Projektne naloge

Porazdeljeni sistemi

Uroš Lotrič, Davor Sluga

Jesen 2022

Splošne informacije

V nadaljevanju dokumenta so opisane štiri različne projektne naloge. Pričakujeva, da boste projektne naloge delali v trojicah. Za vsako drugačno obliko dela se morate dogovoriti z nama. Vsako nalogo si lahko izberejo največ štiri skupine. Izbor nalog bo potekal preko učilnice po pravilu kdor prej pride, prej melje.
Za pozitivno oceno iz projektne naloge:

- pripravite sekvenčno različico algoritma, ki bo služila za primerjavo;
- pripravite paralelno različico algoritma s tehnologijami, predlaganimi pri posamezni nalogi;
- naredite primerjalne teste na superračunalniku NSC;
- pripravite 10-minutno predstavitev, ki opisuje vašo rešitev, rezultate meritev, kot so čas izvajanja, pohitritve, učinkovitosti, in glavne ugotovitve.
- predstavitev in izvorno kodo oddajte preko učilnice.

Nekaj idej za boljšo oceno:

- pripravite izvedbo algoritma za tehnologijo, ki ni eksplicitno navedena pri nalogi;
- izpeljite teoretični model paralelnega algoritma;
- predlagajte in preizkusite dodatne optimizacije algoritma.

Predstavitve projektov bodo v torek, **10. 1. 2023**, popoldan, v času predavanj.

1 Veriženje blokov

Sistem razpršenega veriženje blokov omogoča varno in učinkovito upravljanje seznama lastnikov nečesa digitalnega, ne da bi za to potrebovali osrednjo monopolno agencijo, ki bi ji morali vsi zaupati [1]. Ideja, kako strukturno preprečiti goljufanje in nekontrolirano kopiranje digitalne lastnine (denarja), temelji na dokazu opravljenega dela, ki ga sistem zahteva na nekaj minut. To opravljeno delo mora biti tako veliko, da praktično ni mogoče goljufati, saj bi moral potencialni goljuf opraviti bistveno več dela, kot vsi drugi uporabniki sistema skupaj. Protokol usklajevanja, ki temelji na prikazu opravljenega dela, izhaja iz izvajanja zapletenih računskih nalog, ki zahtevajo veliko računske moči, njihovo rešitev pa je enostavno preveriti in tako potrditi, da je bilo delo dejansko opravljeno in ni prišlo do prevare.

Ključen element usklajevanja je preverjanje istovetnosti podatkov. To se izvaja preko digitalnega prstnega odtisa ali zgoščene vrednosti (angl. hash), ki poljubno dolgemu sporočilu priredi zaporedje znakov nespremenljive dolžine. Z vsako, še tako majhno spremembo podatkov v sporočilu, se spremeni tudi njihova zgoščena vrednost.

Pri veriženju blokov se sporočila o transakcijah med uporabniki oblikujejo v bloke. Vsi, ki sodelujejo pri veriženju blokov, sledijo novim sporočilom o izvedenih transakcijah in na podlagi zgoščene vrednosti zadnjega veljavnega bloka, predloga novega bloka in nekaj naključnih dodanih znakov ali skovanke, izračunavajo novo zgoščeno vrednost. Z dodajanjem poljubno dolge skovanke lahko zgoščeno vrednost novega bloka spreminjamo tako, da ustreza predpisanim zahtevam. Ko najdemo ustrezno zgoščeno vrednost, predlog bloka, v katerem so zabeležena nova sporočila o transakcijah, razpošljemo drugim uporabnikom v odobritev. Uporabniki sistema glasujejo za veljavnost novega bloka tako, da ga po preverjanju dodajo v verigo prejšnjih, že potrjenih. Za dopolnitev verige dobimo nagrado, nato se zgodba začne od začetka.

1.1 Zgoščena vrednost in dokaz opravljenega dela

Za sporočila, sestavljena iz niza znakov, bomo zgoščene vrednosti računali z algoritmom MD5 [2]. Algoritem MD5 [3] za niz znakov vrne zgoščeno vrednost dolžine 16 znakov. Zgoščeno vrednost nato predstavimo z 32 znaki šestnajstiške kode in preštejemo s koliko ničlami se konča. Na primer, algoritem MD5 za vhodne znake $01020304C2AAD20D_{\text{hex}}$ izračuna zgoščeno vrednost $B18F3F414358E488AEFABE8010000000_{\text{hex}}$ s 7 ničlami na koncu.

