ACM Symposium on Parallel in Algorithms and Architectures

Conference SPAA 2020

Sara Bizjak | Bor Brecelj | Zala Erič | Laura Guzelj Blatnik
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

Marec 2021

1 Konferenca SPAA 2020

Med 14. in 16. julijem 2020 je potekal že 32. letni simpozij (SPAA 2020), prvič pa je bil organiziran v Santa Fe (ZDA) leta 1989 [1].
Glavna tema te konference je paralelizem, kar se nanaša na vsak računalniški sistem, ki lahko hkrati izvaja več operacij oziroma nalog. Zaradi velike širitve in vedno večjega pomena paralelizma se povečuje tudi interes in obiskanost konference SPAA, katere cilj je razviti in poglobiti razumevanje paralelne izračunljivosti tako v teoriji kot praksi. Obseg tematik akademskih člankov tako sega vse od zelo teoretičnih do zelo praktičnih tem, ki vključujejo paralelne in porazdeljene algoritme ter podatkovne strukture, energetsko učinkovite modele, upravljanje množičnih podatkovnih nizov, vzporedno teorijo zapletenosti, večjedrne sisteme, prevajalnike in orodja za sočasno programiranje, strojno in programsko opremo za transakcijski pomnilnik, internet in zvetovni splet, teorijo iger in skupno učenje, mobilna in senzorska omrežja, pokrivajo pa še nekatere druge sorodne vsebine.
Akademski raziskovalci in znanstveniki so na 32. simpozij prijavili 127 člankov, odbor pa jih je kot običajne članke sprejel 41 (32 %). Ker je bila leta 2020 SPAA – v skladu z omejitvami glede pandemije COVID-19 – prvič virtualna konferenca, je programski odbor odobril in objavil tudi kar 27 prispevkov v obliki kratkih naznanil, saj so v tem videli priložnost za bolj živahno in aktivno virtualno razpravo udeležencev konference [1].
Prvi dan konference je bil v znamenju delavnic in vaj, v naslednih dveh dneh pa si so sledile predstavitve člankov v dvanajstih sejah, kjer je bila otvoritvena seja namenjena pregledu petih člankov, nominiranih za nagrado najboljšega članka na konferenci [2]. Za najboljši članek so kasneje izbrali članek avotrjev Irvana Jahja in Haifeng Yuja iz Univerze v Singapurju, ki nosi naslov *Sublinear Algorithms in T-interval Dynamic Networks.*

2 Znanstveni članki

V nadaljevanju bodo predstavljeni trije izbrani akademski članki, sprejeti in objavljeni na konferenci SPAA 2020:

1. *"Optimal Parallel Algorithms in the Binary-Forking Model"*, ki so ga napisali Guy E. Blelloch, Jeremy T. Fineman, Yan Gu in Yihan Sun [3].

2. *"Optimal Resource Allocation for Elastic and Inelastic Jobs"*, ki so ga napisali Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang in Justin Whitehouse [4].

3. *"Randomized Incremental Convex Hull is Highly Parallel"*, ki so ga napisali Guy E. Blelloch, Yan Gu, Julian Shun in Yihan Sun [5].

2.1 Optimal Parallel Algorithms in the Binary-Forking Model

Avtorji: Guy E. Blelloch, Jeremy T. Fineman, Yan Gu in Yihan Sun

Za analizo vzporednih algoritmov se povečini uporablja model PRAM (*angl. parallel random access machine*), ki predpostavlja, da se operacije izvajajo sinhrono, kar je težko izpeljati v praksi. Zato se v tem članku avtorji osredotočijo na optimalne algoritme v t. i. binary-forking modelu, ki je bolj realističen, ker opiše moderne večjedrne računalnike. Problem nastane, če želimo prenesti optimalne algoritme iz PRAM modela v binary-forking model, ker se poveča njihov čas izvajanja in algoritmi niso več optimalni.

