SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3253

GPU implementacija vremenski i memorijski učinkovitoga paralelnog algoritma za poravnanje slijedova

Marija Mikulić

Zagreb, lipanj 2013.

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

Hvala Miletu što me motivirao da se posvetim računarstvu, na ukazanom povjerenju, na prenesenom znanju i ponajviše na strpljenju.

SADRŽAJ

Popis tablica **v**

Popis slika **vi**

1. Uvod **1**

2. Poravnavanje struktura **2**

3. Smith-Watermanov algoritam **3**

3.1. Needleman-Wunschov algoritam 3

3.1.1. Algoritam 3

3.1.2. Primjer 4

3.2. Smith-Watermanov algoritam 5

3.2.1. Kažnjavanje praznina 6

4. Predmetačni račun **8**

4.1. Primjer 8

4.2. Algoritam 9

4.2.1. Sekvencijalni algoritam 9

4.2.2. Paralelni algoritam 9

5. CUDA tehnologija **12**

6. Implementacija **14**

7. Rezultati **17**

8. Zaključak **18**

Literatura **19**

POPIS TABLICA

3.1. Primjer matrice sličnosti 4
3.2. Matrica **F** nakon provedbe Needleman-Wunschovog algoritma 5
3.3. Globalno poravnati slijedovi 5
3.4. Matrica **F** nakon provedbe Smith-Watermanovog algoritma 6
3.5. Najbolje lokalno poravnanje 6

POPIS SLIKA

2.1. Primjer globalnog i lokalnog poravnanja dvaju nizova 2

4.1. Algoritam za predmetačni zbroj 9
4.2. Faza redukcije 10
4.3. Algoritam faze redukcije 10
4.4. Faza predtraženja 11
4.5. Algoritam za fazu predtraženja 11

1. Uvod

Od svojih začetaka u 1940-ima, računala su postupno postala dio naše svakodnevice. To je područje ljudskog djelovanja koje se najbrže mijenja i evoluira. Ta evolucija se događa velikim dijelom zbog toga što se računala sve više i na sve više različitih načina primjenjuju u raznim područjima ljudskog djelovanja.

Jedno od tih područja je i biologija. U biologiji su za neke probleme, poput poravnavanja proteina i DNA, računala ključni alat za dolaženje do novih spoznaja. Iz te neodvojive veze nastalo je cijelo područje istraživanja koje nazivamo bioinformatika.

Biolozi ulažu velike napore u proučavanje našeg građevnog materijala, proteina i molekula DNA koji nas tvore. Proučavanje ovih molekula pokazalo se kao vrlo bitno i potencijalno vrlo korisno.

Primjerice, ako bismo mogli manipulirati svojom DNA, mogli bismo ukloniti iz svojih gena mnoge bolesti i sindrome koji danas, nažalost, pogađaju mnoge ljude.

Naravno, to otvara vrata i mnogim drugim stvarima te je predmet brojnih etičkih i filozofskih rasprava, ali prvotni cilj znanosti jest nastojati objasniti kako funkcionira svijet oko nas. Stoga nam je vrlo zanimljivo proučavati kako, u svojoj srži, funkcioniramo mi sami.

Iz te znatiželje i iz činjenice da su u biološkim molekulama spremljeni gigabajti informacija, pojavila se potreba za razvijanjem algoritama kojima bi se moglo učinkovito obrađivati tolike količine informacija.

Konkretan problem kojim se bavi ovaj završni rad jest poravnavanje jednog proteina s listom proteina. Odnosno, zanima nas koliko ima sličnosti između nekog proteina i N drugih proteina. Ovaj rad opisuje rješenje toga problema algoritmom koji koristi predmetačni račun i Smith-Watermanov algoritam.

Poglavlje 2 opisuje općenito problem poravnavanja struktura. U poglavlju 3 izložen je Smith-Watermanov algoritam, a u poglavlju 4 predmetačni račun. Poglavlje 5 opisuje tehnologiju CUDA, a poglavlje 6 specifičnosti implementacije, dok se u poglavlju 7 izlažu dobiveni rezultati. Poglavlje 8 posvećeno je zaključku rada.

