

Analiza porazdeljenega programskega jezika Julia in primerjava z dobro znanimi alternativami

Jan Pelicon, Blaž Rojc

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana, Slovenija

Povzetek. Primerjala sva jezika C/C++ in Python z alternativno izbiro - jezikom Julia. Ugotovila sva, da se po hitrosti bliža C-ju, prinaša pa fleksibilnost in prijaznost Pythona. Težava je le v relativni novosti jezika, kar se kaže kot nedokončanost določenih delov.

Ključne besede: primerjava programskih jezikov, porazdeljeni programski jeziki, hitrost izvajanja, prijaznost do uporabnika

1 UVOD

Porazdeljeno programiranje predstavlja težavo, s katero se večina programerjev nerada spoprijema. V čist, striktno determinističen tok programiranja vnaša nepredvidljivost, simultane dogodke, neurejenost in kup varnostnih ter pravilnostnih lukenj. K zahtevnosti programiranja pa pripomore tudi slaba podprtost porazdeljenega programiranja v bolj permisivnih programskih jezikih - tako Python kot JavaScript zahtevata daljšo poglobitev v njune sisteme, ki omogočajo distribucijo problemov po več jedrih ali procesorjih, poleg tega pa zaradi svojih omejitev ne morata omogočiti tako nizkonivoevne nadzora in optimizacije kot C.

Tu vstopi Julia, dinamičen, prevajan jezik, prvotno namenjen potrebam numerične analize in računske znanosti, a primeren za obilico problemov, kjer je hitrost izvajanja tako pomemben faktor kot enostavnost programiranja.[1] Obljublja enostavno in učinkovito distribucijo dela, hkrati pa ohranja prijaznost do uporabnika.

2 JEZIK

Julia predstavlja zanimivo sredino med hitrostjo C-ja in prijaznostjo do uporabnika raznih dinamičnih programskih jezikov, kot so Python, JavaScript, MATLAB, Lua, ...

Julia se pred izvajanjem optimizira in prevede v strojno kodo z uporabo programske opreme LLVM*. To omogoča učinkovito izrabo pomnilnika in procesorskega časa, posledično pa tvorbo programov, ki se v hitrosti približajo ekvivalentni implementaciji v jeziku C.[2]

Je dinamičen programski jezik v polnem pomenu besedne zveze. Med drugim omogoča dinamično izpeljavo podatkovnih tipov, modificiranje in izvajanje kode iz besedilnih nizov,[3] refleksijo, makre in večmetodnost funkcij.

V času pisanja tega dokumenta je jezik tudi že dosegel različico 1.0, a kljub temu še ni popolnoma stabilen. Kot

bo opisano kasneje ostaja še velik del obljubljenih funkcionalnosti neimplementirane, medtem ko ima stabilnost in hitrost obstoječe kode višjo prioriteto.

3 PORAZDELJENO RAČUNANJE

Julia podpira več oblik delitve dela. Osredotočili se bomo na metode najpodobnejše tistim, ki smo jih obravnavali pri predmetu PS: eksplicitna delitev dela med procesorska jedra in pošiljanje dela koprosesorjem - GPE.

Julia v teoriji podpira delitev dela med procesorska jedra na več načinov. Ti so razdeljeni v dve skupini, na osnovi niti in na osnovi procesov.

3.1 Večnitenje

Večnitenje je v Julii eksperimentalna funkcija.

Uporabnost je v času pisanja zelo omejena, na voljo sta le makra `@threads`, ki na več nitih paralelno izvede `for` zanko, in `@threadcall`, ki funkcijo v jeziku C pokliče v ločeni niti.[4] Druga možnost zahteva pisanje dela programa v različnem programskem jeziku in se ji bomo izognili. Prva pa omogoča določeno stopnjo paralelizacije, kljub temu da morda ni tako učinkovita kot v drugih programskih jezikih.

