Univerzitet Crne Gore

Prirodno-matematički fakultet

Smjer: Računarske nauke

Predmet: Paralelni algoritmi

Seminarski na temu

“Paralelni Bellman-Ford algoritam

za najkraće puteve iz jednog čvora”

Profesor: Dr Igor Jovančević Studentkinje: Itana Radičević 8/24,

Dejana Vukčević 1/24

Podgorica, 15.03.2025.

**1. Uvod**

**1.1 Zašto se vrši paralelizacija Bellman-Ford algoritma?**

Bellman-Ford algoritam rješava problem nalaženja najkraćeg puta iz 1 čvora ka svim drugim čvorovima u težinskom digrafu. Radi i sa negativnim težinama za razliku od Dijkstrinog koji ne može. Jedan od važnih slučajeva Bellman-Ford algoritma je detekcija negativnih ciklusa. On može pronaći negativne cikluse, ali ne može proizvesti tačan najkraći put ukoliko se do negativnog ciklusa može doći iz početnog čvora.

Bellman-Ford algoritam ima vremensku složenost O(n⋅m), gdje je n broj čvorova, a m broj grana u grafu. U slučaju potpunog grafa, gdje svaki čvor ima granu prema svakom drugom čvoru, broj grana postaje m = n², što čini vremensku složenost O(n³). Ovaj vremenski okvir može biti problematičan za grafove sa velikim brojem čvorova, kao što su oni sa hiljadama čvorova, jer može dovesti do značajnih kašnjenja i velikih računarskih troškova, čak i na savremenim računarima.

Bellman-Ford je algoritam dinamičkog programiranja. Pojednostavljuje problem traženja najkraćeg puta tako što problem dijeli u potprobleme na rekurzivni način. Koristi princip relaksacije ivica. Svojstvo relaksacije omogućava ažuriranje minimalnih troškova do čvorova u grafu. Izorni čvor ima trošak 0, dok ostali imaju . Cilj je poboljšanje troškova dosega čvorova uzimajući u obzir alternativne puteve za granu (s, v). Postoje dva glavna problema:

1. Ako ima negativnih težina potražnja će biti beskonačna
2. Biranje lošeg poretka može da vodi ka eksponencijalnoj relaksaciji

Glavna sposobnost je relaksacija kroz sve grane više puta omogućavajući tačno izračunavanje najkraćih puteva čak i u prisustvu negativnih težina.

Kako bi se smanjilo vrijeme izvršavanja vrši se paralelizacija Bellman-Ford algoritma koja omogućava da se određeni koraci algoritma obavljaju istovremeno na više jezgara. Umjesto da se ažuriranje rastojanja za svaki čvor vrši sekvencijalno, proces se paralelizuje, čime se omogućava da se više čvorova obradi istovremeno. Ovaj pristup poboljšava iskoristivost resursa računara i značajno ubrzava cjelokupno izvršavanje algoritma, što je naročito važno u slučaju velikih grafova.

**1.2 Detekcija negativnih ciklusa**

U ovom radu će biti fokus na paralelizaciju Bellman-Ford algoritma u detekciji negativnih ciklusa. Detekcija negativnih ciklusa ima značajnu primjenu u stvarnom svijetu, koristi se za detekciju arbitražnih prilika, optimizaciju rutiranja podataka i u mnogim optimizacionim modelima. Npr. u proizvodnji, distribuciji, planiranju, koriste se grafovi sa težinama. Ako model sadrži negativne cikluse, to može ukazivati na greške u modelu (npr. pogrešno unijete podatke ili nedostatke u logici). Postoji šest verzija implementacije ovog algoritma, a eksperimenti su sprovedeni na potpunim grafovima sa 50, 500, 1000 i 2000 čvorova. Rezultati pokazuju da paralelizacija može postići ubrzanje od 1.11x, 5.45x i 6.19x za grafove sa 50, 500 i 1000 čvorova, respektivno.

**3. Paralelna Implementacija**

**3.3.1 V1**

V1 koristi jedan nivo paralelizacije. Algoritam dijeli iteraciju kroz sve grane grafa na dijelove. Sve grane vode ka istom čvoru. Proces relaksacije se vrši itreativno. Sve grane grafe se provjeravaju i ažurira se najkraći put u čvoru.

A computer code with text

AI-generated content may be incorrect.