Osredotočili se bomo na dokazovanje opravljenega dela. Na vhodu pričakujemo množico sporočil različnih dolžin, na primer $01020304C2AAD2_{\text{hex}}$, $01020304C2_{\text{hex}}$, 01020304_{hex}. Vsako sporočilo posebej dopolnimo s skovanko, da dobimo zgoščeno vrednost z zahtevanim številom ničel na koncu, na primer 7. Za navedena sporočila bomo z intenzivnim preiskovanjem poiskali skovanke $0D_{\text{hex}}$, $AAD20D_{\text{hex}}$ in $C2AAD20D_{\text{hex}}$.

Kodo v jeziku C za iskanje skovanke za eno sporočilo najdete v [4]. Koda uporablja funkcije MD5 za postopno računanje zgoščene vrednosti iz knjižnice OpenSSL.

1.2 Naloga

Ker želimo dobiti veliko nagrad, bomo iskanje zgoščenih vrednosti računali vzporedno. Sporočila obdelujemo zaporedno, enega za drugim. Najti želimo najkrajšo skovanko, s katero dobimo zahtevano zgoščeno vrednost, zato skovanke podaljšujemo postopno, znak po znak. Najprej preverimo osnovno sporočilo. Če ne vrne ustrezne zgoščene vrednosti, preverimo prostor eno-znakovnih skovank. Če sporočilo z nobeno skovanko ne da zahtevane zgoščene vrednosti, začnemo preiskovati prostor dvo-znakovnih skovank. Skovanke podaljšujemo, dokler ne najdemo ustrezne skovanke ali presežemo največjo dovoljeno dolžino.

Za računanje skrbi koordinator, ki sprejema sporočila in razdeljuje delo delavcem. Predpostavite lahko, da je število delavcev omejeno na 256. Tako koordinator delo razdeli samo glede na prvi znak skovanke. Pri štirih delavcih bi prvemu dodelili preverjanje skovank s prvim znakom $00..3F_{\text{hex}}$, drugemu skovanke, ki se začnejo z znaki $40..7F_{\text{hex}}$, tretjemu skovanke s prvimi znaki $80..BF_{\text{hex}}$ in zadnjemu skovanke s prvimi znaki $C0..FF_{\text{hex}}$.

Takoj, ko delavec najde rešitev, mora o tem obvestiti koordinatorja, ki pošlje zahtevo za prekinitev iskanja ostalim delavcem. Ko vsi delavci prekinejo z iskanjem, jim koordinator dodeli naslednjo nalogo.

V osnovni različici je število delavcev znano v naprej. Predpostavite, da je sistem stabilen – koordinator in delavci vestno opravljajo svojo nalogo, z omrežjem ni težav.

Rešitev zasnujte večnitno z uporabo knjižnice Pthread in porazdeljeno s knjižnico MPI ali pa uporabite hibridni pristop s pomočjo obeh knjižnic. Primerjajte čase sekvenčnega preiskovanja z večnitno in porazdeljeno rešitvijo. Pri programiranju si lahko pomagate s koncepti iz navodil [5; 6].

Nekaj idej:

- Kako se čas računanja, pohitritev in učinkovitost spreminjajo s številom zahtevanih ničel na koncu zgoščene vrednosti?
- Izmerite čas, ki poteče od takrat, ko eden od delavcev najde rešitev, do takrat, ko začnejo obdelovati naslednje sporočilo.
- Sistem dopolnite tako, da delo dobijo tudi delavci, ki v sistem pristopijo med iskanjem rešitve za neko sporočilo. Uporabite idejo dinamičnega razvrščanja dela, ki smo jo srečali pri OpenMP.
- Problem rešite z nalogami v OpenMP (deli in vladaj). Rešitev primerjajte z vašo rešitvijo s knjižnico Pthread.
- Porazdeljeni sistem (MPI) naredite robusten, da bo našel rešitev tudi v primeru, ko eden od delavcev odpade.
- Dodajte odjemalce, ki pošiljajo delo koordinatorju. Koordinator naj rezultat posreduje pravemu odjemalcu.

