SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1152

PARALELNA SIMULACIJA GIBANJA NEBESKIH TIJELA

Hrvoje Ban

Zagreb, lipanj 2010.

Sadržaj

Uvod

Problem paralelne simulacije gibanja nebeskih tijela zanimljiv je iz nekoliko razloga. Cilj rješavanja ovog problema, koje spaja područja astrofizike i računarske znanosti, jest na učinkovit način izračunati gibanja dinamičnog sustava tijela koja međusobno utječu jedna na druge demonstrirajući pritom kaotično ponašanje.

Računanje gibanja nebeskih tijela jedan je od problema čije se vrijeme rješavanja može osjetno smanjiti primjenom paralelnog računanja. Umjesto ograničavanja na samo jedno računalo i procesorsku jezgru, moguće je primjenom paralelizacije istovremeno koristiti više procesorskih jezgri i umreženih računala. Postupak paralelizacije uvodi nove probleme jer je originalni slijedni algoritam potrebno prilagoditi paralelnom izvođenju.

Ovaj rad sastoji se od osam poglavlja. U prvom poglavlju opisan je problem simulacije gibanja nebeskih tijela i osnovni načini rada simulacije i utjecanja na njezinu točnost. U drugom poglavlju dan je opis metoda numeričke integracije koje se koriste za rješavanje sustava diferencijalnih jednadžbi koje određuju gibanja tijela. U trećem poglavlju opisani su različiti načini simulacije gibanja, njihova točnost i utjecaj broja tijela na vrijeme računanja. U četvrtom poglavlju dan je opis načina paralelnog računanja korištenjem više umreženih računala ili procesorskih jezgri kao i načini međusobne komunikacije i raspodjele podataka između procesora. U petom poglavlju opisani su načini sinkronizacije rada i razmjene podataka između više dretvi. U šestom poglavlju dan je opis modela simulacije kao i načina implementacije izabranih algoritama za numeričku integraciju, simulaciju gibanja, paralelno računanje i sinkronizaciju. U sedmom poglavlju opisana je implementacija odabranih algoritama u programskom jeziku Java. Osmo poglavlje opisuje način ispitivanja učinkovitosti paralelnog računanja u odnosu na slijedni algoritam kao i rezultate ispitivanja za simulacije gibanja nekoliko različitih sustava nebeskih tijela.

1 Gibanje nebeskih tijela

Problem gibanja nebeskih tijela nastao je zbog potrebe za razumijevanjem gibanja zvijezda, planeta i drugih tijela koja se međusobno privlače gravitacijskim silama. Uz poznate početne uvjete sustava tijela (položaje i brzine), rješavanje problema zahtjeva računanje gibanja tijela za sve buduće ili prošle trenutke. Problem je prvi put formalno definirao Isaac Newton u knjizi *Principia*. Newton je tijela modelirao kao točkaste mase, a gravitacijsko privlačenje izrazio je sustavom diferencijalnih jednadžbi.

Promjena brzine tijela ovisi o njegovom položaju i masi kao i položajima i masama svih drugih tijela. Sila privlačenja između dva tijela proporcionalna je njihovim masama, a obrnuto proporcionalna kvadratu udaljenosti između njih. Sila koja djeluje na određeno tijelo jednaka je sumi pojedinačnih sila kojima na njega djeluju sva ostala tijela. Formalno, jednadžba gibanja za tijelo i u sustavu s ukupno N tijela jest:

$$\ddot{r_i} = -\mathrm{G} \sum_{j=1;\, j\neq i}^{N} \frac{m_j(r_i - r_j)}{\left|r_i - r_j\right|^3} \tag{1.1}$$

gdje je $\ddot{r_i}$ ubrzanje tijela, r_i i r_j su vektori položaja, m_j je masa tijela, a G gravitacijska konstanta ($\approx 6{,}67 \cdot 10^{-11}\ \mathrm{Nm^2kg^{-2}}$). Radi lakšeg računanja za gravitacijsku konstantu može se uzeti vrijednost od $1\ \mathrm{Nm^2kg^{-2}}$ ako se vrijednosti za masu, položaj i brzinu tijela skaliraju kako bi sustav ostao ispravan (Aarseth, 2003).

Diferencijalnu jednadžbu (1.1) drugog reda moguće je rastaviti na dvije jednadžbe prvog reda koje je dodatno po prostornim komponentama moguće rastaviti na tri nove jednadžbe. Za sustav od N tijela tako se dobija $6N$ diferencijalnih jednadžbi prvog reda. Ako se sustav ograniči na samo dvije dimenzije broj jednadžbi će pasti na četiri po tijelu.

Broj jednadžbi moguće je smanjiti uvođenjem linearno neovisnih algebarskih integrala. Za problem gibanja nebeskih tijela moguće je izvesti deset takvih integrala (sedam ako je sustav ograničen na dvije dimenzije). To su integrali za položaj centra mase, linearni moment, kutni moment i energiju sustava. To su ujedno i jedini linearno neovisni algebarski integrali za problem gibanja nebeskih tijela (Beutler, 2005).

Osim za trivijalni slučaj jednog tijela, egzaktna rješenja postoje samo za sustave dva tijela i neke posebne slučajeve sustava tri tijela (Aarseth, 2003). Primjeri takvih sustava su binarne zvijezde ili binarni asteroidi kao sustavi dva tijela te gibanje planeta i njegovog satelita oko sunca kao sustav tri tijela. Sustavi s više od tri tijela demonstriraju kaotično gibanje te ih nije moguće riješiti analitički, već jedino postupcima računalne simulacije i numeričke integracije.

Kako se skoro svi zanimljivi sustavi nebeskih tijela sastoje od više milijuna zvijezda koje je potrebno promatrati kroz duži vremenski period, kao jedini način rješavanja problema gibanja nameće se korištenje računalne simulacije. S obzirom na to da su računala digitalni strojevi, prostor i vrijeme potrebno je aproksimirati vrijednostima konačne preciznosti. Iako takve aproksimacije unose određene pogreške u simulaciju, veličinu pogrešaka često je moguće smanjiti na prihvatljive vrijednosti.

Prva vrsta pogreške nastale simulacijom je ona zbog podijele vremena na diskretne trenutke. Duljina vremenskog intervala između dva susjedna trenutka zove se korak vremena (engl. *time step*). Zadatak simulacije jest na temelju stanja sustava u jednom trenutku izračunati stanje sustava u sljedećem trenutku. Simulacija će biti točnija što je korak vremena manji, ali će vrijeme izvođenja simulacije biti proporcionalno veće.

