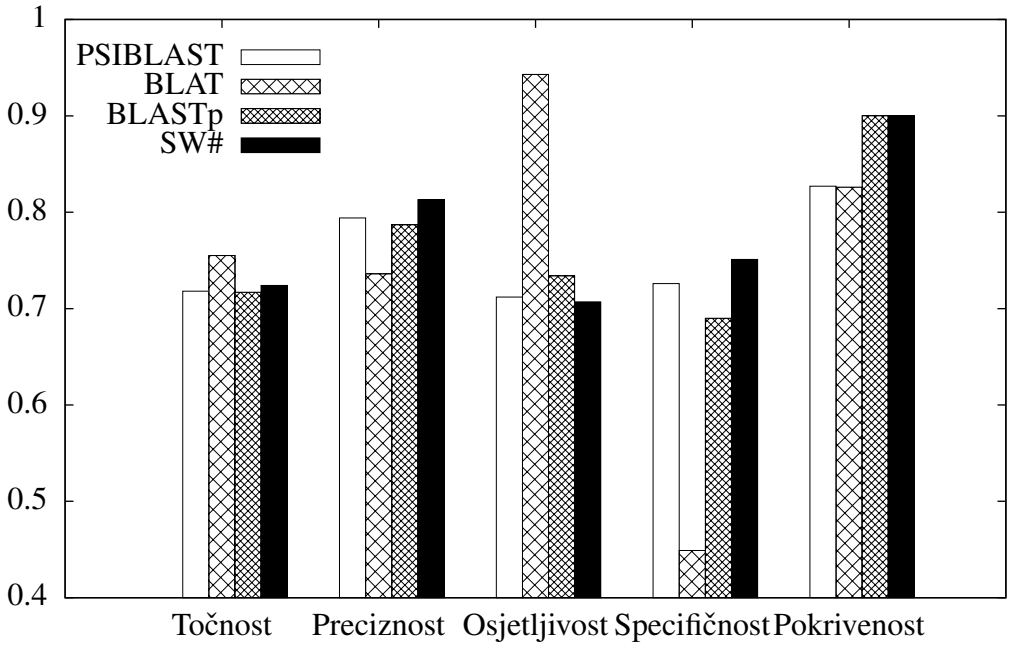


Slika 7.7: Vrijednosti binarne klasifikacije dobivene testiranjem humdiv skupa.

Vrijednosti binarne klasifikacije dobivene testiranjem humvar skupa prikazane su u tablici 7.12 te su grafički prikazane na slici 7.8. Dodatno, vrijednosti PSIBLAST alata preuzete su iz rada [27]. Zbog većeg broja pronađenih poravnanja, modul SW#db ima nižu osjetljivost od ostalih alata, dok alat BLAT zbog veoma malog broja pronađenih poravnanja ima jako visoku osjetljivost te iz istog razloga ima višu točnost. Prema svim ostalim pokazateljima SW#db modul je bolji od ostalih navedenih algoritama.

Tablica 7.12: Vrijednosti binarne klasifikacije dobivene testiranjem humvar skupa.

Algoritam	Točnost	Preciznost	Osjetljivost	Specifičnost	Pokrivenost
PSIBLAST	0.718	0.794	0.712	0.726	0.827
BLAT	0.755	0.736	0.943	0.449	0.826
BLASTp	0.717	0.787	0.734	0.690	0.900
SW#	0.724	0.813	0.707	0.751	0.900



Slika 7.8: Vrijednosti binarne klasifikacije dobivene testiranjem humvar skupa.

8. Zaključak

U radu je predstavljena biblioteka SW# te je objašnjena njezina teorijska podloga. Predstavljene su CUDA inačice Smith-Watermanova i Nedleman-Wunschova algoritma te algoritma poluglobalnog poravnanja. Memorijska složenost navedenih algoritama svedena je na linearnu memorijsku složenost korištenjem Hirschbergova algoritma. Vremenska složenost navedenih algoritama smanjena je upotrebom Ukkonenova algoritma te algoritma dinamičkog odbacivanja elemenata. Dodatno su implemntirane CUDA inačice navedenih algoritama poravnanja sljedova za poravnavanje parova skupova sljedova. Svaki od navedenih CUDA algoritama iskorištava višekartične CUDA sustave. Predstavljeni su SW# moduli. U biblioteci je dostupan modul za poravnavanje parova skupova sljedova koji uz CUDA kartice koristi i MPI tehnologiju. Brzine izvođenja SW# modula uspoređene su, između ostalog, s CPU inačicama navedenih algoritama poravnanja te najpopularnijim determinističkim kao i heurističkim alatima za poravnanje sljedova. Ubrzanja biblioteke SW# u odnosu na CPU inačice sežu i do nekoliko stotina puta. Iako je biblioteka nekoliko puta sporija od najpopularnijih heurističkih algoritama, na nekoliko testnih skupova pokazana je njena viša točnost. Usprkos navedenom, glavna namjena biblioteke SW# je njeno korištenje kao komponente alata za poravnanje sljedova. U tu svrhu, u radu je predstavljeno sučelje biblioteke.