2 Odkrivanje robov v slikah

Odkrivanje robov predstavlja enega temeljnih postopkov pri procesiranju slik [7]. Z njegovo pomočjo iz slike izluščimo informacijo o prisotnih strukturah. Uporablja se v postopkih računalniškega vida, kot na primer razpoznavanje objektov, detekcijo voznega pasu, razpoznavanje prstnih odtisov, obdelava medicinskih slik. Razvitih je bilo veliko metod za odkrivanje robov, ena od bolj znanih je Cannyev detektor [8].

2.1 Cannyev detektor

Postopek odkrivanja robov je sestavljen iz petih korakov:

1. **Glajenje oziroma odstranjevanje šuma z Gaussovim filtrom**

   Odkrivanje robov je zelo občutljivo na šum v sliki, zato je potrebno sliko predhodno zgladiti. Sliko najprej pretvorimo v sivinsko, nato pa izvedemo glajenje. Običajno se uporablja Gaussov filter

   $$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

   z velikostjo okna 5x5, ki deluje dobro v večini primerov. Parameter σ izberemo na intervalu $[1,2]$.

2. **Računanje gradienta slike.**

   V tem koraku izračunamo odvod intenzitete v sliki v navpični in vodoravni smeri. To naredimo s pomočjo Sobelovih operatorjev, tako da z njimi naredimo konvolucijo nad sliko.

   $$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I,$$

   kjer je $*$ operator konvolucije v dveh dimenzijah, I vhodna slika, G_x in G_y pa sliki, ki vsebujeta približek gradienta za vsako točko slike I v vodoravni (G_x) in navpični (G_y) smeri. Na podlagi G_x in G_y lahko v vsaki točki izračunamo magnitudo $G = \sqrt{G_x^2 + G_y^2}$ in smer gradienta intenzitete $\Theta = \arctan2(G_x, G_y)$.

3. **Tanjšanje robov**

   Po drugem koraku se na sliki magnitud gradienta vidijo posamezni robovi, vendar so precej razmazani. V tem koraku poskusimo robove stanjšati in ohraniti le lokalne maksimume. To naredimo s pomočjo metode izločanja neizrazitih slikovnih točk (angl. Non-Maximum suppression).

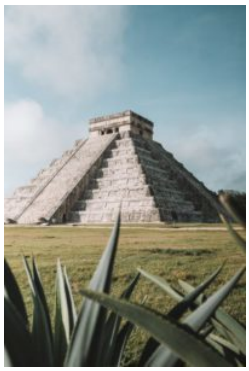
4. **Dvojno upragovanje**

   Po tretjem koraku smo dobili občutno tanjše robove, vendar je v sliki še vedno precej razlik v intenziteti posameznih robov. To bomo odpravili z zadnjima dvema

korakoma. S pomočjo dvojnega upragovanja razvrstimo slikovne točke v tri kategorije: izrazite, šibke in nepomembne. Izberemo si dve mejni vrednosti, visoko in nizko. Slikovne točke z magnitudo gradienta višjo od visoke mejne vrednosti označimo kot izrazite, slikovne točke med obema mejnima vrednostima kot šibke, ostale pa kot nepomembne. Mejni vrednosti določimo empirično.

5. **Sledenje robovom s pomočjo histereze**

   Izrazite točke, ki smo jih dobili v prejšnjem koraku, bodo del robov v izhodni sliki, medtem ko bomo nekatere izmed šibkih slikovnih točk v tem koraku spremenili v izrazite po naslednjem merilu. Za vsako šibko slikovno točko pogledamo njeno neposredno okolico (8 sosednjih slikovnih točk), če je katerakoli od sosednjih slikovnih točk izrazita, spremenimo trenutno šibko točko v izrazito sicer pa točko označimo za nepomembno. Izhodno sliko sestavljajo izrazite točke, ki jim nastavimo intenziteto na najvišjo možno vrednost. Ostalim slikovnim točkam pa nastavimo intenziteto na 0.



(a) Vhodna slika.

(b) Izhodna slika.