Računanje sljedećeg stanja sustava odvija se u dva dijela. Prvo je potrebno izračunati sile koje djeluju na pojedina tijela, a zatim je te sile potrebno integrirati kroz vrijeme određeno veličinom koraka vremena kako bi se izračunali novi položaji i brzine tijela. Tijekom računanja sila moguće je iskorištavanjem određenih svojstava sustava nebeskih tijela postići drastično ubrzanje uz relativno malen gubitak točnosti. Točnost simulacije također ovisi i o izboru metode integracije koje variraju od jednostavnih i vrlo efikasnih do kompleksnih koje nude veliku točnost uz sporije izvođenje.

Ako se tijekom simulacije dva ili više tijela nađu vrlo blizu, sile kojima bi oni djelovali jedni na druge postale bi vrlo velike jer rastu obrnuto proporcionalno kvadratu udaljenosti. Ispravno računanje s tako velikim silama zahtijevalo bi proporcionalno mali korak vremena što bi se negativno odrazilo na trajanje simulacije. Ovaj problem može se zaobići uvođenjem faktora gušenja (engl. *softening factor*) koji će ograničiti iznos sila kojima bliska tijela djeluju jedna na druge i time omogućiti brže simuliranje bliskih susreta, ali uz manju točnost simulacije (Aarseth, 2003).

2 Numerička integracija

Diferencijalne jednadžbe su vrste jednadžbi čije rješavanje zahtjeva pronalaženje nepoznate funkcije ako je poznata njezina derivacija. Obična diferencijalna jednadžba prvog stupnja je svaka jednadžba oblika $y'(t) = f(t, y(t))$ što znači da prva derivacija može ovisiti o vremenu kao i o vrijednosti nepoznate funkcije. Mnoge diferencijalne jednadžbe, poput onih koje se javljaju u simulaciji gibanja nebeskih tijela, nije moguće riješiti analitički. S druge strane, ako je vrijednost derivacije poznata u određenim vremenskim trenucima (koji su razmaknuti za duljinu koraka integracije), vrijednost nepoznate funkcije moguće je izračunati za te trenutke postupkom numeričke integracije.

Metode numeričke integracije unose određenu pogrešku u izračun. Veličina pogreške usko je povezana uz veličinu koraka integracije. Lokalna pogreška računa se za jedan korak integracije, dok se ukupna pogreška računa za veći niz koraka. Ako je veličinu ukupne pogreške moguće prikazati kao potenciju veličine koraka integracije, onda je red numeričke metode integracije jednak eksponentu te potencije (Drmač et al., 2003).

2.1 Metode integracije prvog reda

Za veličinu koraka integracije h, metode integracije prvog reda uzrokuju lokalnu pogrešku reda h^2, te ukupnu pogrešku reda h. Primjer metode prvog reda je Eulerova metoda. Uz poznatu vrijednost tražene funkcije i njezine derivacije u trenutku n, aproksimacija vrijednosti funkcije za trenutak $n + 1$ jest:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n) \tag{2.1}$$

Prednost Eulerove metode je njezina jednostavnost i potreba za računanjem derivacije za samo jedan trenutak što je dobro svojstvo ako je funkcija derivacije komplicirana za računanje. Najveća mana ove metode je relativno veliki iznos pogreške kao i numerička nestabilnost kod određenih vrsta jednadžbi.

2.2 Metode integracije drugog reda

Za veličinu koraka integracije h, metode integracije drugog reda uzrokuju lokalnu grešku reda h^3, te ukupnu pogrešku reda h^2. Primjeri metoda drugog reda su metoda srednje točke i Verletova metoda za integraciju brzine. Za razliku od metoda prvog reda, ove metode zahtijevaju najmanje dva izračuna po koraku integracije.

Metoda srednje točke (engl. *midpoint method*) započinje uzimajući vrijednost derivacije u trenutku n, te na temelju te derivacije računa aproksimaciju funkcije za trenutak $n + 0{,}5$. Koristeći vrijednost u srednjem trenutku (po čemu je metoda dobila ime), računa se vrijednost derivacije u trenutku $n + 0{,}5$. Vrijednost funkcije za trenutak $n + 1$ računa se izrazom:

$$y_{n+1} = y_n + h \cdot f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n)\right) \tag{2.2}$$

što znači da se vrijednost aproksimira koristeći vrijednost funkcije u trenutku n i vrijednost derivacije u trenutku $n + 0{,}5$.

Verletova metoda za integraciju brzine (engl. *Velocity Verlet*) pogodna je jedino za Newtonove jednadžbe gibanja. Verletova metoda računa funkciju položaja odvojeno od funkcije brzine s tim da su trenutci u kojima se računa brzina pomaknuti za pola koraka u odnosu na trenutke u kojima se računa položaj. Izrazi za položaj x i brzinu v s obzirom na ubrzanje a su:

$$x_{n+1} = x_n + h \cdot v_{n+0{,}5} \tag{2.3}$$

$$v_{n+0{,}5} = v_{n-0{,}5} + h \cdot a_n \tag{2.4}$$

Izraze (2.3) i (2.4) moguće je zapisati na način da se i brzina računa u cjelobrojnim koracima čime je pojednostavljena implementacije ove metode jer nije potrebno voditi računa o međukoracima. Verletova metoda najpogodnija je za jednadžbe u kojima ubrzanje ne ovisi o brzini već jedino o položaju tijela (kao što su jednadžbe gibanja nebeskih tijela) jer se inače jednadžba brzine mora rješavati implicitnim načinom koji je sporiji od izravnog računanja.

2.3 Metode integracije višeg reda

U metode integracije višeg reda spadaju vrste Runge-Kuttinih i linearnih višekoračnih metoda. Obje vrste omogućavaju konstrukciju metoda integracije proizvoljnog reda.

Runge-Kuttine vrste metoda rade na sličan način kao metoda srednje točke. Prvo se na temelju vrijednosti derivacije u trenutku n Eulerovom metodom izračuna vrijednost funkcije u trenutku između n i $n+1$. Korištenjem izračunate vrijednosti računa se derivacija za taj trenutak te se Eulerova metoda ponavlja, ali ovaj put koristeći novu vrijednost derivacije. Postupak se ponavlja proizvoljni broj puta, nakon čega se računa težinska sredina svih izračunatih vrijednosti koja predstavlja vrijednost tražene funkcije u trenutku $n+1$. Formalno, Runge-Kuttina metoda je svaka metoda oblika:

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i \cdot f(t_n + c_i \cdot h, y_n + a_i \cdot k_{i-1}) \tag{2.5}$$

gdje se s broj međukoraka, b_i težinski faktor, c_i veličina međukoraka, a a_i težinski faktor kojim se skalira vrijednost prethodnog međukoraka (Drmač et al., 2003).