2. Poravnavanje struktura

Kao jedan od ključnih problema bioinformatike nametnuo se problem poravnanja molekula proteina i DNA. Svrha tog poravnanja jest ustanoviti područja sličnosti između tih struktura, jer ona mogu indicirati funkcijsku, strukturalnu ili evolucijsku vezu između njih.[4]

Za potrebe informatike, problem poravnanja struktura se može svesti na problem poravnavanja slijedova znakova. Za ilustraciju, zamislimo da se slijedovi koje poravnavamo stave jedan ispod drugog. U tom slučaju, cilj je tako postaviti znakove nizova da oni izgledaju najsličnije moguće.

Razlikujemo dvije vrste poravnavanja: globalno i lokalno.

Globalno poravnavanje je postupak kojim se jedan slijed preslikava na drugi od početka do kraja. Pri tome je dozvoljeno u bilo koji slijed umetnuti prazninu na proizvoljno mjesto, osim na prvo. Ukoliko je jedan slijed znatno dulji od drugog, ovakvo poravnanje može za posljedicu imati umetanje velikog broja praznina u kraći slijed.

Za razliku od globalnog, lokalno poravnanje ne vodi računa o tome da se iskoriste svi znakovi oba slijeda. Ovdje je cilj naći podslijedove zadanih slijedova koji su međusobno najsličniji. S obzirom na to da se ne moraju iskoristiti svi znakovi bilo kojeg slijeda, nije dozvoljeno da optimalno poravnanje počinje ili završava prazninom.

```
Globalno        FTFTALILLAVAV
                F--TAL-LLA-AV

Lokalno         FTFTALILL-AVAV
                --FTAL-LLAAV--
```

Slika 2.1: Primjer globalnog i lokalnog poravnanja dvaju nizova

3. Smith-Watermanov algoritam

Smith-Watermanov algoritam[7] među najvažnijim je algoritmima za lokalno poravnanje slijedova. Budući da je algoritam nastao iz Needleman-Wunschovog[5] algoritma za globalno poravnanje slijedova, prvo ćemo razmotriti taj algoritam pa predstaviti potrebne modifikacije kako bismo dobili Smith-Watermanov algoritam za lokalno poravnanje slijedova.

3.1. Needleman-Wunschov algoritam

Poznat još i kao algoritam optimalnog poravnanja, Needleman-Wunschov algoritam koristi se za globalno poravnavanje slijedova. To je prvi algoritam koji je primijenio dinamičko programiranje na problem poravnanja slijedova u biologiji.

3.1.1. Algoritam

Cilj Needleman-Wunschovog algoritma jest pronaći optimalno poravnanje neka dva slijeda A i B, odnosno ono poravnanje koje u obzir uzima sve znakove oba slijeda i usto ima najviši broj bodova s obzirom na sva moguća poravnanja (dozvoljeno je postojanje više rješenja s istim brojem bodova).

Označimo s *n* duljinu slijeda A i s *m* duljinu slijeda B. Za potrebe pronalaska optimalnog globalnog rješenja potrebno je alocirati dvodimenzionalnu matricu dimenzija $(n + 1) \times (m + 1)$. Nulti redak i stupac služe za inicijalizaciju matrice. Gledajući od indeksa 1, svaki redak matrice predstavlja znak u slijedu A, a svaki stupac predstavlja znak u slijedu B. Element matrice F_{ij}, gdje je i indeks retka, a j indeks stupca, predstavlja broj bodova za poravnanje prvih i znakova slijeda A i prvih j znakova slijeda B.

Kako bismo ocijenili poravnanje neka dva znaka, definiramo *matricu sličnosti* **S** nad skupom dozvoljenih znakova Σ. Element matrice S_{σ_i,σ_j} daje ocjenu poravnanja znaka σ_i sa znakom σ_j. Dodatno, potrebno je ocijeniti umetanje praznina u bilo koji

od dva slijeda. Za te potrebe definira se konstanta d kojom se kažnjava svaka praznina.