Slika 3.3.1

**3.3.2 V2-list, V2-vector**

Prvi nivo paralelizma je isti kao i u V1. Drugi nivo paralelizma koristi parMap za iteracije kroz sve grane u svakom djeliću. Ključna razlika između V2 i V2-list, V2-vector leži u tome kako rade korak minimalnog pronalaženja:

1. V2 – radi sekvencijalno, nema paralelizacije za određivanje minimalne vrijednosti
2. V2-list – koristi paralelizam tako što rekurzivno dijeli listu do određene dubine d
3. V2-vector – slično kao V2-list dijeli rekurzivno. Međutim koristi vektore umjesto liste. To je zato što splitAt i length opeacije su brže nad vektorima. Ovdje je vremenska složenost O(1), a kod liste O(n).

|  |  |  |
| --- | --- | --- |
|  | Minimum-finding | Effiency |
| V2 | sequential | N/A |
| V2-list | Recursive list-splitting | O(n) per splitting |
| V2-vector | Recursive vector splitting | O(1) per splitting |

A computer code with text

AI-generated content may be incorrect.

Slika 3.3.2

Prvi nivo je isti kao i u verziji V1 I odnosi se na ***paralelno ažuriranje rastojanja za sve čvorove.***

Svi čvorovi na početku imaju rastojanje , osim početnog čije je rastojanje 0. Niz prethodnika je incijalizovan na None za sve čvorove i koristi se za čuvanje prethodnika. U svakoj iteraciji algoritam dozvoljava relaksaciju svih grana.

Novi niz newDist je prazan i popuniće se rastojanjima za trenutnu iteraciju. newPredecessor je prethodnik za trenutnu iteraciju i prati čvor koji vodi do trenutnog.

Traverse withKey i spawnp su priprema za paralelno pretraživanje i pokretanje paralelnih procesa, odnosno priprema za relaksaciju.

Parallel for v in V je paralelna iteracija nad svim čvorovima v u grafu. candidateDist je privremena lista kreirana za svaki drugi čvor.

Paraller for u in V je paralelna iteracija za sve čvorove u u grafu. Za svaki čvor v ispitujemo rastojanje do svakog drugog čvora u i updejtujemo candidateDist.

candidateDist[u] = dist [u] + - za svaki čvor u privremeno rastojanje se izračunava kao trenutno rastojanje dist[u] + težina grane c(u,y). Ovaj korak izračunava kandidatsko rastojanje za svaki čvor v u, uzimajući u obzir trenutno poznato rastojanje za početni čvor i težinu grane između u i njegovog susjeda y.

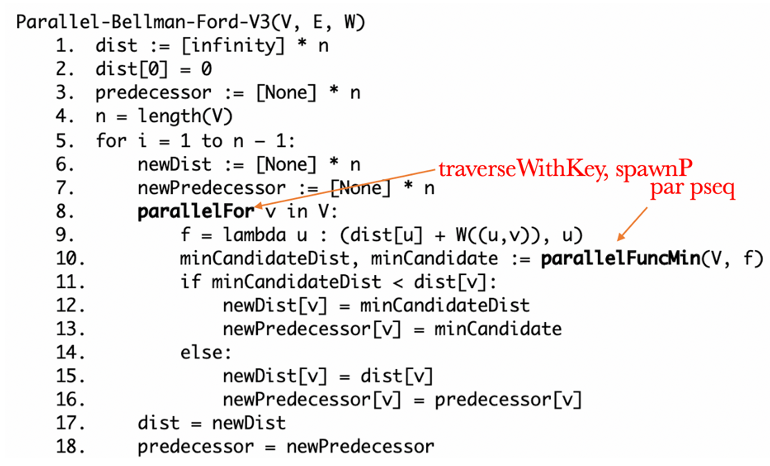
minCandidateDist = parallelMin(candidateDist) – nakon svih paralelnih izvršavanja za candidateDist izračunato je najmanje rastojanje po svim čvorovima koristeći parallelReduction.

newPredecessor[v] = argmin(candidateDist) – za svaki čvor v ova linija predstavlja prethodnika za v kao čvor koji daje najmanju distancu u candidateDist listi. Prethodnici čuvaju najkraći put čuvajući čvor od kojeg je došao najkraći put od trenutnog čvora. Ako nije došlo do promjene trenutna vrijednost rastojanja za čvor v ostaje ista i upisiuje se newDist[v], a rastojanje za čvor v ostaje isto. Nakon što su svi čvorovi obrađeni u paralelnim djelovima dist niz se ažurira novim vrijednostima koje se izračunate u newDist. To znači da su sva rastojanja ažurirana. parallelFor omogućava da se Bellman-Ford ubrza. Svaka iteracija petlje se može izračunati nezavisno i istovremeno. parMap mapira funkciju na više elemenata istovremeno dok parSeq izvršava sekvencijalnu obradu unutar paralelnih sekcija.

**3.3.3 V3-list, V3-vector**

Peta i šesta verziju paralelnog Bellman-Ford algoritma (slika 3.1.3), označene su kao V3-list i V3-vector. Ove verzije predstavljaju unaprijeđenje u odnosu na prethodne verzije (V2-list i V2-vector).