Linearne višekoračne metode računaju vrijednost funkcije za neki trenutak kao linearnu kombinaciju poznatih ili izračunatih vrijednosti funkcije i njezine derivacije u određenom broju prijašnjih trenutaka. Red višekoračnih metoda ovisi o broju prijašnjih trenutaka koji se koriste za izračun. Nedostatak ovih metoda je što zahtijevaju korištenje neke druge metode (koja nije višekoračna) za računanje prvih nekoliko vrijednosti funkcije kako bi višekoračna metoda imala dovoljno početnih podataka za računanje. Linearne višekoračne metode imaju sljedeći oblik:

$$\begin{aligned} &y_{n+s} + a_{s-1}y_{n+s-1} + a_{s-2}y_{n+s-2} + \cdots + a_0 y_n = \\ &h[b_s f(t_{n+s}, y_{n+s}) + b_{s-1} f(t_{n+s-1}, y_{n+s-1}) + \cdots + b_0 f(t_n, y_n)] \end{aligned} \tag{2.6}$$

Linearne višekoračne metode konstruiraju se tako da se prvo odabere željeni red metode. Da bi metoda bila konzistentna koeficijent a_{s-1} mora biti -1, dok svi ostali koeficijenti a moraju biti jednaki nuli. Koeficijenti b biraju se tako da metoda egzaktno interpolira polinomom stupnja jednakog redu metode.

3 Metode simulacije gibanja

Sve metode simulacije gibanja nebeskih tijela imaju isti osnovni algoritam:

- Prvo se za svako tijelo računa rezultanta sila kojima druga tijela djeluju na njega. Ovo je najzahtjevniji korak simulacije čija kompleksnost u *Big O* notaciji iznosi $\mathcal{O}(Nx)$ gdje je N broj tijela, a x vrijeme potrebno za računanje rezultante sila koje djeluju na jedno tijelo.
- Ako se koristi promjenjivi korak vremena njegova veličina određuje se na temelju veličine najveće rezultante sila (umanjene faktorom gušenja).
- Koristeći odabranu metodu numeričke integracije, položaji i brzine nebeskih tijela računaju se za sljedeći korak simulacije. Za veličinu koraka integracije koristi se veličina prethodno odabranog koraka vremena.

Ono po čemu se metode simulacije gibanja razlikuju jest način računanja rezultante sila koje djeluju na jedno tijelo. Sve metode, osim izravne, primjenjuju aproksimacije kako bi vrijeme računanja spustili na prihvatljivu razinu. Iako te aproksimacije smanjuju točnost simulacije, za neke metode moguće je točno izračunati veličinu pogreške i smanjiti je na razinu manju ili jednaku veličini sklopovske pogreške. Time točnost praktički ostaje ista uz drastično brže izvođenje simulacije (Salmon et al., 1994).

3.1 Izravna metoda

Izravna metoda simulacije najjednostavnija je od svih metoda. Ne koriste se aproksimacije, već se za svako tijelo izravno računaju sile kojima sva druga tijela djeluju na njega. Pojedinačne sile sumiraju se kako bi se izračunala rezultanta sila. Metoda ima veliku kompleksnost jer je za svako od N tijela potrebno izračunati $N - 1$ sila kojim djeluju druga tijela što daje $N(N - 1)$ operacija po koraku. Zapisano pomoću *Big O* notacije, kompleksnost metode je $\mathcal{O}(N^2)$ tj. vrijeme simulacije ovisi o kvadratu broja tijela. Zbog toga je ova metoda prihvatljiva samo za sustave s relativno malim brojem tijela.

3.2 Metoda potencijala mreže

Metoda potencijala mreže „prekriva“ prostor mrežom jednako udaljenih točaka. U svakom koraku simulacije računa se:

- Naboj svake točke mreže koji ovisi o tijelima koja se nalaze u blizini te točke. Ovisno o načinu na koje tijelo svojom masom pridonosi naboju obližnjih točaka mreže postoji nekoliko podvrsta ove metode: tijelo može utjecati samo na najbližu točku ili može utjecati na više okolnih točaka. Tijelo može utjecati jednoliko ili prema određenoj funkciji raspodjele naboja koji uzima u obzir položaj tijela u odnosu na obližnje točke.
- Na temelju izračunatog naboja točaka računa se potencijal mreže. Potencijal nije potrebno računati za cijelu mrežu već samo za položaje na kojima se nalaze tijela.
- Pomoću dobivenog potencijala računaju se rezultante sila koje djeluju na tijela.

Najsporiji dio ove metode je računanje potencijala mreže koji se često rješava korištenjem brze Fourierove transformacije (Bertschinger et al., 1991). Podjelom prostora na mrežu točaka kompleksnost jednog koraka simulacije reda je $\mathcal{O}(N + G\ log\ G)$, gdje je G broj točaka mreže. Nedostatak ove metode je što točnost simulacije uvelike ovisi o raspodijeli nebeskih tijela i gustoći točaka mreže. Dodatno, metoda prigušuje jake sile koje nastaju između dva ili više tijela koja se nalaze vrlo blizu jedna drugima. Takvo gušenje može biti prihvatljivo pri simuliranju sustava s velikim brojem tijela jer će sam sustav onda biti stabilniji, ali je neprihvatljivo ako se proučavaju interakcije između bliskih tijela.

