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 deterministične 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 nizkonivojnega 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 jezika C 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,[4] refleksijo, makre in večmetodnost funkcij.

V času, ko sva pisala dokument, je jezik že dosegel različico 1.0, a kljub temu še ni popolnoma stabilen. Kot je opisano pozneje v poročilu, ostaja še velik del obljubljene funkcionalnosti neimplementirane, medtem ko ima stabilnost in hitrost obstoječe kode višjo prioriteto.

*http://llvm.org/

3 Sintaksa

Razvijalci Julie prihajajo s področja računske teorije in matematike[3]. Želeli so ohraniti MATLAB-ovo prijaznost do uporabnika glede vnašanja matematičnih funkcij, hkrati pa vpeljati splošnost jezika Python.

Primer funkcije v jeziku Julia:

```
function onePassSwaps!(src::AbstractArray{T,1}
                              where T<:Number,
                       pivot::Number)

    srcSize = size(src)[1]
    i = 0
    j = srcSize + 1

    while i < j

        i += 1
        while i <= srcSize && src[i] < pivot
            i += 1
        end

        j -= 1
        while j > 0 && src[j] > pivot
            j -= 1
        end

        if i >= j
            return (j, srcSize - j)
        end

        src[i], src[j] = src[j], src[i]

    end
end
```

Funkcija je deklarirana kot $function\langle ime\rangle(\langle parametri\rangle)$. Ime v tem primeru vsebuje klicaj. Ta po dogovoru označuje, da funkcija spreminja argumente (sicer pa ni potreben). Parametrom so določeni tipi, kar ni nujno, ampak omogoča prevajalniku, da izvede dodatne optimizacije, ki na parametrih splošnega tipa niso možne. Določeni tipi

so generični, njihovi parametri se nahajajo v zavitih oklepajih.

Deklaraciji sledi telo funkcije, ki se konča z rezervirano besedo *end*. Ukazi skoraj enaki kot v jeziku MATLAB, le da se v Juliji indeksira z oglatimi oklepaji, okrogli oklepaji kot pripona so rezervirani za klice funkcij. Poleg tega se indeksi v Juliji tako kot v MATLAB-u začnejo z 1.

Podpičja na koncu ukazov niso nujna. V jeziku MATLAB se uporabljajo za preprečevanje izpisovanja rezultatov ukazov, v Juliji pa se rezultati ukazov v funkcijah ne izpisujejo, zato so za ta namen nepotrebna.

Bloki ukazov se v v jeziku C označujejo z zavitimi oklepaji, v Pythonu pa z zamikom v desno. Julia za ta namen uporablja rezervirani besedi *begin* in *end*. Beseda *begin* ni vedno nujna, na primer pri stavkih *if*, *for* in *while* je dovolj le *end* na koncu bloka.

Funkcija lahko podobno kot v jeziku Python vrača različne tipe vrednosti glede na dane parametre. V zgornjem primeru vrne par dveh vrednosti, določen z okroglimi oklepaji.

Ista funkcija v jeziku Python:

```
def onePassSwaps (src: np.ndarray, pivot):

    i = -1
    j = src.size

    while i < j:

        i += 1
        while i < src.size and src[i] < pivot:
            i += 1

        j -= 1
        while j >= 0 and src[j] > pivot:
            j -= 1

        if i >= j:
            return (i, src.size - i)

        src[i], src[j] = src[j], src[i]
```

Razlike so minimalne, le indeksi, bloki in deklaracija funkcije so različni.

Ista funkcija v jeziku C:

```
void onePassSwaps(int id) {

    int left_border = starts[id];
    int right_border = ends[id];

    int left_index = left_border - 1;
    int right_index = right_border + 1;

    while (left_index <= right_index) {

        left_index++;
        while (table[left_index] <= pivot[id] &&
               left_index <= right_border)
            left_index++;

        right_index--;
        while (table[right_index] > pivot[id] &&
               right_index >= left_border)
            right_index--;

        if (left_index < right_index) {
            long long mem = table[left_index];
            table[left_index] =
                table[right_index];
            table[right_index] = mem;
        }
    }
    split[id] = right_index + 1;
}
```

C obravnava tabele popolnoma drugače kot Python in Julia, kar se odraža v zelo različnem izgled kode. V tem primeru se podatki nahajajo v globalnih tabelah *starts*, *ends*, *table*, ...

Vsaka deklaracija spremenljivke vključuje njen tip, vsak ukaz se zaključi s podpičjem. Pogoji v stavkih *for*, *if* in *while* morajo biti zaprti v okroglih oklepajih. Vsak blok je zaprt v zavitih oklepajih.