Število niti, s katerimi naj Julia razpolaga, določimo s sistemsko spremenljivko `JULIA_NUM_THREADS`. Problem, ki ga želimo paralelizirati, predstavimo kot `for` zanko, ki iterira nad problemi, ki jih želimo reševati vzporedno. Nad to zanko kličemo makro `@threads`, ki jo porazdeli med razpoložljive niti.

Ta sistem ni primeren za kompleksnejše probleme, ki zahtevajo sinhronizacijo med nitmi. Če je to potrebno, je bolje problem razdeliti na več podproblemov in vsakega reševati v ločeni paralelizirani `for` zanki. Zaradi eksperimentalnega stanja niti je podpora pri razhroščevanju pomanjkljiva ali celo neobstoječa.[5]

3.2 Večprocesnost

Večprocesnost je veliko bolj fleksibilna možnost. Procesi se v Julii lahko dodajajo poljubno med izvajanjem, komunikacija pa poteka prek klicev iz glavnega procesa

* <http://llvm.org/>

v ostale. Največja ovira tega pristopa pa je potreba po eksplicitnem podajanju podatkov med procesi, ki poleg dodane kompleksnosti prinaša še dodatno zahtevnost izvajanja in počasnejše računanje.

4 PRIMERJAVA Z JEZIKOM PYTHON

Med dinamičnimi programskimi jeziki po popularnosti trenutno kraljuje Python.[6][7] Je anekdotno eden najlažjih jezikov za popolne začetnike, omogoča pa tudi reševanje kompleksnejših problemov s širokim naborom javno dostopnih knjižnic - modulov.[8]

Njegova največja hiba je relativna počasnost. Python je interpretiran jezik, ki se najprej prevede v bitno kodo, nato pa izvaja v virtualnem stroju. Posledično se kompleksni programi izvajajo veliko počasneje kot v C-ju.

Julia ohranja večino prednosti Pythona, predvsem dinamičnost in prijaznost do programerja, prinese pa prednost hitrega izvajanja neposredno prevedene kode.

Na žalost Julia ne uživa take stopnje podpore kot Python glede obstoječih knjižnic, kar pa je posledica relativne novosti jezika in pomanjkanja izkušenih, zavzetih programerjev. K večji sprejetosti jezika naj bi v prihodnosti pripomogla tudi večja zanesljivost, ki naj bi prišla z uporabo jezika v večjih projektih.

4.1 Python multiprocessing

Za deljenje dela med jedra sta na voljo v Pythonu modula "multiprocessing"* in "threading"†, a na žalost se zaradi globalnega zaklepanja interpreterja (GIL)[9] v CPython implementaciji lahko izvaja naenkrat le koda ene niti v vsakem Python procesu. Za učinkovito deljenje dela moramo torej uporabiti modul "multiprocessing". Ta nam omogoča zaganjanje poljubnega števila procesov in deljenje podatkov med njimi.

Tako kot v drugih programskih jezikih je tudi v Pythonu delo s procesi precej počasnejše kot z nitmi, ampak dokler je globalni zaklep prisoten, nimamo možnosti izbire.

5 PRIMERJAVA Z JEZIKOM C/C++

TODO

5.1 pthreads

TODO

5.2 OpenMP

TODO

* <https://docs.python.org/3.7/library/multiprocessing.html>

† <https://docs.python.org/3.7/library/threading.html>

6 ALGORITMA

Za primerjavo jezikov sva implementirala dva algoritma.

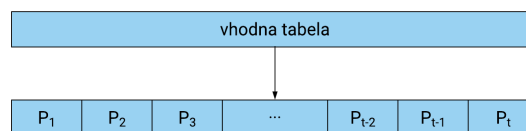
Prvi, Adaptive Quick Sort, je paralelna implementacija sortirnega algoritma Quick Sort. Izvaja se na vseh razpoložljivih nitih sistema. Zasnovan je tako, da paralelno izvede čim večji del računanja, tudi deljenje in združevanje podatkov.

Drugi simulira JPEG kompresijo. Izvaja se na grafični kartici prek knjižnic OpenCL in CUDAnative.