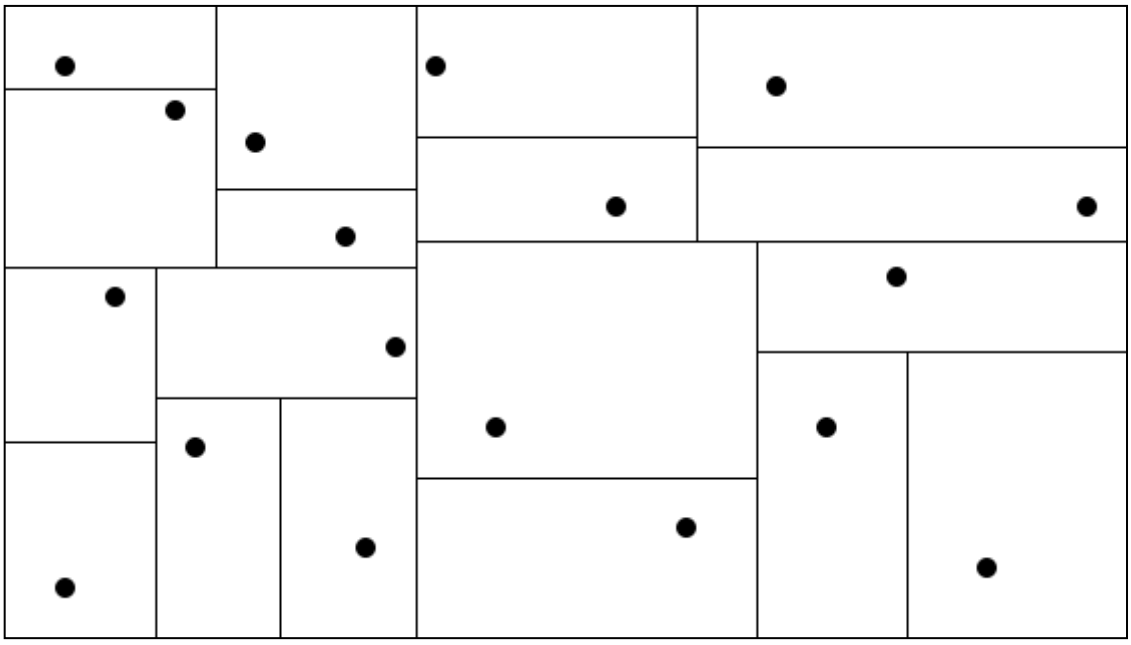
Točnost simulacije sustava gusto raspoređenih tijela može se povećati ako se poveća gustoća točaka mreže. Točnost bi bila najveća kada bi točke bile raspoređene jednako gusto kao i tijela, ali bi se time izgubila svaka dobit u brzini izvođenja koju donosi ova metoda. Rješenje ovog problema jest dinamično dodavanje i premještanje točaka mreže. U dijelovima prostora gdje se nalazi velika koncentracija nebeskih tijela može se povećati gustoća točaka mreže, dok se u dijelovima gdje se nalazi relativno mali broj tijela ili ih opće nema broj točaka može proporcionalno smanjiti. Metoda se može dodatno poboljšati ako se točke mreže učine pomičnima tako da prate gibanja tijela. U tom slučaju dobro raspoređene točke će „putovati“ zajedno s tijelima koja na njih utječu pa broj točaka mreže neće biti potrebno često mijenjati. Kako se tijela gibaju u relativno malim pomacima, tako će i promjene strukture mreže biti minimalne.

3.3 Algoritam Barnes-Hut

Algoritam Barnes-Hut zasniva se na dekompoziciji prostora pomoću strukture stabla. Način dekompozicije ovisi o vrsti stabla koja se koristi – za balansirana stabla upotrebljava se metoda *ortogonalne rekurzivne bisekcije*, a za nebalansirana algoritam gradnje *quadtree* stabla za dvodimenzionalni prostor ili *octree* stabla za trodimenzionalni prostor.

Ortogonalna rekurzivna bisekcija zasniva se na dijeljenju prostora na dvije disjunktivne ćelije tako da svaka ćelija sadrži podjednak broj tijela (Dubinski, 1996). Prostor se dijeli po jednoj od tri glavne osi, a obično se bira ona os po kojoj se prostor ćelija najmanje rasprostire. Proces dijeljenja nastavlja se na obje dobivene ćelije sve dok se prostor ne podijeli na ćelije koji sadrže samo jedno tijelo. Binarno stablo gradi se tako da korijenski čvor predstavlja cijeli prostor, a njegova djeca predstavljaju ćelije dobivene prvim dijeljenjem prostora. Djeca tih čvorova predstavljaju pak rezultat dijeljenja novih ćelija, sve do čvorova listova koji predstavljaju ćelije koje sadrže samo jedno tijelo.

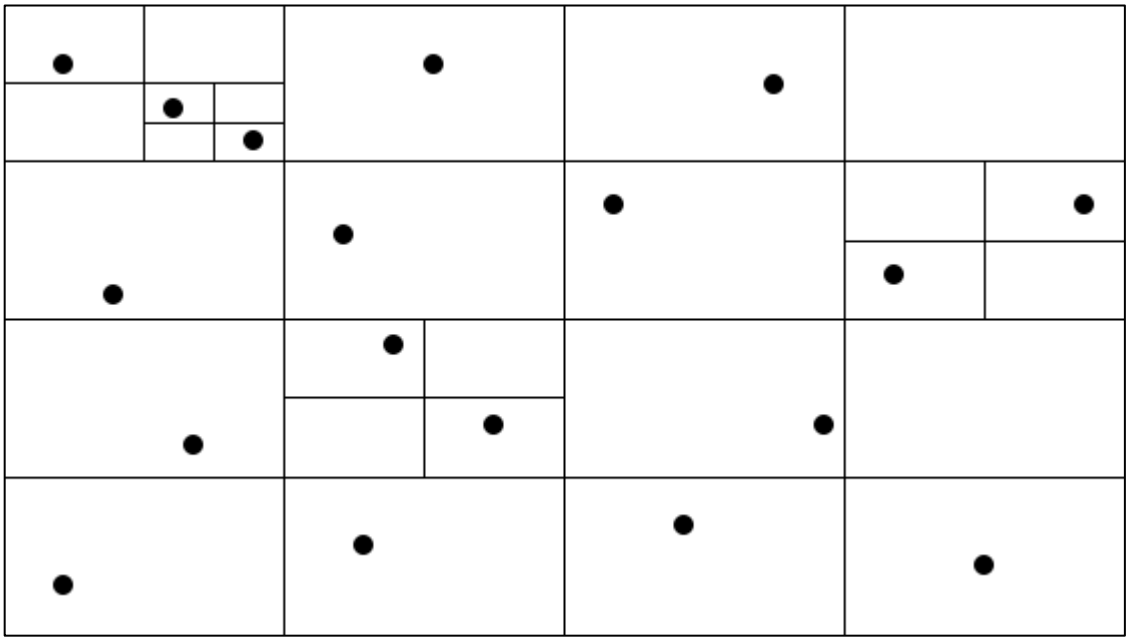
Kako se prostor uvijek dijeli tako da polovice sadrže podjednak broj tijela, dobiveno binarno stablo uvijek će biti balansirano što je velika prednost ovog načina dekompozicije prostora. Nedostatak je što je prostor često vrlo nepravilno podijeljen pa je teško predvidjeti konačni položaj pojedinih ćelija. Slika 3.1 prikazuje primjer dijeljenja prostora korištenjem ortogonalne rekurzivne bisekcije.



