Raspoređivanje višeprocesorski zahtevnih taskova na 3 procesora

Seminarski rad u okviru kursa
Računarksa inteligencija
Matematički fakultet

Ivana Jordanov
ivanajordanov47@gmail.com

20. jun 2020.

Sažetak

Raspoređivanje taskova tako da izvršavani redom zauzmu što manji vremenski interval je problem koji će biti opisan i rešavan u ovom radu. Pod rednim izvršavanjem podrazumeva se da se u jednom trenutku samo jedan task može izvršavati na jednom procesoru, ali je zato paralelno izvršavanje više taskova na različitim procesorima dozvoljeno. Ovaj NP težak problem rešavan je algortimima optimizacije. NP-težak, Raspoređivanje, Optimizacija

Sadržaj

1 Uvod

Za neke slucajeve problem je rešiv u pseudo-polinomijalnom vremenu. Sve u svemu problem jeste NP-težak [1]. Ovde je detaljno opisan postupak rešavanja sa sve koracima koji su vodili menjanju i usavršavanju algoritma. Prikazani su rezultati i jasno se vidi kako kroz male promene algoritma nastaju velike promene u performansama. I dalje algoritam nije usavršen za drastično velike instance, a u zaključku se mogu naći neki od načina za poboljšanje.

U okviru rada predstavljeni su algoritmi kojima je, problem opisan u poglavlju 2, rešavan. Sekcija 3 posvećena je opisu implementacije korišćenih algoritama. Kroz čitavo poglavlje može se pratiti razvoj rešavanja problema, redosled kojim su algoritmi menjani i rezultati koje je to povlačilo sa sobom. Na kraju, u poglavlju 4, rezultati su jasno uoredjeni.

2 Opis problema

Problem raspoređivanja taskova na odredjeni broj procesora može varirati. Osnovne razlike su po broju raspoloživih procesora, načinu na koji su taskovi zadati i vremenu kada su taskovi i procesori poznati. Konkretna verzija koja je ispraćena ovim radom podrazumeva postojanje 3 procesora i pravljenje rasporeda taskova tako da se vreme za izvršavanje ma koja dva taska ne podudara na istom procesoru u isto vreme. Svaki task ima unapred određen skup procesora koje koristi i vreme izraženo u procesorskim jedinicama za koje ih zauzima. Svi taskovi su poznati na početku. Cilj je konstruisati takav raspored tako da ukupno vreme od početka izvršavanja prvog taska do kraja izvršavanja poslednjeg taska bude minimalno.

Pored toga što je problem NP-težak, postoje PTAS (eng. polynomial-time approximation scheme[1]) rešenja i to ne samo za slučaj sa tri već za bilo koji fiksirani broj procesora [3].

3 Algoritmi

. Svi algoritmi[2] implementirani su sa uniformnim interfejsom. Imaju isti ulaz i koriste iste klase za predstavljanje taskova i rasporeda. Svaki task ima informaciju o tome koji mu procesori trebaju i koliko traje njegovo izvrsavanje u tikovima[3] i pregled klase dat je na slici 3.

Kako bi se izbeglo pravljenje nedopustivih rešenja i time ubrzao proces, algoritmi su svedeni na traženje redosleda kojim bi se taskovi ubacivali u raspored a sama klasa TaskSchedule (u daljem tekstu raspored) ume da nadoveže susedne taskove tako da se niti oni preklapaju niti između njih formira nepotrebna praznina. Konkretno pamte se tri niza, po jedan za svaki procesor. Svaki put kada novi task biva ubacen u raspored dodaje se po n elemenata u svaki potrebni niz, tako da su indeksi koje task zauzima u svim nizovima isti, a za početni se bira najveći slobodni što se može videti na slici 3.