LITERATURA

[1] Richard C Deonier, Simon Tavaré, i Michael S Waterman. *Computational Genome Analysis: An Introduction*, svezak 63. Springer, 2005. ISBN 0387987851. URL http://books.google.com/books?hl=en\&lr=\&id=d-x_hgKqaFcC\&oi=fnd\&pg=PR9\&dq=Computational+Genome+Analysis.+An+Introduction.\&ots=KwXwBYc3Rj\&sig=8ziZvaJkDIXAEUj9XPwKlQbUXJs.

[2] V I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. ISSN 00385689.

[3] S B Needleman i 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, Ožujak 1970. ISSN 0022-2836. URL http://www.ncbi.nlm.nih.gov/pubmed/5420325.

[4] W Gish i SF Altschul. Improved sensitivity of nucleic acid database searches using application-specific scoring matrices. *Methods*, 3(1):66–70, Kolovoz 1991. ISSN 10462023. doi: 10.1016/S1046-2023(05)80165-3. URL http://linkinghub.elsevier.com/retrieve/pii/S1046202305801653http://www.sciencedirect.com/science/article/pii/S1046202305801653.

[5] O Gotoh. An improved algorithm for matching biological sequences., Prosinac 1982. ISSN 0022-2836. URL http://www.ncbi.nlm.nih.gov/pubmed/7166760.

[6] T F Smith i M S Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. URL http://www.ncbi.nlm.nih.gov/pubmed/7265238.

[7] S F Altschul, W Gish, W Miller, E W Myers, i D J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–10, Listopad 1990. ISSN 0022-2836. doi: 10.1016/S0022-2836(05)80360-2. URL http://www.ncbi.nlm.nih.gov/pubmed/2231712.

[8] W. J. Kent. BLAT—The BLAST-Like Alignment Tool. *Genome Research*, 12 (4):656–664, Ožujak 2002. ISSN 1088-9051. doi: 10.1101/gr.229202. URL http://www.genome.org/cgi/doi/10.1101/gr.229202.

[9] SF Altschul i Warren Gish. Local alignment statistics. *Methods in enzymology*, 266:460–480, 1996. URL https://www.cs.umd.edu/class/fall2011/cmsc858s/Local_Alignment_Statistics.pdf.

[10] D S Hirschberg. A Linear Space Algorithm for Computing Maximal Common Subsequences. *Communications of the ACM*, 18(6):341–343, 1975. ISSN 00010782. doi: 10.1145/360825.360861. URL http://portal.acm.org/citation.cfm?doid=360825.360861.

[11] E W Myers i W Miller. Optimal alignments in linear space. *Computer applications in the biosciences: CABIOS*, 4(1):11–17, 1988. ISSN 0266-7061. URL http://www.ncbi.nlm.nih.gov/pubmed/3382986.

[12] E Ukkonen. Algorithms for approximate string matching. *Information and control*, 1985. URL http://www.sciencedirect.com/science/article/pii/S0019995885800462.

[13] E S Ristad i P N Yianilos. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998. ISSN 01628828. doi: 10.1109/34.682181. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=682181.

[14] E Sandes i A de Melo. Retrieving Smith-Waterman Alignments with Optimizations for Megabase Biological Sequences using GPU. stranice 1–14, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6226380.

[15] Ian Buck. GPU Computing with NVIDIA CUDA. *ACM SIGGRAPH 2007 courses*, stranica 6, 2007. doi: 10.1145/1281500.1281647. URL http://dl.acm.org/citation.cfm?id=1281647.

[16] JJ Dongarra, R Hempel, AJG Hey, i DW Walker. A proposal for a user-level, message passing interface in a distributed memory environment. *Chemistry & ...*, 1993. URL http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstracthttp://www.osti.gov/energycitations/product.biblio.jsp?osti_id=10157109.

[17] Mengyao Zhao, WP Lee, i GT Marth. SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications. *arXiv preprint arXiv:1208.6350*, 00(00):1–3, 2012. URL http://arxiv.org/abs/1208.6350.

[18] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, i D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–402, Rujan 1997. ISSN 0305-1048. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=146917\&tool=pmcentrez\&rendertype=abstract.