Slika 3.1: Dekompozicija prostora ortogonalnom rekurzivnom bisekcijom

Quadtree stablo je vrsta stabla čiji unutarnji čvorovi imaju točno četvero čvorova djece, dok unutarnji čvorovi *octree* stabla imaju točno osmero čvorova djece. Algoritam za gradnju *quadtree* stabla počinje tako da cijeli prostor predstavi kao jednu pravokutnu ćeliju. Ta ćelija se onda dijeli na četiri kvadranta jednakih dimenzija. Svaki kvadrant predstavlja jednu novu manju ćeliju prostora. Postupak dijeljenja rekurzivno se nastavlja na novim ćelijama sve dok se prostor ne podijeli na ćelije koje predstavljaju dio koji sadrži samo jedno tijelo ili je prazan (Warren, 1992). Struktura ćelija izravno se preslikava u strukturu *quadtree* stabla – svaka ćelija predstavljana je jednim čvorom stabla, a četiri ćelije nastale njezinim dijeljenjem predstavljene su kao djeca toga čvora. Algoritam gradnje *octree* stabla radi na sličan način, samo što je ćelija oblika trodimenzionalnog kvadra koji se dijeli na osam manjih dijelova jednakih dimenzija. Slika 3.2 prikazuje dekompoziciju pomoću *quadtree* stabla istog prostora koji je korišten za prethodni primjer.

Struktura stabla određena je raspodjelom tijela u prostoru jer se ćelije dijele na manje dijelove samo ako sadrže više od jednog tijela. Zbog toga će algoritam velike površine praznog prostora predstaviti relativno malim brojem ćelija i dio stabla za taj prostor će biti male visine. S druge strane područja u kojima se nalazi velika koncentracija tijela biti će potrebno podijeliti na ćelije vrlo malih dimenzija zbog čega će se broj potrebnih ćelija proporcionalno povećati, a samim time i visina dijela stabla za taj dio prostora. Iako dobiveno stablo nije balansirano, položaji i dimenzije ćelija koje dijele prostor slijede strogo određena pravila, što olakšava obradu i analizu ove vrste stabla.



Slika 3.2: Dekompozicija prostora pomoću strukture *quadtree* stabla

Tijekom računanja sila koje djeluju na pojedina tijela algoritam Barnes-Hut primjenjuje sljedeću korisnu aproksimaciju: grupa tijela koja se nalazi na relativno velikoj udaljenosti djeluje silom sličnoj onoj kojom bi djelovalo jedno tijelo mase jednakoj ukupnoj masi cijele grupe i smješteno u centru mase te grupe. Ovo omogućava da se udaljena skupina tijela aproksimira jednim pseudo-tijelom koje zahtjeva dosta manje računanja (Warren, 1992). U svakom koraku simulacije algoritam Barnes-Hut:

- Gradi stablo dekompozicijom prostora pomoću metode ortogonalne rekurzivne bisekcije ili metode gradnje *quadtree* ili *octree* stabla.
- Za svaku ćeliju prostora računa se ukupna masa i položaj centra mase svih tijela sadržanih u toj ćeliji. Računanje počinje od listova stabla koji sadrže samo jedno tijelo. Ukupna masa i položaj centra mase čvora lista jednaki su masi i položaju tijela kojega taj list sadrži. Ukupna masa i položaj centra mase unutarnjih čvorova računa se pomoću podataka čvorova djece.
- Pri računanju rezultante sila koje djeluju na jedno tijelo algoritam počinje od korijenskog čvora. Računa se omjer između duljine stranice ćelije koju čvor predstavlja i udaljenosti tijela od centra mase te ćelije. Ako je omjer manji od prethodno zadane granice, *kuta otvaranja*, sila koja djeluje na tijelo računa se na temelju ukupne mase i položaja centra mase ćelije. Ako je omjer veći od kuta otvaranja, algoritam se rekurzivno spušta na čvorove djecu sve dok ne dođe ili do ćelije koja zadovoljava uvjet aproksimacije ili čvora lista.

Uz dobro izabranu veličinu kuta otvaranja moguće je izbjeći veliku količinu računanja jer će se velike skupine bliskih tijela tretirati kao jedno veliko tijelo. S obzirom na to da je broj operacija tijekom jednog koraka određen brojem obrađenih čvorova stabla, a visina stabla ovisi o logaritmu broja tijela, kompleksnost jednog koraka algoritma Barnes-Hut iznosi $\mathcal{O}(N\ logN)$.

Greška aproksimacije ovisi o veličini kuta otvaranja pa je veličinu te greške moguće izračunati i smanjiti na prihvatljivu razinu. Točnost simulacije može se dodatno poboljšati izborom pametnijeg kriterija za otvaranje ćelija (Salmon, 1994) kao i korištenjem više podataka o raspodijeli tijela, osim ukupne mase i položaja centra mase. Dodatno ubrzanje može se postići ako se prostor ne dijeli do ćelija koje sadrže samo jedno tijelo, već manjeg broja više tijela. Interakcije tijela unutar jedne ćelije moraju se računati izravnom metodom, ali će algoritam biti učinkovitiji jer će visina stabla biti manja.

3.4 Brza višepolna metoda

Brza višepolna metoda (engl. *fast multipole method*) radi na sličan način kao algoritam Barnes-Hut, ali umjesto sa silama radi s poljem potencijala. Metoda koristi dva različita načina za računanje iznosa potencijala: višepolne ekspanzije za udaljene točke i lokalne ekspanzije za bliske točke.

Algoritam počinje dekompozicijom prostora korištenjem strukture stabla. Počevši od listova stabla algoritam računa ekspanzije polja potencija koje generiraju tijela. Tijela utječu na potencijal čvorova koji predstavljaju ćeliju unutar koje se ta tijela nalaze, a čvorovi utječu na iznos potencijala čvorova roditelja. Nakon što se izračuna potencijal korijenskog čvora, algoritam obavlja prolaz od korijena prema čvorovima listovima pri čemu ispravlja prije izračunati potencijal koristeći drugu vrstu ekspanzija. Kada ispravno izračuna potencijal čvorova listova, koristi te vrijednosti kako bi izračunao sile koje djeluju na tijela unutar ćelija koje predstavljaju ti čvorovi (Greengard et al., 1987).