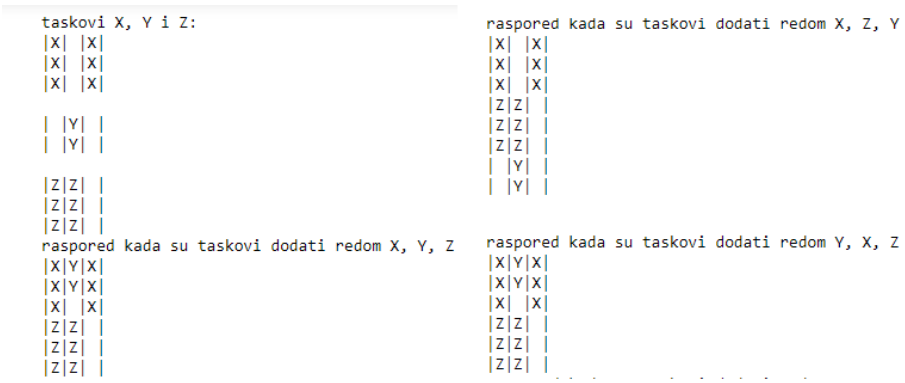
[1] PTAS algoritmi optimizacije su oni za koje važi da se pri dobijanju ulaznih parametara završavaju u polinomijalnom vremenu u odnosu na n, za svako fiksirano $\epsilon, npr. \mathrm{O}(\mathrm{n}^{1/\epsilon})$. Dok je dobijeno rešenje optimalno do na $\epsilon + 1$

[2] Svi algoritmi su dostupni na adresi https://github.com/greenera/Processor_scheduler

[3] Tik - procesorska jedinica.

```python
class Task:
    def __init__(self, len, processorsRequired):
        self.len = len
        self.p = processorsRequired

    def requiers_p1(self):
        return (1 in self.p)

    def requiers_p2(self):
        return (2 in self.p)

    def requiers_p3(self):
        return (3 in self.p)

    def length(self):
        return self.len
```

Kod 1: Taskovi



Slika 1: Zavisnost rasporeda od redosleda dodavanja taskova

3.1 Naivni algoritmi

U svrhu provere ispravnosti rešenja dobijenih naprednim tehnikama, prvo su implementirani naivni algoritmi. Prvi je implementiran backtracking. Već za ulaz od 10 elemenata ima da proveri 10! različitih rešenja što je prilično sporo. Zbog toga je implementiran branch and bound (u daljem tekstu bnb) algoritam koji se pokazao malo bolje. I dalje za ulaz dužine 50 elemenata, pronalaženje optimalnog rešenja traje preko 1h. Ono čime bi mogao da se unapredi ovaj algoritam je zamena rekurzije i DFS sa BFS.

3.2 Genetski algoritam

Isto kao i u prethodnom slučaju, implementaciji algoritma simuliranog kaljenja (u daljem tekstu ša") prethodio je neoptimalan pokušaj. U ovom slučaju prethodnik je genetski algoritam, a pokazao se loš u fazi ukrštanja. Kako je model predstavljen nizom taskova, nije moguće ukrštanje izvršiti prostom zamenom na n-tom indeksu. U tom slučaju dobijena deca bila nosila bi nedopustivo rešenje jer bi se neki taskovi ponavljali a nekih ne bi bilo uopšte.

Ono što je moglo da popravi to je drugačiji model koji je ustvari niz n brojeva, od kojih svaki broj k na poziciji j predstavlja p-ti task koji treba

da se nađe na poziciji j, gde je p jednak broju elemenata n-točlanog niza koji su manji od k ili jednaki k, a sa manjim indeksom j. Iako bi time bio rešen problem dopustivosti rešenja, ovako dobijena deca ne bi direktno bila zavisna od roditelja već bi nosila neku određenu mutaciju. Time bi se prenošenje genetskog materijala otežalo i brzina konvergencije bi se dosta smanjila, a sama funkcija za ukrštanje bila bi neefikasna.

Zaključak iz toga je, da je problem pogodniji za neki algoritam koji ne bi kombinovao rešenja, već bi povremeno menjao postojeće, a takav je SA.

3.3 Simulirano kaljenje

Ovaj algoritam pokazao se kao dobar odabir. Za razliku od BNB, izvršavanje se meri u sekundama. Za kriterijum zaustavljanja postavljeno je vreme trajanja izvršavanja na manje od 90 sekundi, maksimalan broj iteracija na 100.000, a maksimalan broj ponavljanja pivotirajućeg elementa na 50.

Verovatnoća prihvatanja lošijeg rešenja za referentno, u svrhu iskakanja iz lokalnog ekstremuma, postavljena je srazmerno iteracijama tj. $1/\sqrt{i}$.