Pošto tražimo optimalno globalno poravnanje, želimo biti sigurni da će poravnanje krenuti od prvog znaka slijeda A i prvog znaka slijeda B. Zbog toga se postavljaju sljedeći početni uvjeti:

$$\begin{aligned} F_{0j} &= d \times j, & j &= 0...m-1 \\ F_{i0} &= d \times i, & i &= 0...n-1 \end{aligned}$$

Svi ostali elementi matrice računaju se prema formuli:

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + S(A_i, B_j) \\ F_{i,j-1} + d \\ F_{i-1,j} + d \end{cases} \tag{3.1}$$

Nakon što su izračunati svi elementi matrice, element F_{nm} sadrži broj bodova optimalnog globalnog poravnanja.

3.1.2. Primjer

Za ilustraciju rada algoritma poslužit će primjer poravnanja DNA slijedova *AGACTAGTTAC* i *CGAGACGT*.

Nad abecedom (koja je u ovom slučaju {A, C, G, T}) definira se sljedeća matrica sličnosti **S**: Usto, neka je vrijednost funkcije kazne $d = -5$. Nakon izvođenja algo-

	A	**G**	**C**	**T**
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

Tablica 3.1: Primjer matrice sličnosti

ritma, matrica **F** sadrži sljedeće vrijednosti:

Vidimo da optimalno poravnanje ima broj bodova 16 i da su slijedovi poravnati na sljedeći način:

		A	G	A	C	T	A	G	T	T	A	C
	0	-5	-10	-15	-20	-25	-30	-35	-40	-45	-50	-55
C	-5	-3	-8	-13	-6	-11	-16	-21	-26	-31	-36	-41
G	-10	-6	4	-1	-6	-9	-12	-9	-14	-19	-24	-29
A	-15	0	-1	14	9	4	1	-4	-9	-14	-9	-14
G	-20	-5	7	9	9	6	3	8	3	-2	-7	-12
A	-25	-10	2	17	12	7	16	11	6	1	8	3
C	-30	-15	-3	12	26	21	16	11	11	6	3	17
G	-35	-20	-8	7	21	23	20	23	18	13	8	12
T	-40	-25	-13	2	16	29	24	19	31	26	21	16

Tablica 3.2: Matrica **F** nakon provedbe Needleman-Wunschovog algoritma

A	G	A	-	-	C	T	A	G	T	T	A	C
C	G	A	G	A	C	G	-	-	T	-	-	-

Tablica 3.3: Globalno poravnati slijedovi

3.2. Smith-Watermanov algoritam

Za razliku od Needleman-Wunschovog, Smith-Watermanov algoritam koristi se za lokalno poravnanje slijedova. Ideja algoritma je ista, ali su potrebne modifikacije.

Ostat ćemo pri istim oznakama: razmatramo slijedove A (duljine n) i B (duljine m) definirane nad abecedom Σ. S d ćemo označavati kaznene bodove za umetanje praznine, a poravnanje neka dva elementa abecede Σ definirano je matricom sličnosti **S**. Potrebno je popuniti matricu **F**.

Pošto poravnanje ne mora početi s prvim znakovima oba niza, kao donja granica poravnanja postavi se vrijednost 0. Ovo omogućuje da poravnanje prekinemo kad postane nepovoljno, odnosno kad bi ocjena poravnanja postala negativna, i krenemo razmatrati poravnanje koje počinje nekim drugim znakom. S obzirom na ovaj zahtjev, početni uvjeti su sljedeći:

$$F_{0j} = 0, \qquad j = 0...m-1$$
$$F_{i0} = 0, \qquad i = 0...n-1$$

a algoritam je modificiran tako da ne dopušta negativno poravnanje:

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + S(A_i, B_j) \\ F_{i,j-1} + d \\ F_{i-1,j} + d \\ 0 \end{cases} \tag{3.2}$$

		A	G	A	C	T	A	G	T	T	A	C
	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	9	4	0	0	0	0	0	9
G	0	0	7	2	4	6	3	7	2	0	0	4
A	0	10	5	17	12	7	16	11	6	1	10	5
G	0	5	17	12	12	9	11	23	18	13	8	5
A	0	10	12	27	22	17	19	18	19	14	23	18
C	0	5	7	22	36	31	26	21	18	19	18	32
G	0	0	12	17	31	33	30	33	28	23	18	27
T	0	0	7	12	26	39	34	29	41	36	31	26

Tablica 3.4: Matrica **F** nakon provedbe Smith-Watermanovog algoritma

		A	G	A	C	T	-	A	G	T	T	A	C
C	G	A	G	A	C	-	G	-	-	T			

Tablica 3.5: Najbolje lokalno poravnanje

3.2.1. Kažnjavanje praznina

Opisani algoritam kao funkciju kazne koristi *linearnu* funkciju kazne - svaka praznina se kažnjava s d bodova, što znači da je praznina duljine k ocijenjena s $k \cdot d$. Međutim, kako bi bio kvalitetan za poravnanje bioloških struktura (proteina i nizova DNA), ovaj algoritam potrebno je dodatno modificirati.