Slika 1: Odkrivanje robov v sliki s Cannyevim detektorjem.

Podroben opis posameznik korakov in izvorno kodo v jeziku Python najdete na povezavi. Za branje in zapisovanje slik lahko uporabite ustrezno knjižnico, na primer STB [9] ali FreeImage [10]. V kolikor želite, lahko uporabite knjižnico OpenCV, ki že vsebuje implementacije nekaterih funkcij (e.g. Sobelov operator, branje, pisanje slik, itd.). Primer uporabe OpenCV v jeziku Python za odkrivanje robov najdete na povezavi. Knjižnico OpenCV lahko uporabite v kombinaciji z jezikom C/C++, navodila za namestitev najdete tukaj.

2.2 Naloga

Na voljo imamo večje število slik, v katerih želimo poiskati robove. Postopek želimo čim bolj pohitriti. Iskanje robov v sliki je sestavljeno iz petih zgoraj navedenih korakov

ter branja vhodne ter zapisovanja izhode slike. Izvajanje lahko organiziramo kot cevovod, kjer posamezne stopnje cevovoda predstavljajo posamezne korake odkrivanja robov. Počasnejše stopnje lahko vzporedno izvajamo na več procesorskih jedrih.

Pripravite dve rešitvi ali eno hibridno, ki bodo uporabljale knjižnici Pthread in MPI. Komunikacijo med stopnjami pripravite po konceptu proizvajalec-porabnik. V osnovni različici ni potrebno, da je vrstni red slik na izhodu enak kot na vhodu. Nekaj idej:

- Izmerite čase procesiranja na posameznih stopnjah cevovoda, izračunajte pohitritve, učinkovitost.
- Koncept proizvajalec-porabnik razširite tako, da bo proizvajalec naloge lahko oddajal v medpomnilnik.
- Preverite, koliko pomaga uporaba več vzporednih procesorjev v izbranih stopnjah cevovoda. Lahko obdelujete več slik naenkrat v isti stopnji cevovoda ali pa uporabite več jeder za obdelavo ene slike.
- Za koliko se zmanjšajo pohitritve, če zahtevate, da je vrstni red slik na izhodu enak tistemu na vhodu (obdelava videa)?
- Program izboljšajte tako, da število vzporednih procesorjev na vsaki stopnji določate dinamično, glede na potrebe.

3 Modeliranje rasti snežnih kristalov

Snežni kristali lahko nastopajo v zelo različnih oblikah z značilno šesterokotno simetrijo. Dvodimenzionalne oblike so lahko dendritne, zvezdaste, sektorske in ploščate oblike. Fizikalne študije so pokazale, da je oblika snežnega kristala odvisna od temperature in vlage v rastnem okolju. V nadaljevanju se bomo omejili na preprost dvodimenzionalni model s tremi parametri, s katerimi lahko dosežemo želeno raznolikost kristalov [11].

3.1 Reiterjev model

Reiterjev model je dobro opisan v delu [12]. Model za osnovo uporabi dvodimenzionalno mrežo celic. Vse celice so šestkotne oblike in imajo po šest sosedov (slika 2). V času t se v izbrani celici z nahaja količina vode $s_t(z)$ v enem od agregatnih stanj (led, voda, para). V

Slika 2: Heksagonalna mreža celic [12]

sistemu nastopajo štirje tipi celic: zamrznjene (zelene), mejne (modre), robne (oranžne) in nedovzetne (bele). Celica je zamrznjena, če je $s_t(z) \geq 1$. Mejna celica ni zamrznjena ($s_t(z) < 1$), se pa dotika vsaj ene zamrznjene celice. Zamrznjene in mejne celice so celice, ki so dovzetne za rast kristalov. Robne celice označujejo rob mreže, preko njih v sistem stalno dovajamo vodo za rast kristalov. Nedovzetne celice so vse ostale celice, v katerih kristali ne morejo rasti.

Vodo v celici ločimo na del $u_t(z)$, ki se spontano razširja med celicami (difuzija), in na del $v_t(z)$, ki se ne vključuje v difuzijske procese,

$$s_t(z) = u_t(z) + v_t(z) \quad . \tag{1}$$

Zamrznjena voda iz dovzetnih celic (zamrznjenih in mejnih) ne more prehajati v ostale celice (ni difuzije), zato je $u_t(z) = 0$ in $v_t(z) = s_t(z)$. V nedovzetnih celicah se v difuzijske procese vključuje vsa voda, zato velja $u_t(z) = s_t(z)$ in $v_t(z) = 0$.