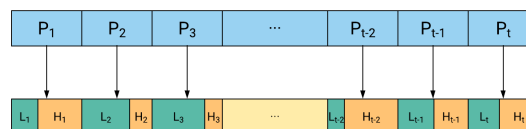
6.1 Adaptive Quick Sort

Veliko vzporednih implementacij algoritma Quick Sort paralelizira le sortiranje, ne pa tudi deljenja in združevanja podatkov. Adaptive Quick Sort porazdeli vse, kar omogoča hitrejše izvajanje pri zelo velikih naborih podatkov.

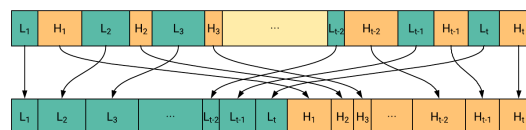
Idejno se vsak korak algoritma izvede v 3 delih. Naj bo "n" število podatkov v tabeli, "t" pa število niti oz. procesov, ki jih ima algoritem na voljo. V prvem delu se tabela razdeli na t delov in določi se pivotni element:



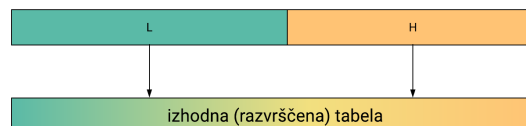
Nato vsaka nit "in-place" razdeli svoj del na elemente manjše ali enake pivotu (L) ter elemente večje ali enake pivotu (H):



Vse niti se nato uporabijo za premik "spodnjih" delov na začetek in "zgornjih" delov na konec tabele, vsaka nit premakne svoj kos L in kos H:



Na koncu koraka se rekurzivno na obeh novo ustvarjenih delih tabele kliče naslednji korak v dveh ločenih nitih, vsaka dobi del nabora niti za delo:



Po koncu vseh rekurzivnih klicev je tabela sortirana. Če je v nekem koraku na voljo le ena nit, se v tej niti izvede navaden Quick Sort.

Ta implementacija zahteva relativno veliko koordinacije med nitmi in usklajevanja velikosti delov. Posledično zahteva več premisleka in dela kot le parale-

lizacija zanke. Ta algoritem dobro pokaže prednosti in slabosti jezikov ko pride do fleksibilnosti paralelizacije.

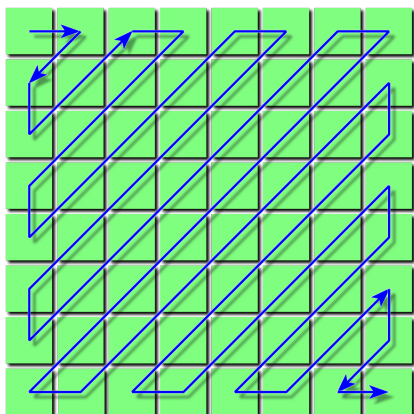
6.2 JPEG

Format JPEG* je eden izmed najbolj razširjenih digitalnih formatov za shranjevanje slik. Za zmanjševanje količine podatkov, ki ga slike zasedajo, izrablja lastnosti človeškega vida - ljudje težje zaznamo spremembe pri visokofrekvenčnih komponentah slike kot pri nizkofrekvenčnih.

Kodiranje je sestavljeno iz treh glavnih delov, preslikave 8x8 kosov slike v frekvenčno domeno, kvantizacije in končnega stiskanja.

Preslikava je navadno diskretna kosinusna transformacija.[†] Ta je implementirana kot množenje matrike vhodnih podatkov z matriko realnih koeficientov.

Kvantizacija poteka v treh korakih. Najprej se podatki delijo z ustreznimi vrednostmi v kvantizacijski matriki. Ta določa, katere frekvence naj se bolje ohranijo. Nato se elementi matrike zaokrožijo na cela števila med -128 in 127. Nazadnje pa se še matrika pretvori v vektor dolžine 64 prek poševnega cik-cak vzorca:



Slika 1: vir: Wikipedia

Končno stiskanje je brez izgub. Navadno se uporablja Huffmanovo kodiranje.[‡] Pri dobro izbrani kvantizacijski matriki se večina ničelnih elementov nahaja na koncu vektorja, kar lahko uporabimo, da podatke še bolj stisnemo.