4 PORAZDELJENO RAČUNANJE

Julia podpira več oblik delitve dela. Osredotočila sva se na metode podobne tistim, ki smo jih obravnavali pri predmetu Porazdeljeni sistemi: eksplicitna delitev dela med procesorska jedra in pošiljanje dela koprocesorjem - 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.

4.1 Večnitenje

Večnitenje je v Juliji eksperimentalna funkcija.

Uporabnost je bila v času pisanja zelo omejena, saj sta bila na voljo le makra `@threads`, ki na več nitih paralelno izvede `for` zanko, in `@threadcall`, ki funkcijo v jeziku C pokliče v ločeni niti.[5] Druga možnost zahteva pisanje dela programa v drugem programskem jeziku, vendar sva se ji izognila. 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 in razdeli delo, ki ga 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.[6]

4.2 Večprocesnost

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

5 PRIMERJAVA Z JEZIKOM PYTHON

Med dinamičnimi programskimi jeziki po popularnosti trenutno kraljuje Python.[7][8] Je anekdotno eden najlažjih jezikov za popolne začetnike, omogoča pa tudi reševanje kompleksnejših problemov s širokim naborom javno dostopnih knjižnic - modulov.[9]

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 jeziku C.

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.

5.1 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)[10] 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.

6 PRIMERJAVA Z JEZIKOM C/C++

C je učinkovit nizkonivojski programski jezik, katerega zansova leži zelo blizu strojne kode. Njegov naslednik C++, ki se je razvil kot njegova razširitev, ohranja hitrost in prednost, da se prevede v strojni jezik, kar omogoča, da program izvaja bistveno hitreje od ostalih programskih jezikov. Prav tako dopušča precej svobode pri ravnanju s pomnilnikom pri čemer lahko dober programer to izkoristi v svoj prid. Slabost tega je, da moramo biti zelo pazljivi pri pisanju, če se hočemu izogniti mučnemu in dolgotrajnemu razhroščevanju.

Omeniti moramo tudi vrsto različnih knjižnic in API-jev, ki uporabniku olajšajo delo in široko podporo, ki jo uživa C++. Vendar kljub vsem prednostim, pa delo z njim včasih težavno, saj moramo večino dela postoriti sami, medtem ko so višjenivojske funkcije in udobnost programiranja stalnica pri nekaterih drugih jezikih.

*https://docs.python.org/3.7/library/multiprocessing.html
†https://docs.python.org/3.7/library/threading.html

6.1 pthreads

Pri C++ imamo velik izbor knjižnic za paralelno procesiranje in nitenje. Nekatere omogočaje več svobode pri upravljanju z njimi, druge pa delitev dela poenostavijo in optimizirajo.

POSIX Threads ali krajše pthreads so nizkonivojski API za delo z nitmi. Dopušča nam veliko nadzora in modularnosti za ceno nekoliko kompleksnejše implementacije. Pazljivi moramo biti predvsem z konfliktnimi dostopi in sinhronizacijo. Uporabniku omogoča nadzor nad nitjo (kreiranje, uničevanje, odklapljanje,...), upravljanje ključavnic, sinhronizacijo niti z uporabo zapornic in pogojne spremenljivke.

6.2 OpenMP

Višjenivosjki OpenMP (Open Multi-Processing) predstavlja nekoliko drugačen način razporejanja dela.

Je zbirka direktiv prevajalniku in je nekoliko lažje nastavljiv, sintaktično krajši ter bolj enostaven v primerjavi s pthreads. Uporabja posebne oznake (#pragma), ki služijo kot navodila prevajalniku in zagotavlja široko izbiro visoko in nižjenivojskih opcij. Pomembna lastnost pragme je tudi da v kolikor jo prevajalnik ne prepozna, preprosto nadaljuje z izvajanjem programa. Ta možnost omogoča programerju lažje razhroščevanje kot tudi enostavno določanje kaj se bo izvajalo paralelno.

Njegova hitrost v nekaterih primerih prekaša hitrost pthreads ali pa je približno enaka. Poleg preprostosti ponuja zelo zanesljivo in hitro vzporedno procesiranje.

7 ALGORITMA

Za primerjavo jezikov sva implementirala dva algoritma.

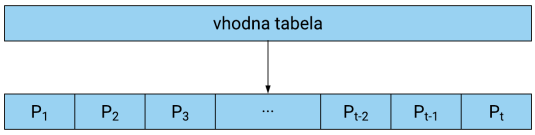
Prvi, Adaptive Quick Sort, je paralelna implementacija sortirnega algoritma Quick Sort. Izvajal se je na vseh razpoložljivih nitih sistema. Zasnovala sva ga tako, da vzporedno izvede čim večji del računanja, tudi deljenje in združevanje podatkov.

Z drugim algoritmom pa sva simulirala JPEG kompresijo. Izvajal se je na grafični kartici prek knjižnic OpenCL in CUDAnative.