Po Reiterjevem modelu v dovzetnih celicah voda hitro zamrzne, zato se količina vode v njih nikoli ne zmanjšuje. Dovzetne celice lahko nase pritegnejo tudi nekaj vode izven

ravnine, v kateri raste kristal. Predpostavimo, da vsaka dovzetna celica lahko nase pritegne konstantno količino vode γ. Voda iz nedovzetnih celic prosto prehaja iz celic z večjo količino v celice z manjšo količino po difuzijski enačbi, $\partial u/\partial t = a\nabla^2 u$. Voda iz nedovzetnih celic lahko prehaja v dovzetne celice, obratno ne.

Po zgornjih pravilih stanje dovzetnih in nedovzetnih celic z v času t računamo po enačbi

$$s_{t+1}(z) = s_t(z) + \frac{\alpha}{2}\left(\frac{\sum_{z' \in N(z)} u_t(z')}{6} - u_t(z)\right) + \gamma D(z) \quad , \tag{2}$$

v kateri $N(z)$ predstavlja vseh šest sosedov celice z, za dojemljivost $D(z)$ pa velja

$$D(z) = \begin{cases} 1 & \text{dovzetne celice} \\ 0 & \text{ostale celice} \end{cases} \quad . \tag{3}$$

Prvi člen enačbe (2) je diskretna verzija difuzijske enačbe za šestkotno mrežo. Iz enačbe sledi, da celica z ohrani $(1-\alpha/2)u_t(z)$ vode in med sosednje celice enakomerno porazdeli $\alpha/2 \cdot u_t(z)$ vode. Ker hkrati od vsake sosednje celice z' prejme $\alpha/12 \cdot u_t(z')$ vode, se skupna količina vode v sistemu ohranja.

Simulacijo začnemo tako, da v sredino mreže postavimo eno zamrznjeno celico, vsem ostalim celicam pa damo začetno ne-ničelno količino vode (β),

$$s_0(z) = \begin{cases} 1 & \text{srednja celica} \\ \beta & \text{ostale celice} \end{cases} \quad . \tag{4}$$

Količina vode v robnih celicah je ves čas simulacije enaka β.

S spreminjanjem parametrov α, β in γ lahko oblikujemo različne vzorce snežnih kristalov, ki jih najdemo v naravi. Množico parametrov in pripadajočih vzorcev snežnih kristalov najdemo v literaturi [11]. Na voljo je tudi spletna simulacija rasti kristalov [13].

3.2 Naloga

Pripravite vzporedno verzijo programov za sisteme s skupnim (Pthread) in sisteme s porazdeljenim (MPI) pomnilnikom. Rešitev mora delovati za poljubno velikost mreže in za poljubno število procesorjev. Nekaj idej:

- Primerjajte vzporejanje s knjižnicama Pthread in OpenMP.
- Analizirajte vpliv delitve (horizontalni pasovi, vertikalni pasovi, bloki) na izvajanje programa.
- Za zakrivanje latence pri prenosu podatkov se splača med procesi izmenjati več kot samo robne celice in zaradi tega nekaj več poračunati. Analizirajte vpliv širine pasov na izvajanje programa.
- Poskrbite za učinkovito shranjevanje slik rasti po vsakem k-tem koraku simulacije, mogoče MPI-IO. Iz slik sestavite videoposnetek.

- Ločeno ocenite stroške komunikacije in stroške računanja.
- Za določitev blokov, ki se prenašajo med procesi, uporabite podatkovne tipe MPI.
- Za prenašanje podatkov med vozlišči uporabite knjižnico MPI, za računanje na vozlišču pa knjižnico OpenMP.
- Uporabite asinhrono prenašanje podatkov.