Naime, praznine koje se umeću u slijedove predstavljaju mutacije, a veća je vjerojatnost da se dogodila jedna mutacija nad nekim podslijedom nego nekoliko mutacija duljine *1* zaredom.

Empirijski dokazana formula za ocjenjivanje procjepa jest parabola. Stoga se definira afina funkcija kazne, koja za produljenje postojeće praznine dodjeljuje manju kaznu nego za otvaranje nove. Označimo s o kaznu za otvaranje praznine i s e kaznu za produljenje postojeće praznine. Tada je za prazninu duljine n kazna $o + (n-1)e$. Uporabom ovakve funkcije dobiju se najbolji praktični rezultati.[8]

S obzirom na ovu promjenu, potrebno je modificirati i sam algoritam, što nije trivijalno. Označimo s A slijed koji ćemo postaviti kao referentni za retke matrice, a s B slijed koji je referentan za stupce matrice **F**.

Kako bismo uspješno pratili sve moguće situacije, potrebne su nam tri matrice: **N** (*no gaps*), koja pokušava poravnati iduća dva elementa slijedova, **H** (*horizontal gap*), koja pokušava idući element slijeda B poravnati s prazninom i **V**(*vertical gap*), koja pokušava idući element slijeda A poravnati s prazninom.

Početni uvjeti za sve tri matrice su:

$$\begin{aligned} N_{0j} = 0, H_{0j} = 0, V_{0j} = 0, \qquad & j = 0...m-1 \\ N_{i0} = 0, H_{i0} = 0, V_{i0} = 0, \qquad & i = 0...n-1 \end{aligned}$$

a ostali elementi matrica računaju se prema sljedećim formulama:

$$H_{i,j} = \max \begin{cases} N_{i,j-1} + o \\ H_{i,j-1} + e \\ 0 \end{cases} \tag{3.3}$$

$$V_{i,j} = \max \begin{cases} N_{i-1,j} + o \\ V_{i-1,j} + e \\ 0 \end{cases} \tag{3.4}$$

$$N_{i,j} = \max \begin{cases} N_{i-1,j-1} + S(A_i, B_j) \\ H_{i,j} \\ V_{i,j} \\ 0 \end{cases} \tag{3.5}$$

4. Predmetačni račun

Problem predmetačnog računa definira se pomoću niza A = {a_0, a_1, a_2, ..., a_{n-1}} i općenitog binarnog operatora $\oplus$. Cilj je generirati niz B = {0, $0 \oplus a_0$, $0 \oplus a_0 \oplus a_1$, ..., $0 \oplus a_0 \oplus ... \oplus a_{n-2}$}.

Ovaj se problem pojavljuje često kao usko grlo u raznovrsnim problemima. Neki od njih su:

- leksička usporedba nizova znakova (npr. određivanje da niz "strategija" dolazi ispred niza "strateški" u rječniku)
- paralelno ostvarenje *radix-sort* i *quick-sort* algoritama
- određivanje vidljivosti točaka u 3D krajoliku – uz operator *max*
- zbrajanje s brojevima višestruke (proizvoljne) preciznosti
- alokacija procesora/memorije
- pretraživanje regularnih izraza (npr. u implementaciji *grep* naredbe u *UNIX*-u)
- izvedba nekih operacija nad stablima (npr. dubina svakog čvora u stablu)

4.1. Primjer

Za ilustraciju problema, pogledajmo slijedeći primjer:

Zadan je niz A:

indeks	0	1	2	3	4	5	6	7
vrijednost	5	3	-8	4	12	6	-9	1

Ako je binarni operator "+", tada je cilj dobiti niz B:

indeks	0	1	2	3	4	5	6	7
vrijednost	0	5	8	0	4	16	22	13

Gore navedeni primjer opisuje isključivi predmetačni račun. To znači da za element b_i ($i = 0...n$) niza B u obzir uzimamo elemente niza A čiji je indeks manji od i, a b_0 popunimo neutralnim elementom (primjerice, za zbrajanje je to 0).