Ovaj način računanja sila dovodi do kompleksnosti $\mathcal{O}(N)$ za jedan korak simulacije što je bolje od algoritma Barnes-Hut, ali zato dosta kompliciranije.

3.5 Metoda samo-konzistentnog polja

Metoda samo-konzistentnog polja (engl. *self-consistent field*) može se koristiti za simulaciju gibanja nebeskih tijela bez sudara. Metoda ne računa međusobna djelovanja tijela, već njihove doprinose gravitacijskom polju sustava. Ova metoda ne uzima u obzir relativne položaje tijela, već samo raspodjelu njihove mase u prostoru.

U svakom koraku simulacije računa se iznos gravitacijskog polja sustava, te djelovanje tog polja na svako od tijela. Prednost ove metode je što se svako tijelo može promatrati neovisno o drugim tijelima. Umjesto jednog problema gibanja više tijela, rješava se više problema gibanja jednog tijela (Hernquist et al., 1992).

Metoda postiže vrlo dobru kompleksnost od $\mathcal{O}(N)$ za jedan korak simulacije, ali se ne može koristiti za simuliranje sudara jer zanemaruje međusobne interakcije bliskih tijela.

4 Metode paralelnog računanja

Paralelno računanje je način ubrzanja izvođenja određenih algoritama. Osnovna ideja je da se problem podijeli u više manjih neovisnih problema koji se onda mogu rješavati istovremeno tj. paralelno. Paralelno računanje može se ostvariti korištenjem više procesora koji zasebno rješavaju pojedine manje probleme. Većina problema sastoji se od podataka koje treba obraditi i skupa operacija koje se obavljaju nad tim podacima. Veće skupine povezanih operacija čine zadatke. Prema tome, dva načina podjele problema na manje su:

- *Podatkovni paralelizam* kada se skup podataka za obradu dijeli na procesore, a svi procesori izvršavaju iste operacije nad tim podacima. Poželjno je podatke podijeliti tako svaki procesor bude podjednako opterećen i da računanje bude što je moguće manje ovisno o računanju ostalih procesora.
- *Paralelizam zadataka* raspoređuje zadatke po procesorima dok podaci nad kojima se oni izvršavaju ostaju zajednički. Paralelizam zadataka može podrazumijevati i podatkovni paralelizam ako su podaci koje obrađuju pojedini zadaci neovisni. Ovaj način je koristan kada svi zadaci zahtijevaju približno jednako vremena za rješavanje, inače može doći do nepoželjne situacije kada je zbog jednog zahtjevnog zadatka samo jedan procesor zauzet računanjem dok su svi ostali procesori slobodni i čekaju.

Nakon što se odluči o načinu podjele problema na manje, potrebno je odabrati način paralelnog računanja. Ako se želi koristiti više umreženih računala koja mogu međusobno komunicirati dobar izbor je MPI, dok se za paralelno računanje na jednom računalu pomoću više dretvi može koristiti OpenMP ili programsko sučelje za višedretvenost.

Paralelnim računanjem najčešće se može postići tek linearno ubrzanje, proporcionalno broju korištenih procesora. Kod nekih algoritama koji se koriste za pretraživanje prostora stanja ili intenzivno koriste priručnu memoriju moguće je postići ubrzanje veće od linearnog, tzv. *superlinearno* ubrzanje. Ipak, najveće se ubrzanje može postići korištenjem algoritama manje složenosti, stoga paralelno računanje ima smisla primijeniti samo nakon što je izabran dobar algoritam za rješavanje problema.

4.1 Message Passing Interface

Message Passing Interface (MPI) je specifikacija programskog sučelja za međusobnu komunikaciju između umreženih računala. MPI je neovisan o programskom jeziku te omogućuje izravnu razmjenu poruka između dva računala, slanje poruka s jednog računala prema više njih i slanje s više računala prema jednom (MPI Forum, 2009).

MPI je osmišljen za podršku paralelnom računanju zasnovanom na sustavu raspodijeljene memorije gdje svaki procesor ima svoj dio memorije te ne može izravno pristupiti memoriji drugih procesora. Jedini način interakcije između procesora je preko slanja i primanja poruka.

Radi lakše organizacije, MPI specificira objekt komunikator koji služi za grupiranje skupine procesora. Svaki procesor ima identifikator koji je jedinstven unutar njegove grupe. Dijeljenje procesora u neovisne grupe korisno je ako procesori rješavaju više različitih zadataka jer je za svaki zadatak moguće stvoriti posebnu grupu. Dijeljenje po grupama obavlja se pomoću algoritma za bojanje grafa koji nastoji topologiju grupiranih procesora učiniti što učinkovitijom. Svaki procesor ima jedinstveni identifikator ili rang unutar svake grupe kojoj pripada. Identifikatori su cijeli brojevi koji se kreću od 0 do $N - 1$ gdje je N broj procesora u grupi. Identifikator 0 često se posebno tretira, npr. za određivanje glavnog procesora koji će koordinirati rad svih ostalih.

Način izravnog slanja omogućuje slanje poruke točno određenomu procesoru. Primatelj poruke određuje se pomoću jedinstvenog identifikatora. Ovo omogućuje laku implementaciju algoritama u kojima procesori redovito moraju razmjenjivati podatke, a svaki procesor zna od kojeg točno drugog procesora mora primiti podatke tj. komu ih mora poslati. Izravne poruke korisne su i za *master-slave* arhitekturu kada se jedan procesor odredi kao glavni koji raspoređuje zadatke i ostali procesori komuniciraju samo s njim.

MPI podržava dva načina slanja poruka: blokirajući i neblokirajući. Pri slanju blokirajuće poruke procesor mora čekati dok MPI obrađuje podatke koji se šalju. Nakon slanja neblokirajuće poruke procesor može odmah nastaviti s radom, ali zato mora sam provjeravati stanje poruke kako bi mogao ustanoviti kada je MPI završio s obradom podataka i kada je druga strana tu poruku primila. Korištenje neblokirajućih poruka u pravilu smanjuje vrijeme potrošeno na komunikaciju, ali komplicira algoritam.