4 Problem n teles

Mnoge probleme v fiziki in kemiji lahko rešimo s proučevanjem interakcij med vsemi telesi v sistemu. Na področju molekulske dinamike je na primer zanimivo obnašanje molekul v Lennard-Jonesovem potencialu, na področju astronomije pa vpliv gravitacijskih sil na gibanje teles. Ker računamo interakcije vseh teles na izbrano telo, je računska kompleksnost osnovnega sekvenčnega algoritma $O(n^2)$ za vsak časovni korak, kjer je n število vseh teles v sistemu. Kljub temu, da obstaja mnogo izboljšav osnovnega algoritma, se bomo v nadaljevanju omejili na vzporejanje osnovnega algoritma.

4.1 Gibanje nebesnih teles

Vzemimo, da imamo v tridimenzionalnem prostoru n teles. Položaj telesa i v kartezičnem koordinatnem sistemu zapišemo kot $\mathbf{r}_i = (x_i, y_i, z_i)$, njegovo hitrost pa kot $\mathbf{v}_i = (v_{ix}, v_{iy}, v_{iz})$.

Težišči teles i in j lahko povežemo z vektorjem

$$\mathbf{r}_{ij} = (x_{ij}, y_{ij}, z_{ij}) = (x_j - x_x, y_j - y_i, z_j - z_i) \quad , \tag{5}$$

ki določa razdaljo med telesoma, $r_{ij} = |\mathbf{r}_{ij}| = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2}$.

Telo j deluje na telo i s silo F_{ij}, telo i pa na telo j s silo F_{ji}, pri čemer velja

$$\mathbf{F}_{ij} = -\mathbf{F}_{ji} = \kappa \frac{m_i m_j}{|\mathbf{r}_{ij}|^2} \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \quad , \tag{6}$$

kjer je κ gravitacijska konstanta. Silo telesa j na telo i lahko zapišemo po komponentah

$$\mathbf{F}_{ij} = (F_{ijx}, F_{ijy}, F_{ijz}) = \left(\kappa \frac{m_i m_j}{r_{ij}^2} \frac{x_{ij}}{r_{ij}}, \kappa \frac{m_i m_j}{r_{ij}^2} \frac{y_{ij}}{r_{ij}}, \kappa \frac{m_i m_j}{r_{ij}^2} \frac{z_{ij}}{r_{ij}} \right) \quad . \tag{7}$$

Gibanje telesa i je odvisno od sile vseh ostalih teles nanj,

$$\mathbf{F}_i = \sum_{j \neq i} \mathbf{F}_{ij} \quad . \tag{8}$$

Pri numeričnem računanju sile predpostavimo, da je masa telesa zbrana v poljubno majhni točki. Ker se zato lahko zgodi, da sta dve telesi hkrati na isti točki, nam zahteva $j \neq i$ precej upočasni simulacijo. Zato razdaljo r_{ij} v izračunu sile raje nadomestimo z

$$R_{ij}^2 = r_{ij}^2 + \varepsilon^2 \tag{9}$$

kjer je ε poljubno majhna konstanta. Tako se ognemo deljenju z nič, ko je $j = i$, še vedno pa velja $\mathrm{F}_{ii} = \mathbf{0}$. Zdaj lahko zapišemo

$$\mathbf{F}_i \approx \left(\kappa m_i \sum_{j=1}^{n} m_j \frac{x_{ij}}{R_{ij}^3}, \kappa m_i \sum_{j=1}^{n} m_j \frac{y_{ij}}{R_{ij}^3}, \kappa m_i \sum_{j=1}^{n} m_j \frac{z_{ij}}{R_{ij}^3} \right) \quad . \tag{10}$$

Ko poznamo skupno silo vseh teles na telo i, $\mathbf{F}_i = (F_{ix}, F_{iy}, F_{iz})$, lahko za telo i izračunamo njegov nov položaj in novo hitrost. Najprej za telo i določimo pospešek,

$$\mathbf{a}_i = (a_{ix}, a_{iy}, a_{iz}) = \left(\frac{F_{ix}}{m_i}, \frac{F_{iy}}{m_i}, \frac{F_{iz}}{m_i},\right) \quad . \tag{11}$$