Binary-forking model dovoljuje, da se procesi izvajajo asinhrono, vsi procesi imajo dostop do skupnega pomnilnika in razcep procesa (*angl. fork*) je možen le v dva procesa naenkrat. Pomembno je, da lahko procesa na koncu združimo. Recimo, da imamo ukaza `end` in `join`. Ukaz `end` konča proces, `join` pa počaka, da se drugi proces iz zadnjega razcepa konča in šele nato nadaljuje z izvajanjem. Avtorji model poenostavijo tako, da namesto ukaza `join`, uporabijo atomarni ukaz Test-and-Set (v nadaljevanju TS), ki naenkrat prebere bit in ga nato nastavi na 1. S tem ukazom je mogoče implementirati `join` tako, da ko procesa prideta do `join` ali `end`, izvedeta ukaz TS nad vnaprej določenim bitom, ki je bil nastavljen na 0. Če TS vrne 0, pomeni, da se drugi proces še ni končal, in se proces, ki je klical TS, konča. Če TS vrne 1, pomeni, da se je drugi proces že končal in mora ta proces nadaljevati izvajanje. Uporaba ukaza TS v modelu je smiselna, ker je implementiran v vseh novejših računalnikih in s tem je binary-forking model še vedno realističen.

Predstavljenih je nekaj novih optimalnih algoritmov v binary-forking modelu, ki rešujejo osnovne probelme. Ti algoritmi so optimalni glede na število operacij vseh procesov skupaj (*angl. work*) in glede na čas izvajanja, ki je definirano kot največje število zaporednih operacij.

Podrobneje predstavimo problem, kjer kot vhod dobimo dvosmeren povezan seznam. Naša naloga je vsakemu vozlišču določiti rang oziroma razdaljo od začetka seznama (*angl. list ranking problem*). Problem bomo reševali tako, da bomo seznam krčili dokler ne dobimo seznama z enim vozliščem (*angl. list contraction*). Vozlišča lahko brišemo vzporedno, če zagotovimo, da

ne bomo naenkrat izbrisali dveh sosednjih vozlišč. To dosežemo tako, da vsakemu vozlišču vnaprej naključno določimo pomembnost. Brisanje vozlišča dovolimo le, če imata oba soseda večje pomembnosti kot on. Algoritem se razdeli v n procesov, kjer je n dolžina seznama. Skupaj opravi $O(n)$ operacij, kar je enako kot s sekvenčnim algoritmom. Čas algoritma je $\mathcal{O}(\log n)$, kar je optimalno, ker že razdelitev v n procesov potrebuje $\mathcal{O}(\log n)$ operacij.

V članku so predstavljeni še algoritmi za urejanje seznama, operacije nad urejenimi množicami (unija, presek in razlika), generiranje naključnih permutacij, iskanje minimuma na intervalu (*angl. range minimum query*) in krčenje dreves (*angl. tree contraction*).

2.2 Optimal Resource Allocation for Elastic and Inelastic Jobs

Avtorji: Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang in Justin Whitehouse

V članku je predstavljen model, ki poskuša optimizirati uporabljenost strežnikov z uporabo določenih lastnosti opravil. Ločimo elastična opravila, ki lahko vzporedno tečejo na več strežnikih, in neelastična, ki potrebujejo natanko enega.

Motivacija za raziskavo je problem neizkoriščenih strežnikov v podatkovnih centrih zaradi opravil, ki se različno učinkovito porazdelijo. V praksi se s tem problemom velikokrat uspešno kosajo s polavtomatizacijo urnikov, vendar teoretično ni zanesljiva. Model problema poskusi poenostaviti resnično stanje in hkrati obdržati nianse glede strežnikom nepoznanih in naključnih velikosti opravil. Še s tako preprostim modelom je problem netrivialen, še težje pa je z dodajanjem raznih parametrov, ki se v resnici pojavljajo. V članku je pojasnjen tudi razlog za uporabo stohastične analize, s katero se izognemo nekaterim pomanjkljivostim prejšnjih raziskav, saj nekatere upoštevajo le neelastična opravila poznanih velikosti in se zanimajo le za analizo najslabšega primera.