Druga vrsta predmetačnog računa jest uključivi predmetačni račun, koji za izračun elementa b_i koristi sve elemente od a_0 do a_i, uključivo. Za gore navedeni primjer, niz B bi izgledao ovako:

indeks	0	1	2	3	4	5	6	7
vrijednost	5	8	0	4	16	22	13	14

Isključivi predmetačni račun se jednostavno generira iz uključivog tako da se svi elementi pomaknu za jedno mjesto udesno, a kao nulti element se doda neutralni element.

4.2. Algoritam

Predmetačni račun je jedan od algoritama koji se čine inherentno sekvencijalnima, ali za koje postoji učinkovita paralelna implementacija.

4.2.1. Sekvencijalni algoritam

Sekvencijalna implementacija ovog algoritma je trivijalna i svodi se na dinamičko programiranje. Verzija algoritma sa binarnim operatorom "+" (predmetačni zbroj) dana je u ispisu:

```
1 b[0] := 0
2 za svaki k od 1 do n-1
3   b[k] := a[k-1] + b[k-1]
```

Slika 4.1: Algoritam za predmetačni zbroj

4.2.2. Paralelni algoritam

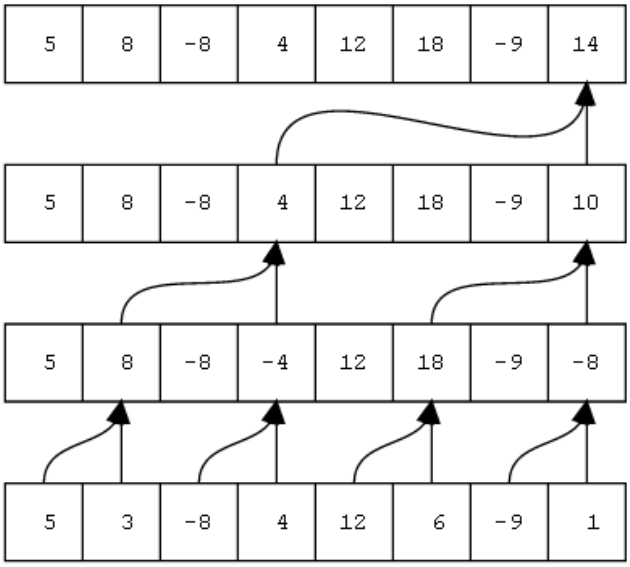
Učinkovita paralelna implementacija predmetačnog računa temelji se na radu Blellocha[3] i koristi balansirano stablo. Ideja je izgraditi balansirano binarno stablo nad ulaznim podacima i zatim njegovim obilaskom do listova do korijena pa od korijena do listova napraviti račun.

Binarno stablo s n listova ima dubinu $d = log_2 n$, a svaka dubina ima 2^d čvorova. Ako izvodimo jedan izračun po čvoru, u jednom obilasku stabla izvodimo $O(n)$ izračuna. Ovo binarno stablo nije fizički implementirano, nego je koncept koji koristimo kako bismo odredili što je zadatak svake dretve u obilasku stabla, a svi izračuni izvode se nad ulaznim nizom podataka.

Izračun se izvodi u dvije faze: reduciranje i predtraženje.

U fazi reduciranja, stablo se obilazi od listova do korijena i izvode se djelomični izračuni, te na kraju ove faze korijen stabla sadrži izračun za sve čvorove u stablu.

Slika i ispis predstavljaju fazu reduciranja na gore navedenom primjeru.