Zdaj s preprosto integracijo najprej nov položaj $\mathbf{r}_i = (x_i, y_i, z_i)$,

$$(x_i, y_i, z_i) \leftarrow \left(x_i + v_{ix}dt + \frac{1}{2}a_{ix}dt^2, y_i + v_{iy}dt + \frac{1}{2}a_{iy}dt^2, z_i + v_{iz}dt + \frac{1}{2}a_{iz}dt^2\right) \quad , \tag{12}$$

nazadnje pa še hitrost $\mathbf{v}_i = (v_{ix}, v_{iy}, v_{iz})$,

$$(v_{ix}, v_{iy}, v_{iz}) \leftarrow (v_{ix} + a_{ix}dt, v_{iy} + a_{iy}dt, v_{iz} + a_{iz}dt) \quad . \tag{13}$$

4.2 Naloga

Pripravite program za sekvenčno simulacijo obnašanja sistema več nebesnih teles. Gravitacijsko kontantko κ in mase teles nastavite tako, da bo potek simulacije zanimiv. Program prilagodite za učinkovito računanje na sistemih s skupnim in sistemih s porazdeljenim pomnilnikom. Rešitev mora delovati za poljubno število teles in za poljubno število procesorjev. Nekaj idej:

- Primerjajte vzporejanje s knjižnicama Pthread in OpenMP.
- Problem lahko razdelite med procesorje tako, da vsak računa skupno silo na izbrano število teles. Ker pa velja $\mathbf{F}_{ij} = -\mathbf{F}_{ij}$, lahko število izračunov prepolovite. Primerjajte obe rešitvi.
- Poskrbite za učinkovito shranjevanje položaja teles po vsakem k-tem koraku simulacije, mogoče MPI-IO in izdelajte film.
- Pri MPI preverite ali je podatke o telesih bolje organizirati v polje struktur (podatkov o telesih) ali v strukturo z več polji (eno za položaj, drugo za hitrost vseh teles).
- Raziščite uporabo podatkovnih tipov MPI.
- Za prenašanje podatkov med vozlišči uporabite knjižnico MPI, za računanje na istem vozlišču pa knjižnico OpenMP.
- Uporabite asinhrono prenašanje podatkov.

Literatura

[1] Kvarkadabra, Kaj je blockchain?, https://kvarkadabra.net/2016/10/kaj-je-blockchain/, 2021.

[2] MD5, https://en.wikipedia.org/wiki/MD5, 2021.

[3] Openwall, A portable, fast, and free implementation of the MD5 Message-Digest Algorithm (RFC 1321), https://openwall.info/wiki/people/solar/software/public-domain-source-code/md5, 2021.

[4] Lotrič, U., Iskanje skovanke za dano sporočilo, https://ucilnica.fri.uni-lj.si/mod/resource/view.php?id=47001, UL FRI, 2021.

[5] Beschastnikh, I., 416 Distributed Systems: Assignment 2 [Proof-of-work], https://www.cs.ubc.ca/~bestchai/teaching/cs416_2020w2/assign2/index.html, University of British Columbia, 2021

[6] Beschastnikh, I., 416 Distributed Systems: Assignment 3 [Distributed proof-of-work], https://www.cs.ubc.ca/~bestchai/teaching/cs416_2020w2/assign3/index.html, University of British Columbia, 2021.

[7] Ziou, Djemel and Tabbone, Salvatore and others, Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii, Vol. 8, pp 537–559, 1998.

[8] Canny, John, A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence, Vol. 6, pp. 679-698, 1986.

[9] STB library, https://github.com/nothings/stb, 2021.

[10] FreeImage, https://freeimage.sourceforge.io/, 2021

[11] C. A. Reiter, A local cellular model for snow crystal growth, Chaos, Solitons, and Fractals, Vol. 23, pp. 1111-1119, https://patarnott.com/pdf/SnowCrystalGrowth.pdf, 2005.

[12] J. Li, L. P. Schaposnik, Interface control and snow crystal growth, Physical Review E, Vol. 93, 023302, https://arxiv.org/pdf/1505.02042.pdf, 2016.

[13] Y. Huck, A Cellular Automaton Model for Snow Crystal Growth, https://itp.uni-frankfurt.de/~gros/StudentProjects/Projects_2020/projekt_yan_huck/, 2022

[14] N-body problem, Wikipedia, https://en.wikipedia.org/wiki/N-body_problem, 2022