V3-list i V3-vector koriste dva nivoa paralelizacije, pri čemu V3-list koristi listu a V3-vector koristi vektor.

****

Slika 3.3.3

**1. Prvi nivo paralelizacije**

Prvi nivo paralelizacije je isti kao u verziji V1 i odnosi se na ***paralelno ažuriranje rastojanja za sve čvorove****.*

Ovo je implementirano na liniji 8: ***parallelFor*** v in V:

* Ova linija označava početak petlje koja paralelno iterira kroz sve čvorove *v* u grafu što znači da se ažuriranje rastojanja za različite čvorove može obavljati istovremeno na više jezgara.
* Na primjer, ako imamo 1000 čvorova i 8 jezgara, svako jezgro može obraditi po 125 čvorova istovremeno, što značajno smanjuje vrijeme izvršavanja.

**2. Drugi nivo paralelizacije**

Drugi nivo paralelizacije fokusira se na ***paralelizaciju koraka pronalaženja minimuma***u algoritmu. Ovo je implementirano u linijama 9 i 10:

9. f = lambda u: (dist[u] + W((u, v)), u)

10. minCandidateDist, minCandidate: = ***parallelFuncMin(V, f)***

- Linija 9 definiše lambda funkciju f koja izračunava potencijalno novo rastojanje od čvora *u* do čvora *v*. Funkcija vraća par: (novo rastojanje, čvor *u*), gde je novo rastojanje izračunato kao dist[u] + W((u,v)).

- Linija 10 koristi paralelnu funkciju*parallelFuncMin* da pronađe minimum među svim kandidatima za novo rastojanje. Ova funkcija paralelno izračunava vrijednosti za sve čvorove *u* i pronalazi onaj sa najmanjim rastojanjem.

Za razliku od V2-list i V2-vector, koje prvo iteriraju kroz sve čvorove da izračunaju sve nove vrijednosti, a zatim pronađu minimum, V3-list i V3-vector **direktno započinju korak pronalaženja minimuma** i izračunavaju nove vrijednosti tokom tog koraka. Ovo predstavlja značajno poboljšanje u odnosu na V2-list i V2-vector verzije, jer se eliminiše potreba za dodatnom iteracijom složenosti *O*(*n*) kroz sve čvorove radi izračunavanja novih vrijednosti.

**4. Rezultati eksperimenta**

Svi eksperimenti su izvedeni na MacBook Pro računaru sa 2.3 GHz 8-jezgrenim Intel Core i9 procesorom i 16 GB DDR4 memorije (2667 MHz). Sekvencijalna verzija algoritma (V1) korišćena je kao referentna tačka i izvršavana je na jednom jezgru.

**4.1 Performanse na grafovima sa 50 čvorova**

Za analizu performansi, testirano je šest verzija paralelnog Bellman-Ford algoritma na 50 nasumično generisanih potpunih grafova sa 50 čvorova, pri čemu svaki sadrži negativni ciklus dužine 5. Vrijeme izvršavanja predstavljeno u tabeli 4.1 predstavlja prosječne vrijednosti:

Tabela 4.1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Jezgra/Verzija | | V1 | V2 | V2-List | V2-Vector | V3-List | V3-Vector |
| 2 | **0.0248s** | 0.0258s | 0.0327s | 0.0293s | 0.0272s | 0.0255s | |
| 4 | 0.0250s | 0.0257s | 0.0277s | 0.0253s | 0.0253s | 0.0261s | |
| 8 | 0.0264s | 0.0280s | 0.0276s | 0.0276s | 0.0270s | 0.0257s | |

Sekvencijalna verzija algoritma ostvarila je prosječno vrijeme od 0.0275s. Najbolji rezultat u ovoj kategoriji postigla je verzija V1 na 2 jezgra, uz ubrzanje od 1.11 puta. Ostale verzije nisu pokazale značajno poboljšanje, uglavnom zbog dodatnog opterećenja drugim slojem paralelizacije.

**4.2 Performanse na grafovima sa 500 čvorova**

Kod većih instanci problema, algoritmi su testirani na 20 nasumično generisanih potpunih grafova sa 500 čvorova i negativnim ciklusom dužine 50. Prosječno vrijeme izvršavanja prikazano je u tabeli 4.2.