Postoji nekoliko načina višestrukog slanja poruka: jedan procesor može poslati istu poruku svim drugim procesorima u grupi, svi procesori mogu istodobno poslati dio iste poruke svim procesorima ili svi procesori mogu poslati poruke jednom procesoru. Kod zadnjeg načina slanja moguće je nad svim porukama primijeniti operaciju redukcije koja će više poruka akumulirati u jednu. Često korištena redukcija jest zbrajanje kada se primljeni podaci sumiraju i procesor koji prima poruke koristi tu sumu kao rezultat.

MPI ima ugrađenu podršku za neke osnovne vrste podataka kao što su okteti, znakovi, cijeli i realni brojevi te polja. MPI također omogućava definiranje struktura koje će predstavljati novu vrsta podataka. Strukture mogu sadržavati osnovne vrste podataka i druge strukture. Pri definiranju nove strukture potrebno je navesti sve njezine članove, veličine tih članova i njihov razmještaj unutar strukture. Iako su praktične, razmjena struktura pomoću poruka je sporija zbog dodatnih operacija potrebnih za pakiranje struktura u poruke prije slanja i raspakiravanja iz poruka nakon primanja.

Operacije primanja poruka mogu se koristiti kao mehanizmi sinkronizacije procesora. Procesor koji očekuje poruku mora zaustaviti svoj rad i čekati na pristizanje poruke. Procesor može čekati poruku od točno određenog procesora ili može prihvatiti prvu poruku koja stigne. Kako sve poruke ne nose iste informacije, svakoj poruci moguće je dodijeliti oznaku (engl. *tag*) – numerički identifikator koji jedinstveno određuje vrstu poruke i podatke koje nosi. Procesor koji čeka poruku tako može odrediti da ne očekuje bilo koju vrstu poruke, već poruku s točno određenom oznakom.

Skup računala koja koriste MPI za paralelno računanje često su definirana u trenutku pokretanja računanja te nije moguće dinamički dodavati ili uklanjati procesore. MPI verzije 2 donosi tu mogućnost čime se može koristiti za pravo raspodijeljeno računanje. Nova verzija dodatno donosi podršku za paralelni rad s ulazno/izlaznim jedinicama. Procesori tako mogu paralelno čitati i zapisivati u dijeljene datoteke koje se koriste kada se obrađuju podaci koji zbog svoje veličine ne mogu stati u memoriju računala već moraju biti pohranjeni na vanjske diskove.

Iako se MPI najčešće koristi na grozdovima računala, moguće ga je koristiti i za raspodijeljeno računanje kao i za rad na samo jednom računalu kada se paralelno koristi više procesorskih jezgri. MPI može ustanoviti na kakvoj se arhitekturi koristi i prema tome prilagoditi način razmjene poruka s ciljem smanjenja vremena komunikacije.

4.2 Open Multi-Processing

Open Multi-Processing (OpenMP) je programsko sučelje koje se može koristiti za paralelno računanje na arhitekturi dijeljene memorije. OpenMP omogućava paralelno izvršavanje programa na više procesorskih jezgri pomoću dretvi. Velika prednost ove metode je u tome što je moguće prvo implementirati slijedni algoritam, koji je često jednostavniji i lakši za testiranje od paralelnog, a onda naknadno dodati OpenMP direktive za paralelno računanje bez potrebe za velikim izmjenama (OpenMP ARB, 2009).

OpenMP implementacija postoji za programske jezike C, C++ i Fortan, a radi na principu dodavanja direktiva koje mijenjaju programski kod, te skupa pomoćnih biblioteka koje pružaju podršku paralelnom računanju. Program se izvodi u samo jednoj dretvi dok ne dođe do dijela koji je označen za paralelno izvođenje. Tada OpenMP stvara dodatne dretve, čiji broj može ovisiti o broju procesorskih jezgri, i dodjeljuje im podatke i zadatke koje trebaju obraditi. Kada sve dretve završe sa svojim dijelom posla OpenMP ih sinkronizira te dodatne dretve zaustavlja, a glavnoj dretvi dopušta daljnje izvođenje.

Dijelove programskog koda koji se trebaju paralelno izvršavati potrebno je označiti posebnim OpenMP direktivama. Dijelovi koji se najčešće označavaju su petlje i blokovi programskih naredbi. U programskim jezicima C i C++ direktive se dodaju kao *pragma* pretprocesorske naredbe koje prevoditelji koji podržavaju OpenMP razumiju i programski kod mijenjaju u skladu s tim direktivama. Prevoditelji koji ne podržavaju OpenMP jednostavno će ignorirati direktive i programski kod će prevesti u slijedni program.

Osnova direktiva je *parellel* koju je moguće postaviti iznad bilo kojeg bloka, petlje ili naredbe. Direktiva će uzrokovati paralelno izvršavanje tih naredbi, s tim da će svaka dretva obavljati identičan posao. Kako paralelno izvođenje ima smisla samo ako dretve obavljaju različite poslove, potrebno je dodati naredbe i direktive koje će podijeliti podatke ili zadatke. Direktiva *parellel* može se koristiti unutar druge direktive za paralelno računanje ako se koristi s opcijom *single* ili *master*. Te opcije će ograničiti izvođenje niza naredbi na samo jednu dretvu što je korisno ako se dretve periodički moraju sinkronizirati tako da samo jedna dretva obavi usklađivanje. Korištenjem opcije *single* sve dretve će čekati da dretva koja radi završi prije nego nastave s daljnjim izvođenjem, dok će s opcijom *master* odmah nastaviti s radom.

Ako se direktiva *parellel* primjeni na *for* petlju kojoj su točno zadane cjelobrojne granice intervala, OpenMP će automatski izvršavanje petlje podijeliti po dretvama tako da svaka dretva obradi jedan dio intervala. Ako se unutar petlje akumulira rezultat korištenjem jedne ili više varijabli potrebno je dodati direktivu za redukciju. U slučaju da se rezultat zbraja treba dodati redukciju sume, pa će svaka dretva onda sumirati međurezultate u svoju varijablu, a po kraju izvođenja petlje svi će se međurezultati sumirati (reducirati) u jednu varijablu koja se dalje može koristiti kao rezultat.

OpenMP ima koncept privatnih i dijeljenih varijabli. Ako je varijabla označena kao privatna tada sve dretve sadrže vlastiti kopiju te varijable te ne postoji način da čitaju ili mijenjaju privatne varijable drugih dretvi. S druge strane, za dijeljene varijable postoji samo jedna instanca kojoj sve dretve mogu pristupati. Dijeljene varijable su korisne ako dretve trebaju čitati zajedničke podatke ili ako je potrebno izvesti redukciju koju OpenMP ne podržava jer je reduciranje moguće izvesti promjenom vrijednosti dijeljene varijable.