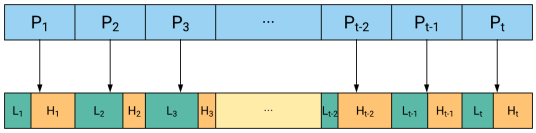
7.1 Adaptive Quick Sort

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.

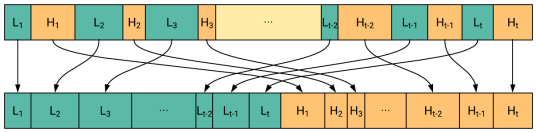
Idejno se vsak korak algoritma izvede v 3 delih. Naj bo "n" število podatkov v tabeli, "t" pa število niti oz. procesov, ki jih ima algoritem na voljo. V prvem delu se tabela razdeli na t delov in določi se pivotni element:



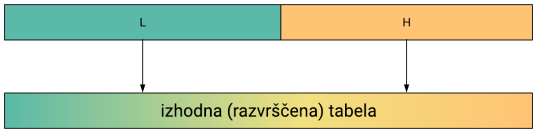
Nato vsaka nit operira nad svojim delom tabele in uredi števila glede na pivot. V L dobimo elemente manjše ali enake pivotu, v H pa elemente večje od pivota:



Po urejanju vsaka nit sodeluje v premiku svojega dela elementov, ki so manjši ali enaki pivotu (L), na levo stran tabele. Cilj koraka je, da niti združijo vse dele L in D tako, da ima tabela na levi vse L dele ter na desni vse D dele. To pomeni, da bo pivot delil celotno tabelo na elemente, ki so manjši ali enaki pivotu in elemente, ki so večji od pivota. Zaradi pomnjenja odmika vsakega izmed delov lahko vsaka nit natačno izračuna, kam mora vsaka premakniti svoj del L.



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

7.2 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.

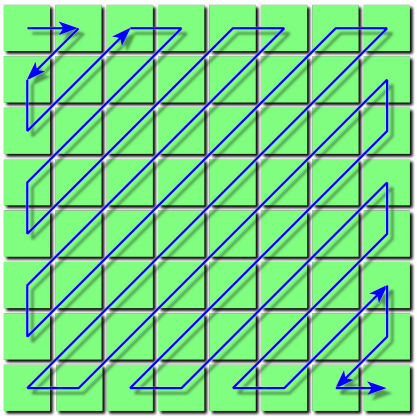
Kodiranje je sestavljeno iz treh glavnih delov, preslikave 8x8 kosov slike v frekvenčno domeno, kvantizacije in končnega stiskanja.

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

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

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

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 hitrostmi prevajanja in izvajanja programov na GPE med različnimi jeziki.

8 Okolje

Testirala sva 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.

Programe v jeziku C/C++ sva prevajala in zaganjala 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 sva je zaganjala v ukaznem pozivu Powershell, Python verzija 3.7.1 64 bit, numpy verzija 1.15.4.

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

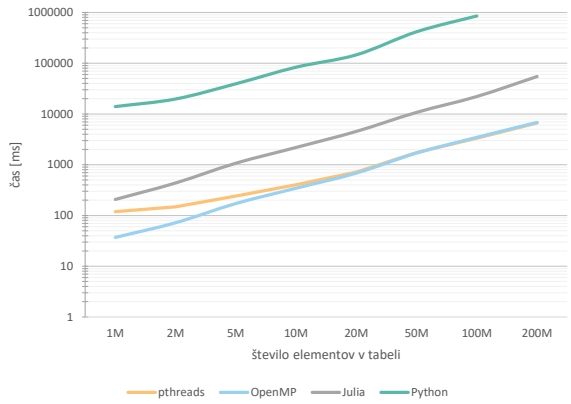
Programe v jeziku Julia sva zaganjala v ukaznem pozivu Powershell, Julia verzija 1.0.3 64 bit, CUDAnative verzija 0.9.1.

9 Analiza hitrosti

9.1 Adaptive Quick Sort

Algoritem očitno izpostavi obseg razlik hitrosti posameznih jezikov in kako velikost podatkov vpliva nanje. Programe sva testirala na eksponento povečujoči tabeli naključno generiranih predznačenih 64-bitnih celih števil. Za vsako velikost tabele sva vsak program izvedla petkrat in po koncu izvajanja izračunala povprečje izmerjenih časov sortiranj. Največja tabela je vsebovala 200 milijonov elementov.

Vsak program sva spisala na način, ki je minimiziral število nepotrebnih dostopov do pomnilnika in hkrati kar najbolje sledil prej opisanemu načrtu. V primeru Pythona in Julie sva bila omejena s strani jezika, 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.