Biranje suseda se vrši tako što se izabere oko jedna desetina random elemenata i oni menjaju mesto sa takođe random izabranim elemntima. Na ovaj način većina susednih taskova ostaje susedna, tako da susedne instance zaista jesu sličane. Mana ovog metoda je što vrlo sporo vodi u konvergenciju kada se dodje do instance blizu optimalnog. Pošto se zamenjuje n taskova, gde je n oko 1/10 ukupnog broja taskova, vrlo je tesko za dugacke nizove gde je $n > 100$ dobiti bliskog suseda.

Druga mana je ta što taskovi koji zahtevaju sva tri procesora mogu biti izabrana za pomeranje i mogu se lako naći između taskova koji bi mogli da se izvršavaju paralelno. Ova mana potvrđena je i eksperimentalno. Naredni algoritam je kopija kostura ovog algoritma sa popravkom nedostataka.

3.4 Hibridni algoritam

Usled problema kod SA, nastaje ovaj algoritam. Sustinski kao kopija prethodnog, samo sa značajnim efektnim izmenama.

Prva izmena je stavljanje svih troprocesorski zahtevnih taskova na sam početak. Ti taskovi svakako nisu mogli da se paralelno izvršavaju ni sa jednim drugim, a mogli su da smetaju da se drugi taskovi uspešno ukombinuju. Dalje se SA odvija kao i ranije, samo bez mešanja ovih taskova sa ostalima. Samo ova promena dovela je do toga da se dobija optimalno rešenje za male instance gde je $n < 20$, u većini slučajeva za $n < 50$, a za $n > 100$ sa procentom manjim od 10% pronalazi optimalno rešenje.

Sledeći problem koji je uočen je slaba konvergencija. To potvrđuje i činjenica da se algoritam nikad ne završi zbog prekoračenja vremena već ponavljanja pivotirajuće instance. Bilo korisno da se traženje suseda menja u srazmerno poodmaklosti iteracije od početka pretrage. Ova promena uvodi nešto karakteristično za algoritam optimizacije rojem čestica. Ispravljenje ove mane, time da što je veće ponavljanje to je broj taskova koji se menjaju manji, doprinosi konvergenciji, međutim iziskuje promenu vremena izvršavanja budući da je ovog puta 90 sekundi dostignuto već za instancu od dužine 100.

Sledi izmena koja će drugi element sa kojim se vrši zamena birati polurandom, a ne sasvim random kao do sada. Cilj je da od nekoliko

random izabranih elemenata, element ne bude zamenjen već ubacen pored nekog poklapajućeg taska. Time bi se povećala verovatnoća da sused bude bolji od pivotirajućeg elementa.

4 Rezultati

Kratak pregled sporednih algoritama:

- backtracking - spor, zagarantovano optimalno rešenje
- BNB - za 10 taskova vec mu treba preko 20min, zagarantovano optimalno rešenje
- SA - brzo, inskorvergira u lokalni ekstremum i za male ulaze
- Hibridni alg - brz, nalazi optimalno rešenje u preko 90% slučajeva za n ¡ 70, već za veće instance daje približno optimalno rešenje

Drugi najznačajniji algoritmi ove oblasti su Goemans-ov algoritam, normalni-raspoređivač... [2]

5 Zaključak

U radu je pokazano kako se malim promenama algoritama optimizacije mogu dobiti značajne razlike u efikasnosti. Sa nemogućnosti da se problem od preko 10 taskova reši za manje od 20 minuta, dolazi se do toga da problem od 30 taskova biva rešen za manje od 2 sekunde i to sa optimalnim rešenjem. Na uštrb efikasnosti, implementirani algoritam ne garantuje optimalno rešenje za veće instance. Ono što može da se uradi po tom pitanju je implementacija novog traženja suseda za dati algoritam. Neka naredna verzija ovog hibridnog algoritma mogla bi da uvede prilagođeno traženje suseda, u zavisnosti na odmaklost konvergencije. Slično algoritmu optimizacije rojem čestica. Druga opcija je reimplementacija funkcije za traženje suseda tako da "pametno"bira gde će da ubaci selektovani task.

Literatura

[1] MINIMUM 3-DEDICATED PROCESSOR SCHEDULING. https://www.csc.kth.se/~viggo/wwwcompendium/node188.html. Accessed: 2020-06-10.

[2] Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

[3] Jianer Chen and Antonio Miranda. A polynomial time approximation scheme for general multiprocessor job scheduling. *SIAM Journal on computing*, 31(1):1–17, 2001.