Slika 4.2: Faza redukcije

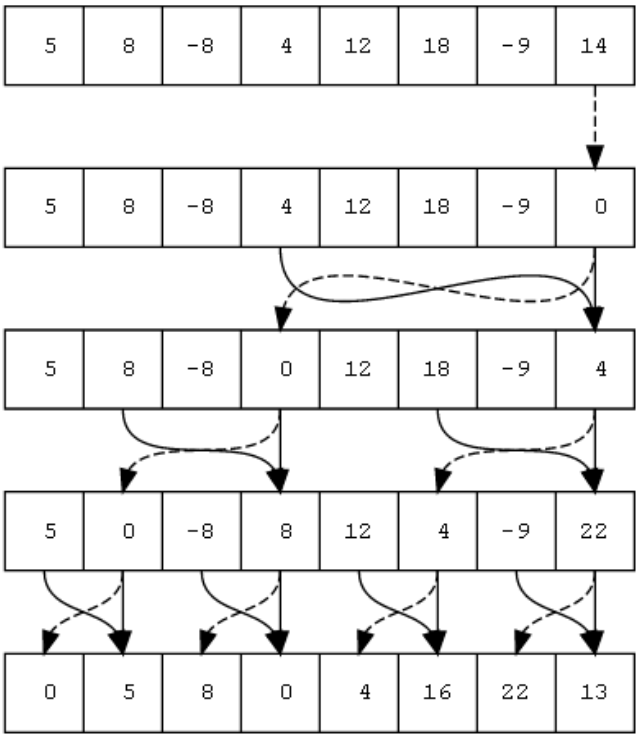
```
1 za svaki d od 0 do log2n - 1:
2   za svaki k od 0 do n - 1 s korakom 2^(d+1) u paraleli:
3     a[k + 2^(d+1) - 1] = a[k + 2^d - 1] + a[k + 2^(d+1) - 1]
```

Slika 4.3: Algoritam faze redukcije

U fazi predtraženja stablo se obilazi od korijena do listova i koriste se djelomični izračuni iz faze reduciranja kako bismo izveli predmetačni račun za svaki element.

U korijen stabla se umetne neutralni element. Zatim, u svakom koraku čvor trenutne razine svom lijevom djetetu preda svoju vrijednost, a zbroj svoje vrijednosti i prethodne vrijednosti lijevog djeteta preda desnom djetetu.

Slika i ispis predstavljaju fazu predtraženja na gore navedenom primjeru.



Slika 4.4: Faza predtraženja

```
1  a[n-1] = 0
2  za svaki d od log2n - 1 do 0:
3    za svaki k od 0 do n - 1 s korakom 2^d + 1:
4      temp = a[k + 2^d - 1]
5      a[k + 2^d - 1] = a[k + 2^(d+1) - 1]
6      a[k + 2^(d+1) - 1] = temp + a[k + 2^(d+1) - 1]
```

Slika 4.5: Algoritam za fazu predtraženja

5. CUDA tehnologija

Grafičke kartice su prvotno dizajnirane kao grafički ubrzivači i podržavale su cjevovode točno određenih funkcija. No, u 1990-im godinama doživjele su dramatičan razvoj i postajale sve više programabilne. NVIDIA, tvrtka koja je i skovala termin GPU (*graphics processing unit - grafička procesna jedinica*), svoj je prvi GPU proizvela 1999. Od tad su moć te tehnologije, osim proizvođača računalnih igara i umjetnika, prepoznali i mnogi istraživači te ju počeli koristiti za svoje potrebe.

Međutim, u to doba je bilo potrebno "prevariti" grafičku karticu koju se htjelo iskoristiti, to jest, trebalo podatke pretočiti u probleme koji se mogu prikazati trokutima i poligonima. Kako bi se to izbjeglo, razvio se GPGPU (*general purpose graphics processing unit - grafička procesna jedinica opće namjene*).

Uskoro su se pojavile i prilagodbe općih programskih jezika za novonastalu tehnologiju.