Formalna definicija modela problema je podana kot sistem k strežnikov s procesno hitrostjo 1 enota dela na sekundo, na katerih tečeta dve vrsti opravil, elastična in neelastična. Elastičnim pripišemo faktor pohitritve, ki je sorazmeren s številom strežnikov, na katerih teče. Vsako opravilo ima še velikost, ki določa kako hitro je opravljen na enem strežniku. Velikosti opravil so sistemu neznane, vendar pripadajo eksponentnima razporeditvama μ_I in μ_E. Opravila v sistem prihajajo s Poissonovim postopkom (naključno) glede na λ_I in λ_E. Dodelitvena strategija π določa, koliko strežnikov lahko določeno opravilo dobi v času t. Sistem je nato modeliran kot nepretrgana Markova veriga, kjer vsako stanje določa število elastičnih in neelastičnih opravil trenutno v sistemu. Definirana je še časovna spremenljivka, cilj pa jo je minimizirati. Vpeljani sta strategiji Elastična-prvo (EF, *angl. Elastic-First*) in Neelastična-prvo (IF, *angl. Inelastic-First*).

V tretjem poglavju so opisana poprejšnja dela, ki nas opomnijo, da v praksi veliko podatkovnih centrov uporabnikom omogoča rezervacijo števila strežnikov, avtomatizirano pa je zgolj dodeljevanje posameznih. Članek je podoben tudi nekaterim starejšim teoretično orientiranim delom, a se v tem članku osredotočamo samo na analizo najslabšega primera, ki pa ni reprezentativna.

V četrtem poglavju so predstavljeni rezultati optimalosti v različnih primerih. V primeru, ko so neelastična opravila generalno enako velika kot elastična, se izkaže, da je strategija IF optimalna. V primeru večjih neelastičnih opravil se za boljšo, vendar ne optimalno strategijo, ponovno pozkaže IF. Poglavje zaokroži primer manjših neelastičnih opravil, kjer se za boljšo, vendar ne optimalno stretegijo, izkaže EF .

V petem poglavju sledi analiza odzivnih časov, pridobljenih v prejšnjem poglavju. Izpeljan je povprečni odzivni čas strategije EF za različne vrednosti μ_I, μ_E, λ_I, λ_E, in k. Najprej so za potrebe EF vpeljane $2D$-neskončne Markove verige, ki so v nadaljevanju s stohastičnimi tehnikami prevedene v $1D$-neskončne. Tako lahko z metodami iz matrične analize rešimo enodimenzionalno verigo in pridobimo porazdelitev ter s tem tudi povprečni odzivni čas EF. Podobno je narejeno za strategijo IF.

Članek je prvo delo na področju avtomatizacije urnika opravil, kjer so opravila glede na paralelizabilnost heterogena. Sledi povzetek, kako so ob različnih velikostih elastičnih in neelastičnih opravil različne strategije boljše. Z obzirom na primer opravil, ki so elastična zgolj do neke mere, je še pojasnjeno, da bi se lahko tematika razširila na več področij.

2.3 Randomized Incremental Convex Hull is Highly Parallel

Avtorji: Guy E. Blelloch, Yan Gu, Julian Shun in Yihan Sun

V članku je predstavljena ideja paralelizacije algoritma za reševanje problema inkrementalne konveksne lupine točk. Konveksna lupina točk je definirana kot najmanjša konveksna množica, ki vsebuje vse podane točke (na robu ali v notranjosti). Ideja inkrementalnega algoritma je, da neko konveksno množico točk že imamo (začnemo s tremi točkami in poiščemo njihovo konveksno lupino), v vsakem naslednjem koraku algoritma pa bi množico radi razširili z eno točko. Obravnavati moramo dva primera – če točka leži v notranjosti obstoječe lupine, algoritem nadaljuje z obravnavo naslednje točke, če pa točka leži izven lupine, moramo le-to razširiti tako, da ohranimo konveksnost lupine. Problem lahko posplošimo na poljubno dimenzijo prostora. Glavna ideja za paralelizacijo algoritma je, da bi si želeli dva neodvisna robova v konveksno množico dodati istočasno oziroma ju istočasno izbrisati, v kolikor sta ju nadomestila nova robova. Povezavo namreč izbrišemo natanko tedaj, ko ni del zunanjega roba, hkrati pa je vidna iz neke točke v množici.