[19] Panagiotis D Vouzis i Nikolaos V Sahinidis. GPU-BLAST: using graphics processors to accelerate protein sequence alignment. *Bioinformatics (Oxford, England)*, 27(2):182–8, Siječanj 2011. ISSN 1367-4811. doi: 10.1093/bioinformatics/btq644. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3018811\&tool=pmcentrez\&rendertype=abstract.

[20] David W Mount. Using a FASTA Sequence Database Similarity Search. *CSH protocols*, 2007(14):pdb.top16, 2007. URL http://www.cshprotocols.org/cgi/doi/10.1101/pdb.top16.

[21] Yongchao Liu, Bertil Schmidt, i Douglas L Maskell. CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. 2010.

[22] Peiheng Zhang, Guangming Tan, i Guang R. Gao. Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform. U *Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications held in conjunction with SC07 - HPRCTA '07*, stranica 39, New York, New York, USA, 2007. ACM Press. ISBN

9781595938947. doi: 10.1145/1328554.1328565. URL http://portal.acm.org/citation.cfm?doid=1328554.1328565.

[23] Dennis a Benson, Ilene Karsch-Mizrachi, Karen Clark, David J Lipman, James Ostell, i Eric W Sayers. GenBank. *Nucleic acids research*, 40(Database issue): D48–53, Siječanj 2012. ISSN 1362-4962. doi: 10.1093/nar/gkr1202. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3245039\&tool=pmcentrez\&rendertype=abstract.

[24] Matija Korpar. Implementacija Smith Waterman algoritma koristeći grafičke kartice s CUDA arhitekturom. 2011. URL http://bib.irb.hr/prikazi-rad?lang=en\&rad=545011.

[25] AM Hosny i HA Shedeed. An efficient solution for aligning huge DNA sequences. *International Journal of Computer Applications*, 32(6):1–8, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6141059.

[26] The Uniprot Consortium. The Universal Protein Resource (UniProt) in 2010. *Nucleic Acids Research*, 38(Database issue):D142–D148, 2010. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2808944\&tool=pmcentrez\&rendertype=abstract.

[27] Ngak-Leng Sim, Prateek Kumar, Jing Hu, Steven Henikoff, Georg Schneider, i Pauline C Ng. SIFT web server: predicting effects of amino acid substitutions on proteins. *Nucleic acids research*, 40(Web Server issue): W452–7, Srpanj 2012. ISSN 1362-4962. doi: 10.1093/nar/gks539. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3394338\&tool=pmcentrez\&rendertype=abstract.

[28] A Bairoch i B Boeckmann. The SWISS-PROT protein sequence data bank: current status. *Nucleic Acids Research*, 22(17):3578–3580, 1994. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=308324\&tool=pmcentrez\&rendertype=abstract.

[29] Robert Vaser. Evaluacija aplikacija za pretraživanje baze proteinskih sljedova. 2013.

SW# - Biblioteka za poravnanje sljedova korištenjem grafičkih procesora

Sažetak

Problem poravnanja sljedova jedan je od najstarijih problema u bionformatici. Iako je prošlo nekoliko desetljeća, problem poravnanja je još uvijek aktualan. Trenutna rješenja su kompromisi između potrošnje vremena, memorije i točnosti. Smith-Watermanov i Needleman-Wunschov algoritam najpoznatiji su deterministički algoritmi poravnanja sljedova. Unatoč svojoj točnosti, rijetko se koriste samostalno zbog svoje velike memorijske i vremenske potrošnje. Ipak, često se koriste kao komponente različitih heurističkih alata. U radu je predstavljena biblioteka SW#. Biblioteka nudi CUDA implentacije memorijski učinkovitih inačica determinističkih algoritama poravnanja, čija ubrzanja sežu i do nekoliko stotina puta u odnosu na CPU inačice, kao i sučelje za korištenje s drugim aplikacijama.

Ključne riječi: SW#, Smith-Waterman, CUDA, paralelizacija, bioinformatika, poravnavanje sljedova

SW# - Sequence alignment library utilizing graphics processing units

Abstract

Sequence alignment is one of the oldest and the most famous problems in bioinformatics. Even after a few decades, this problem is still relevant. Current solutions are trade-offs between execution time, memory consumption and accuracy. Smith-Waterman and Needleman-Wunsch algorithms are the most famous deterministic sequence alignment algorithms. Despite their accuracy, they are rearly used independently because of their high time and memory consumption. However, they are commonly used as components of different heuristic tools. In this thesis SW# library is introduced. The library offers CUDA implementations of memory efficient versions of different deterministic sequence alignment algorithms, whose speedups reach several hundred times the speed of the CPU versions, as well as the interface for usage with other applications.

Keywords: SW#, Smith-Waterman, CUDA, parallelization, bioinformatics, sequence alignment