Tabela 4.2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Jezgra/Verzija | | V1 | V2 | V2-List | V2-Vector | V3-List | V3-Vector |
| 2 | 20.026s | 23.080s | 25.460s | 26.149s | 23.687s | 21.967s | |
| 4 | 11.096s | 12.554s | 13.742s | 14.097s | 12.830s | 11.935s | |
| 8 | **6.326s** | 7.746s | 8.650s | 9.026s | 7.493s | 7.063s | |

Sekvencijalna verzija je za ovakav graf imala projsečno vrijeme od 34.487s. Najbolje ubrzanje postignuto je korišćenjem V1 verzije na 8 jezgara, gdje je ostvareno 5.45 puta brže izvršavanje u poređenju sa sekvencijalnim algoritmom. Kod drugih verzija, veći stepen paralelizacije nije doveo do značajnijeg poboljšanja performansi zbog složenije komunikacije i sinhronizacije između niti.

**4.3 Performanse na grafovima sa 1000 čvorova**

Za analizu performansi, testirano je šest verzija paralelnog Bellman-Ford algoritma na 10 nasumično generisanih potpunih grafova sa 1000 čvorova, pri čemu svaki sadrži negativni ciklus dužine 100. Vrijeme izvršavanja predstavljeno u tabeli 4.3 predstavlja prosječne vrijednosti:

Tabela 4.3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Jezgra/Verzija | V1 | V2 | V2-List | V2-Vector | V3-List | V3-Vector |
| 2 | 182.087s | 201.616s | 222.537s | 230.242s | 199.271s | 183.265s | |
| 4 | 95.238s | 108.844s | 117.880s | 122.676s | 107.271s | 98.333s | |
| 8 | **54.976s** | 64.715s | 70.613s | 81.488s | 62.018s | 56.704s | |

Sekvencijalna verzija algoritma ostvarila je prosječno vrijeme od 340.125s. Najbolji rezultat u ovoj kategoriji postigla je verzija V1 na 8 jezgara, uz ubrzanje od 6.19 puta. Verzije sa drugim nivoom paralelizacije sustižu V1, koja ima samo jedan nivo paralelizacije. Performanse V3-vektora su na nivou performansi V1.

**4.4 Performanse na grafovima sa 2000 čvorova**

Za analizu performansi, testirano je šest verzija paralelnog Bellman-Ford algoritma na 5 nasumično generisanih potpunih grafova sa 2000 čvorova, pri čemu svaki sadrži negativni ciklus dužine 200. Vrijeme izvršavanja predstavljeno u tabeli 4.4 predstavlja prosječne vrijednosti:

Tabela 4.4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Jezgra/Verzija | | V1 | V2 | V2-List | V2-Vector | V3-List | V3-Vector |
| 8 | 580.242s | 638.057s | 773.990s | 711.130s | 590.045s | **531.204s** | |

Sekvencijalnoj verziji treba previše vremena i ne završi. Najbolje ubrzanje postignuto je korišćenjem V3 verzije na 8 jezgara, gdje je ostvareno 1.1 puta brže izvršavanje u poređenju sa drugom najboljom verzijom V1. Kada graf ima 2000 čvorova, V3-vektor počinje da nadmašuje V1.

**4.5 Uticaj parametra *depth***

U sprovedenom eksperimentu analiziran je uticaj parametra depth na brzinu izvršavanja paralelne verzije Bellman-Ford algoritma. Parametar depth određuje koliko će slojeva (nivoa) grafa algoritam obraditi paralelno prije nego što se vrati na sekvencijalnu obradu. Ovaj parametar igra ključnu ulogu u definisanju dubine pretrage i raspodjele zadataka među procesima u paralelnim verzijama algoritma, poput V2-vector, V2-list, V3-list i V3-vector.

Testiranje algoritma V3-vector je sprovedeno na dva potpuna grafa sa 1000 čvorova i negativnim ciklusom dužine 100, kao i na dva potpuna grafa sa 2000 čvorova i negativnim ciklusom dužine 200. Rezultati, koji prikazuju prosječna vremena izvršavanja za različite vrijednosti dubine, prikazani su u tabeli 4.5.

Tabela 4.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Čvor/Dubina | 2 | 4 | 6 | 8 | 10 |
| 1000 | 62.849s | 58.786s | **54.881s** | 59.448s | 60.205s |
| 2000 | 589.686s | 558.092s | 569.507s | **543.906s** | 618.661s |

Na osnovu rezultata, optimalna vrijednost parametra depth za grafove sa 1000 čvorova bila je 6, dok je za veće grafove sa 2000 čvorova optimalna vrijednost porasla na 8. Međutim, uticaj parametra depth na ukupne performanse algoritma nije bio značajan. Razlog za to može biti što koraci koji se odnose na pronalaženje minimuma zauzimaju samo mali dio ukupnog vremena izvršavanja algoritma. Kao rezultat toga, promjene u dubini pretrage nisu dovele do velikih razlika u performansama.

Referenca: [Microsoft Word - COMS 4995 report.docx](https://www.cs.columbia.edu/~sedwards/classes/2020/4995-fall/reports/Bellman-Ford.pdf)