CUDA tehnologija (punog naziva *Compute Unified Device Architecture - računski ujedinjena arhitektura uređaja*) je programski model i platforma za paralelno programiranje koju je razvila tvrtka NVIDIA i implementirala na svojim grafičkim procesnim jedinicama.[1]

Prva verzija CUDA-inog SDK-a (engl. *software development kit* - oprema za razvoj programske podrške) izdana je 15. veljače 2007. Ona je omogućila korištenje resursa grafičkih kartica jedino u programskom jeziku C. Od tada do danas izdane su još dvije velike iteracije i nekoliko manjih te danas možemo koristiti i podskup jezika C++.[6]

Za shvaćanje načina rada grafičke procesne jedinice s *CUDA* arhitekturom, moramo razumjeti sljedeće pojmove[6]:

- *host* tj. domaćin - centralna procesna jedinica računala (*CPU*)
- *device* tj. uređaj - grafička procesna jedinica
- *thread* tj. dretva - najmanja jedinica izvršavanja, sve naredbe unutar dretve izvršavaju se slijedno
- *block* tj. blok - skup dretvi koje se izvršavaju paralelno

– *grid* tj. mreža - skup blokova koji se izvršavaju paralelno
– *warp* tj. osnova - skup dretvi unutar bloka koje se izvršavaju istodobno i za koje je dozvoljeno pretpostaviti da se u svakom trenutku nalaze na istom mjestu u kodu (stoga ih nije potrebno ručno sinkronizirati)

6. Implementacija

Ovaj rad modifikacija je rada Alurua i ostalih[2]. Oni su pokazali kako se predmetačni račun može koristiti za ubrzanje postupka poravnanja dvaju bioloških slijedova. U svome radu opisuju postupak za globalno poravnanje slijedova.

Inovativnost ovog rada jest u tome da se jedan protein poravnava s N proteina. Označimo protein koji poravnavamo s A, a listu proteina na koji poravnavamo protein A s L. Cilj je naći optimalno lokalno poravnanje između A i bilo kojeg proteina L[i] ($i = 0...N-1$).

Svi elementi L[i] ($i = 0...N-1$) konkatenirani su u jedan niz (nazovimo i njega L) i međusobno odvojeni znakom za prekid. Znak za prekid se koristi kako bismo mogli prekinuti pratiti trenutno poravnanje i krenuti s novim.

Kako bi se ocijenilo poravnanje neka dva elementa a i l abecede nad kojom su definirani slijedovi (označimo ju sa Σ), uvodi se jednostavna funkcija:

$$f(a,l) = \max \begin{cases} 1 & a = l, \quad a,l \in \Sigma \\ 0 & a \neq l, \quad a,l \in \Sigma \end{cases} \tag{6.1}$$

Kao funkcija kažnjavanja procjepa uvodi se afina funkcija kojom se otvaranje procjepa kažnjava s o, a produljenje postojećeg procjepa s e.

Kao što je prethodno opisano, koriste se tri matrice: **N**, **H** i **V**. Svaka matrica je dimenzija $(n+1) \times (m+1)$, gdje je n duljina niza A, a m duljina niza L. Početni uvjeti u matricama postavljaju se na sljedeći način:

$$\begin{aligned} N_{0j} = 0, H_{0j} = 0, V_{0j} = 0, \qquad & j = 0...m \\ N_{i0} = 0, H_{i0} = 0, V_{i0} = 0, \qquad & i = 0...n \end{aligned}$$

a preostali elementi matrica ($i = 1...n, j = 1...m$) pomoću formula:

$$N_{i,j} = f(a_i, l_j) + \max \begin{cases} N_{i-1,j-1} \\ H_{i-1,j-1} \\ V_{i-1,j-1} \\ 0 \end{cases} \tag{6.2}$$

$$H_{i,j} = \max \begin{cases} N_{i,j-1} + o \\ H_{i,j-1} + e \\ V_{i,j-1} + o \\ 0 \end{cases} \tag{6.3}$$

$$V_{i,j} = \max \begin{cases} N_{i-1,j} + o \\ H_{i-1,j} + o \\ V_{i-1,j} + e \\ 0 \end{cases} \tag{6.4}$$

Pri izvođenju ovakvog algoritma na grafičkim procesnim jedinicama, često se primjenjuje popunjavanje matrice po sporednim dijagonalama, jer se elementi potrebni za popuniti neki element nalaze na prethodnoj i trenutnoj dijagonali. Međutim, s obzirom na to da veličine dijagonala variraju, ovakav pristup ima za posljedicu prazan rad dijela jedinice na kraćim dijagonalama.