Preslikava in prva dva koraka kvantizacije sta zelo primerna za izvajanje na GPE. Algoritem simulira popačenje, ki se pojavi pri stiskanju, tako, da izvede DCT, deljenje s kvantizacijskimi vrednostmi in zaokroževanje, nato pa izvede obraten postopek, podatke množi s kvantizacijskimi vrednostmi in izvede inverzen DCT.

Ta algoritem je relativno enostaven pri implementaciji, je pa dovolj zahteven, da prikaže razlike med

*<https://en.wikipedia.org/wiki/JPEG>

[†]https://en.wikipedia.org/wiki/Discrete_cosine_transform

[‡]https://en.wikipedia.org/wiki/Huffman_coding

hitrostmi prevajanja in izvajanja programov na GPE med različnimi jeziki.

7 OKOLJE

Testiranje se je izvajalo na namiznem računalniku s procesorjem AMD Ryzen Threadripper 1920X na tovarniško nastavljeni hitrosti, 32 GB pomnilnika s hitrostjo 3000 MHz in grafično kartico Nvidia GeForce GTX 1080 Ti. Nameščen je bil operacijski sistem Windows 10 - 64 bit, različica 1809.

Programi v jeziku C/C++ so se prevajali in zaganjali v programskem okolju Microsoft Visual Studio 2017 z nastavitvami za "pthreads", "OpenMP" in "OpenCL" kot je zapisano v navodilih za vaje pri predmetu Porazdeljeni sistemi.

Program v jeziku Python se je zaganjal v ukaznem pozivu Powershell, Python verzija 3.7.1 64 bit, numpy verzija 1.15.4.

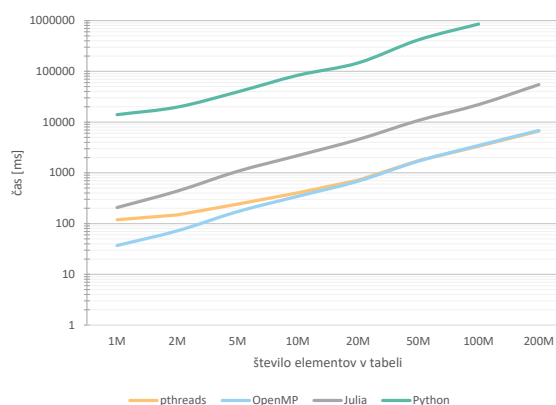
Programi v jeziku Julia so se zaganjali v ukaznem pozivu Powershell, Julia verzija 1.0.3 64 bit, CUDAnative verzija 0.9.1.

8 ANALIZA HITROSTI

8.1 Adaptive Quick Sort

Algoritem očitno izpostavi obseg razlik hitrosti posameznih jezikov in kako velikost podatkov vpliva nanje. Programi so bili testirani na eksponentno povečujoči tabeli naključno generiranih predznačenih 64-bitnih celih števil. Za vsako velikost se je vsak program izvedel petkrat, vsakokrat z na novo generiranimi števili. Največja tabela je vsebovala 200 milijonov elementov. Končne vrednosti so povprečja petih izvajanj.

Vsak program je bil spisan na način, ki je minimiziral število nepotrebnih dostopov do pomnilnika in hkrati kar najbolje sledil prej opisanemu načrtu. V primeru Pythona in Julie je jezik omejeval ta aspekt optimizacije, kar pa bo obravnavano kasneje.



Slika 2: Čas sortiranja tabele

Med pthreads in OpenMP je razlika največja pri majhnih tabelah, kjer je OpenMP veliko hitrejši. Ta razlika se bolj ali manj izniči pri tabelah z več kot 20 milijoni elementov.