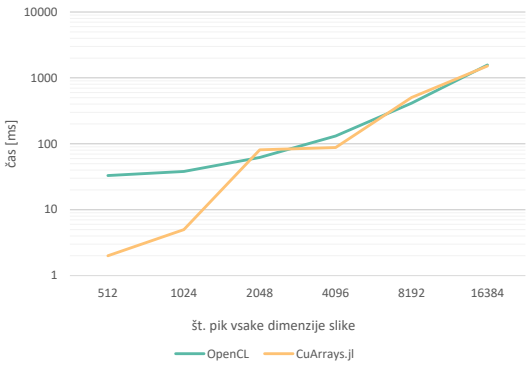
Julia je le slabo magnitudo počasnejša. Za majhne tabele je skoraj tako hitra kot pthreads. Med 2 in 20 milijoni elementov počasi izgublja hitrost. Pri 200 milijonih pa nastopi težava s količino pomnilnika, zato se izvajanje še dodatno upočasni.

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.

9.2 JPEG

Za razliko od sortirnega algoritma sta tu implementaciji skoraj enakovredni. Programa sva testirala na čedalje večjih kvadratnih slikah, velikosti dimenzij so bile naraščajoče potence števila 2. Za vsako velikost sta se programa najprej dvakrat zagnala brez merjenja, kar odpravi počasen začetek izvajanja, nato pa je sledilo 5 meritev. Končne vrednosti meritev sva povprečila.

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

Implemetacija 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.

10 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.

10.1 C/C++

C/C++ sta znana po svoji hitrosti in široki uporabnosti.

Večina jezikov deluje podobno in so bili modelirani prav po njiju. Medtem ko C podpira uporabu struktur in predstavlja surovo osnovo je C++ objektno orientiran ter je njegova nadgradnja in razširitev.

Izjemna podpora, kopica knjižnic in razširitev, služijo kot orodje za lažje delo. Zaradi manjše abstrakcije v primerjavi z javo in pythonom je včasih programer prikrajšan na času in preprostosti kode.

V kolikor smo pripravljeni vložiti trud za kompleksnejšo rešitev problema lahko veliko pridobimo z optimizacijo in posledično dosežemo večje hitrosti pri izvajanju.

10.2 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 namestitev več kot 160.000 zunanjih modulov, ki se pogosto posodabljajo.

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 nizkonivojnega 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.

10.3 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 defeniramo med izvajanjem, dopolnjujemo lahko vgrajene module, na voljo je vgrajen upravitelj paketov itd.

Največja hiba jezika trenutno je to, da ni še dokončan. Veliko funkcionalnosti, ki so obljubljene kot del jezika, je še nedokončanih 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. Zaragi tega sva se odločila implementacijo JPEG simulacije spisati prek vmesnika CUDA.

11 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 presenetljivo, 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 neupravičena. Prinaša svež pristop do računanja, ki je modularen, prijazen in hiter. Potrebuje le vztrajnost razvijalcev in čas.

LITERATURA

[1] The Julia Language. [Online] Dosegljivo: https://www.julialang.org/. [Dostopano 24. januar 2019].

[2] Julia Micro-Benchmarks. [Online] Dosegljivo: https://www.julialang.org/benchmarks/. [Dostopano 24. januar 2019].

[3] Why We Created Julia. [Online] Dosegljivo: https://julialang.org/blog/2012/02/why-we-created-julia. [Dostopano 24. januar 2019].

[4] Metaprogramming. *The Julia Language*. [Online] Dosegljivo: https://docs.julialang.org/en/v1/manual/metaprogramming/index.html. [Dostopano 24. januar 2019].

[5] Multi-Threading (Experimental). *The Julia Language*. [Online] Dosegljivo: https://docs.julialang.org/en/v1/manual/parallel-computing/index.html#Multi-Threading- (Experimental). [Dostopano 24. januar 2019].

[6] hackguy. (2018). Threading issues. *JuliaLang (discourse)*. [Online] Dosegljivo: https://discourse.julialang.org/t/threading-issues/18758. [Dostopano 24. januar 2019].

[7] TIOBE Index for January 2019. *TIOBE Index*. [Online] Dosegljivo: https://www.tiobe.com/tiobe-index/. [Dostopano 24. januar 2019].

[8] Pierre Carbonnelle (2018). PYPL PopularitY of Programming Language. [Online] Dosegljivo: http://pypl.github.io/PYPL.html. [Dostopano 24. januar 2019].

[9] The Python Package Index. [Online] Dosegljivo: https://pypi.org/. [Dostopano 24. januar 2019].

[10] Global Interpreter Lock. *Python Wiki*. [Online] Dosegljivo: https://wiki.python.org/moin/GlobalInterpreterLock. [Dostopano 24. januar 2019].