Stoga se u ovom radu elementi matrica izračunavaju red po red. Vidimo da to ne predstavlja nikakav problem pri izračunu elemenata matrica **N** i **V**, budući da se potrebne informacije nalaze u prethodnom redu. No, ne možemo jednostavno paralelno izračunati sve elemente matrice **H**.

Uvedimo sljedeće oznake:

$$w_j = \max \begin{cases} N_{i,j-1} + o \\ V_{i,j-1} + o \\ 0 \end{cases} \tag{6.5}$$

$$x_j = \max \begin{cases} H_{i,j} + j \cdot e \\ 0 \end{cases} \tag{6.6}$$

Tada se $H_{i,j}$ računa kao:

$$H_{i,j} = \max \begin{cases} H_{i,j-1} + e \\ w_j \\ 0 \end{cases} \tag{6.7}$$

Odnosno,

$$\begin{aligned} x_j &= \max \begin{cases} H_{i,j} + j \cdot e \\ 0 \end{cases} \\ &= \max \begin{cases} H_{i,j-1} + (j-1) \cdot e \\ w_j + j \cdot e \\ 0 \end{cases} \\ &= \max \begin{cases} x_{j-1} \\ w_j + j \cdot e \\ 0 \end{cases} \end{aligned}$$

Pošto za svaki *j* uvijek znamo vrijednosti $w_j + j \cdot e$, za izračun x_j koristit će nam predmetačni račun gdje je binarni operator $\max$. Kad su nam poznate sve vrijednosti x_j, $H_{i,j}$ je jednostavno izračunati:

$$H_{i,j} = \max \begin{cases} x_j - j \cdot e \\ 0 \end{cases}$$

Vidimo da moramo pamtiti samo dva retka matrice, što nam daje memorijsku složenost O(*n*), dok je vremenska složenost O(*mn*).

7. Rezultati

8. Zaključak

Zaključak.

LITERATURA

[1] *CUDA*. URL `http://www.nvidia.com/object/cuda_home_new.html`.

[2] Srinivas Aluru, Natsuhiko Futamura, i Kishan Mehrotra. Parallel biological sequence comparison using prefix computations. *Journal of Parallel and Distributed Computing*, 63:264–272, 2002.

[3] Guy E. Blelloch. *Prefix Sums and Their Applications*. 1990.

[4] Mount. *Bioinformatics: Sequence and Genome Analysis*. 2004.

[5] Saul B. Needleman i Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[6] Bruno Rahle. Swig - poravnanje struktura korištenjem iterativne primjene smith-waterman algoritma. Završni rad, 6 2012.

[7] Temple F. Smith i Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[8] Taylor i Munro. *Multiple sequence threading: conditional gap placement*. 1997.

GPU implementacija vremenski i memorijski učinkovitoga paralelnog algoritma za poravnanje slijedova

Sažetak

Poravnavanje slijedova proteina bitan je dio istraživanja moderne biologije. Kako se radi o velikoj količini podataka, njihova računalna obrada je ključna za učinkovitost istraživanja.

Ovaj rad bavi se pronalaženjem optimalnog poravnanja jednog proteina i liste od N proteina. U informatici se proteini mogu prikazati kao nizovi znakova nad fiksnom abecedom te se njihovo poravnavanje svodi na poravnavanje tih nizova znakova.

Obrada podataka i nalaženje optimalnog poravnanja izvodi se pomoću modificiranog Smith-Watermanovog algoritma s afinom funkcijom kazne i predmetačnim računom.

Ključne riječi: Smith-Waterman, CUDA, paralelizacija, poravnanje, protein, predmetačni račun

GPU implementation of a space and time optimal parallel sequence alignment algorithm

Abstract

Protein sequence alignment makes for a big portion of modern day research in biology. Since the amount of data that needs to be processed is vast, it is crutial to develop optimal software to facilitate this procedure.

This paper deals with finding an optimal alignment between a protein and a list of N proteins. For the purpose of informatics, protein sequences are represented as strings over a fixed alphabet and their alignment comes down to string alignment.

Score of the optimal alignment is found using a modified version of the Smith-Waterman algorithm that uses an affine gap penalty function and prefix computing.

Keywords: Smith-Waterman, CUDA, parallelization, sequence alignment, protein, prefix computing, scan