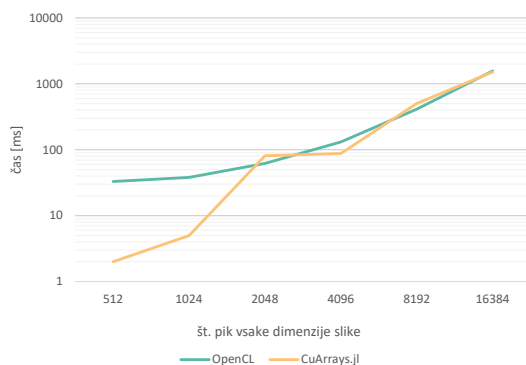
Julia je le slabo magnitudo počasnejša. Za majhne tabele je skoraj tako hitra kot pthreads. Med 2 in 20 milijoni elementov počasi izgublja hitrost. Pri 200 milijonih pa nastopi težava s količino pomnilnika, zato se izvajanje še dodatno upočasni.

Python je od vseh jezikov najpočasnejši. Pri 100 milijonih elementov doseže razmerje časa izvajanja v Pythonu s časom izvajanja s pthreads faktor 250. Zaradi zelo dolgega časa testiranja pri tabeli velikosti 100 milijonov - približno 1 uro in 15 minut - se za 200 milijonov elementov ni testiralo.

8.2 JPEG

Za razliko od sortirnega algoritma sta tu implementaciji skoraj enakovredni. Programa sta bila testirana na čedalje večjih kvadratnih slikah, velikosti dimenzij so bile naraščajoče potence števila 2. Za vsako velikost sta se programa najprej dvakrat zagnala brez merjenja, nato pa je sledilo 5 meritev. Končne vrednosti so povprečje teh meritev.

Programa sta semantično skoraj identična. Edine razlike so sintaktične, kjer se jezika razlikujeta. Sicer pa je postopek izvajanja popolnoma enak, noben program ni optimiziran bolj kot drugi.



Slika 3: Čas simulacije JPEG popačenja brez prevajanja

Implementacija CUDA vmesnika v jeziku Julia je v hitrosti za velike slike skoraj popolnoma primerljiva s C implementacijo vmesnika OpenCL. Če zanemarimo čas prevajanja, je pri majhnih slikah Julia kar 15-krat hitrejša, ker med klici ohranja naložene funkcije v spominu GPE.

Zgodba se popolnoma obrne, če upoštevamo le prvi zagon skupaj s prevajanjem. V okolju Visual studio od začetka prevajanja do konca shranjevanja preteče le slabih 5 sekund, ukazni poziv Julie pa porabi kar pol minute.

9 ZAHTEVNOST IN IZKUŠNJA PROGRAMIRANJA

Hitrost izvajanja je le ena plat zgodbe. Vse privarčevane sekunde ne pomenijo nič, če mora programer ure in ure razhroščevati težave brez ali s slabimi sporočili napak.

9.1 C/C++

TODO

9.2 Python

Kodiranje je v Pythonu izjemno uporabniku prijazno.

Spremenljivke pridobijo tip na podlagi argumentov, ki jih prejmejo. Če želimo spremeniti tip spremenljivke med izvajanjem, ji le priredimo novo vrednost. Funkcije lahko definiramo med izvajanjem programa, prejmejo lahko argumente poljubnega tipa.

Če potrebujemo dodatno funkcionalnost, nam je na voljo vgrajen upravitelj paketov `pip`. Ta nam omogoča prenos in namestitve več kot 160.000 zunanjih modulov, ki se pogosto posodablja.

Interpreter javi napako med izvajanjem in izpiše lokacijo v programu, kjer se je zgodila. Zaradi dobre dokumentiranosti napak in ogromne skupnosti je iskanje rešitev enostavno.