Paralelnost tako dosežemo z vpeljavo pojma nosilne množice (*angl. support set*), ki loči "pomembne" robove od "nepomembnih" (takih, ki jih lahko izbrišemo). Nosilna množica je pri problemu konveksne lupine sestavljena iz parov robov, ki imajo skupno krajišče. Ko na novo dodano točko p zunaj konveksne lupine povežemo z že obstoječo lupino preko roba $r = \langle p, t \rangle$, bo ta rob zamenjal enega od robov, ki so v nosilni množici roba r oziroma enega od robov s krajiščem v točki t. Tako velja, da je rob, ki ga dobimo z dodajanjem nove točke, odvisen od natanko dveh že obstoječih robov. Ključnega pomena, da je paralelnost algoritma sploh mogoča, je dejstvo, da je velikost nosilne množice konstantna, kar pa v našem primeru drži. Ker je velikost nosilne množice za vsak novo dodani rob enaka 2 to pomeni, da je konfiguracijski prostor pri problemu konveksne lupine 2-nosilen.

Paralelizabilnost in njeno globino danega algoritma izpeljemo s pomočjo konfiguracijskega grafa, definiranega na nosilnih množicah. S pomočjo nekaj novo definiranih pojmov in trditev pridemo do glavnega izreka, ki pravi, da ima konfiguracijski graf odvisnosti za reševanje inkrementalne konveksne lupine v d dimenzijah na n točkah globino $\mathcal{O}(\log n)$ z visoko verjetnostjo.

Kot posledica glavnega izreka je predstavljen paralelizabilen algoritem za reševanje zastavljenega problema. Algoritem temelji na zaporednem algoritmu, le da je zaradi enostavnejšega vrstnega reda dodajanja povezav ta enostavnejši in v večji meri paralelizabilen. Predstavljen algoritem ima pričakovano časovno zahtevnost $\mathcal{O}(n^{\lfloor d/2 \rfloor} + n \log n)$, globina rekurzije pa je $\mathcal{O}(\log n)$.

V zaključku članka je predstavljena še razširitev na degeneracijske primere – primere, ko množica točk v ravnini vsebuje tri kolinearne točke oziroma štiri ali več točk leži v isti ravnini. Avtorji se na kratko dotaknejo še drugih problemov iz računalniške geometrije, pri katerih bi se dalo uporabiti ugotovitve članka. Dva izmed takih problemov sta iskanje presečišč množice enotskih krogov in iskanje presečišč polprostorov v d dimenzijah.

Literatura

[1] *ACM Symposium on Parallel in Algorithms and Architectures, conference SPAA 2020*, opis konference, [ogled 10. 3. 2021], dostopno na `dl.acm.org/doi/proceedings/10.1145/3350755`.

[2] *ACM Symposium on Parallel in Algorithms and Architectures, conference SPAA 2020*, urnik konference, [ogled 10. 3. 2021], dostopno na `spaa.acm.org/2020/SPAA-Program.pdf`.

[3] Guy E. Blelloch, Jeremy T. Fineman, Yan Gu in Yihan Sun, *Optimal Parallel Algorithms in the Binary-Forking Model*, SPAA 2020, virtualni dogodek, ZDA (julij 2020), 89–102, [ogled 12. 3. 2021], dostopno na `dl.acm.org/doi/pdf/10.1145/3350755.3400227`.

[4] Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang in Justin Whitehouse, *Optimal Resource Allocation for Elastic and Inelastic Jobs*, SPAA 2020, virtualni dogodek, ZDA (julij 2020), 75–87, [ogled 12. 3. 2021], dostopno na `dl.acm.org/doi/pdf/10.1145/3350755.3400265`.

[5] Guy E. Blelloch, Yan Gu, Julian Shun in Yihan Sun, *Randomized Incremental Convex Hull is Highly Parallel*, SPAA 2020, virtualni dogodek, ZDA (julij 2020), 103–115, [ogled 12. 3. 2021], dostopno na `dl.acm.org/doi/pdf/10.1145/3350755.3400255`.