Zaradi relativno visokega nivoja abstrakcije te prednosti pridejo na račun manjše hitrosti. Poleg tega programerju ne omogoča zelo nizkonivajnega nadzora nad izvajanjem (vsaj CPython implementacija). Zaradi tega in GIL je Python neprimeren kot jezik za visoko učinkovito računanje, vsaj v čisti obliki brez klicev v zunanje funkcije.

9.3 Julia

Julia predstavlja zanimivo sredino med C in Pythonom. Zaradi prevajanja v zbirni jezik in optimizacije se v večini situacij po hitrosti približuje C-ju. Hkrati pa so nam na voljo polni izpisi napak in dobra dokumentacija. Jezik izpeljuje tipe, funkcije lahko definiramo med izvajanjem, dopolnjevamo lahko vgrajene module, na voljo je vgrajen upravitelj paketov itd.

Največja hiba jezika trenutno je to, da ni še dokončan. Veliko funkcionalnosti, ki je obljubljena kot del jezika, je še nedokončane ali v eksperimentalnem stanju.

Primer nedokončane funkcionalnosti so niti. Trenutno je makro `@threads` edini način za delitev dela med več niti. V prihodnosti naj bi se spisal vmesnik, ki bo omogočal poljubno zaganjanje niti, ampak trenutno so implementirani le mutex-i, ki pa sami po sebi niso uporabni.

Poleg tega modul `OpenCL.jl` v času pisanja ni podpiral še Julie verzije 1.0. Zaradi tega je bila implementacija JPEG simulacije spisana prek vmesnika CUDA.

10 SKLEP

Julia je v konceptu zelo zanimiv jezik. Prinaša obljubo hitrosti C-ja z enostavnostjo in prijaznostjo Pythona. Večinoma se teh obljub tudi drži, kolikor je le možno.

Njena največja hiba je nedokončanost, kar pa ni prese-netljivo, saj je šele lani dosegla izdajo 1.0. Razvoj jezika je zelo hiter, tako zaradi fleksibilnosti jezika samega kot tudi vztrajno povečujoče se skupnosti.

Navdušenost nad jezikom Julia ni neupravičeno. Prinaša svež pristop do računanja, modularen, prijazen in hiter. Potrebuje le vztrajnost razvijalcev in čas.

LITERATURA

- [1] The Julia Language. [Online] Dosegljivo: <https://www.julialang.org/>. [Dostopano 13. januar 2019].
- [2] Julia Micro-Benchmarks. [Online] Dosegljivo: <https://www.julialang.org/benchmarks/>. [Dostopano 13. januar 2019].
- [3] Metaprogramming. *The Julia Language*. [Online] Dosegljivo: <https://docs.julialang.org/en/v1/manual/metaprogramming/index.html>. [Dostopano 13. januar 2019].
- [4] Multi-Threading (Experimental). *The Julia Language*. [Online] Dosegljivo: [https://docs.julialang.org/en/v1/manual/parallel-computing/index.html#Multi-Threading-\(Experimental\)](https://docs.julialang.org/en/v1/manual/parallel-computing/index.html#Multi-Threading-(Experimental)). [Dostopano 13. januar 2019].
- [5] hackguy. (2018). Threading issues. *JuliaLang (discourse)*. [Online] Dosegljivo: <https://discourse.julialang.org/t/threading-issues/18758>. [Dostopano 13. januar 2019].
- [6] TIOBE Index for January 2019. *TIOBE Index*. [Online] Dosegljivo: <https://www.tiobe.com/tiobe-index/>. [Dostopano 13. januar 2019].
- [7] Pierre Carbone (2018). PYPL PopularitY of Programming Language. [Online] Dosegljivo: <http://pypl.github.io/PYPL.html>. [Dostopano 13. januar 2019].
- [8] The Python Package Index. [Online] Dosegljivo: <https://pypi.org/>. [Dostopano 13. januar 2019].
- [9] Global Interpreter Lock. *Python Wiki*. [Online] Dosegljivo: <https://wiki.python.org/moin/GlobalInterpreterLock>. [Dostopano 13. januar 2019].