U slučaju pisanja u zajednički dio memorije potrebno je dretve sinkronizirati inače se može dogoditi da jedna dretva prepiše podatke koje druga dretva upravo čita ili mijenja. OpenMP podržava sinkronizaciju korištenjem direktive *critical*. Naredbe označene tom direktivom može istovremeno izvršavati samo jedna dretva, dok sve druge dretve moraju čekati da ona završi izvođenje tih naredbi. Sinkronizaciju je moguće izvesti i korištenjem direktive *barrier* koja će uzrokovati čekanje dretvi koje dođu do te direktive dok sve dretve ne dođu do nje. Za direktive koje implicitno uzrokuju takvo čekanje (kao što je *parallel* s opcijom *single*) moguće ga je isključiti korištenjem direktive *nowait*.

Prednost OpenMP-a je u njegovoj jednostavnosti. Dok je pri korištenju MPI-a potrebno algoritam oblikovati pazeći na slanje i primanje poruka, kod OpenMP-a dovoljno je dodati par direktiva u slijedni algoritam kako bi se omogućilo njegovo paralelno izvršavanje. Sam program ostaje neovisan o načinu izvođenja i arhitekturi na kojoj se izvodi jer OpenMP automatski otkriva broj procesora i procesorskih jezgri, te sam raspoređuje podatke po dretvama. Mana OpenMP-a je što zahtjeva prevoditelj koji ga podržava, te ne pruža načine za veću kontrolu izvođenja dretvi. Njegova jednostavnost može biti nedostatak ako je dretve potrebno sinkronizirati na način koji nije predviđen OpenMP-om, ili ako je potrebno implementirati vlastiti način raspodjele podataka. S obzirom na to da pretpostavlja arhitekturu dijeljene memorije, nije ga moguće koristiti za paralelno računanje na više računala.

4.3 Višedretvenost

Ako je potrebna potpuna kontrola nad izvršavanjem dretvi, raspodjelom podataka te međusobnom komunikacijom i sinkronizacijom dretvi najbolji izbor je korištenje programskog sučelja za višedretvenost. Takvo sučelje postoji za većinu programskih jezika kao funkcionalnost ugrađena u sam jezik ili u sklopu pomoćnih biblioteka.

Programsko sučelje pruža mogućnosti za stvaranje dretvi, pokretanje, čekanje da završe s izvođenjem i njihovo uništavanje. Pri stvaranju, dretvi je potrebno dodijeliti niz naredbi koje će izvršavati. U proceduralnim jezicima to je najčešće moguće pisanjem posebne funkcije dok je u objektno-orijentiranim jezicima potrebno napisati poseban razred koji implementira rad dretve.

Svi globalno vidljivi podaci automatski su vidljivi i svim dretvama te one mogu čitati i pisati po njima. Ipak, da bi program ispravno radio dretve ne bi smjele pristupati podatcima koje trenutno mijenja neka druga dretva. Zbog toga je potrebno koristiti sinkronizaciju koja će dretve koje žele pristupiti podacima koji se trenutno mijenjaju staviti na čekanje. Dretva koja čeka ne može obavljati nikakav koristan rad sve dok se podaci ne oslobode i dretvi bude dozvoljen nastavak rada. Dugi periodi čekanja mogu narušiti pa i posve eliminirati dobitke u brzini izvođenja koje donosi paralelno računanje.

Korištenje programskog sučelja za višedretvenost daje veliku fleksibilnost pri izradi algoritama za paralelno računanje jer je moguće točno odrediti broj dretvi i podatke koje će pojedine dretve obrađivati. Sinkronizaciju i komunikaciju između dretvi moguće je ostvariti na proizvoljan način. Algoritam i način rada dretvi moguće je prilagoditi samom problemu s ciljem povećanja učinkovitosti.

Zbog velike fleksibilnosti koju pruža programsko sučelje za višedretvenost, njegovo korištenje je kompliciranije i podložnije greškama od korištenja OpenMP-a ili MPI-a. Algoritam je potrebno implementirati tako da eksplicitno koristi dretve zbog čega je često nemoguće dobiti slijedni algoritam osim ako se posebno ne implementira. Greške u sinkronizaciji i komunikaciji dretvi teško je reproducirati i ispraviti. S obzirom na to da koristi isti način paralelizacije kao i OpenMP, programsko sučelje za višedretvenost postiže slično ubrzanje. Stoga ga ima smisla koristiti jedino ako korištenje OpenMP-a nije moguće ili praktično za problem koji se rješava.

5 Sinkronizacija dretvi

Već je nekoliko puta naglašeno kako je potrebno uvesti sinkronizaciju u slučaju da više dretvi pokušava pisati po istim podacima ili u slučaju da jedna dretva čita podatke dok ih druga dretva mijenja. Svi načini sinkronizacije u principu rade isto: dretvu koja trenutno nema pravo nastaviti izvođenje stavljaju na čekanje. Dretva koja čeka se ne izvršava i samim time ne troši sredstva (kao što je npr. procesorsko vrijeme), ali isto tako ne obavlja rad i produžuje vrijeme izvođenja programa. Jednom kada se ispune uvjeti za nastavak rada, dretva će biti vraćena u aktivno stanje.

Moguće je da dretva po povratku s čekanja opet nema uvjete za nastavak rada jer je druga dretva prije nje rezervirala pristup podacima, pa će se dretva morati vratiti na čekanje. Ako se dretva neprestano vraća na čekanje dolazi do situacije koja se naziva izgladnjivanje (engl. *starvation*). Dobar algoritam sinkronizacije nastojati će smanjiti vjerojatnost pojave izgladnjivanja i svakoj dretvi dati podjednak pristup sredstvima. Takav način sinkronizacije naziva se poštena (engl. *fair*) sinkronizacija. Brzo izvođenje metode sinkronizacije također je važno, pogotovo ako se sinkronizacija dretvi često koristi.

5.1 Spajanje

Spajanje je najosnovniji način sinkronizacije izvođenja dretvi. Za spajanje je potrebna jedna *radna* dretva koja obavlja neku radnju i druga *čekajuća* dretva koja mora pričekati prvu da završi kako bi mogla nastaviti sa svojim radom. Spajanje može uzrokovati potpuni zastoj (engl. *deadlock*) ako radna dretva nikad uredno ne završi s radom jer onda čekajuća dretva nikad neće moći nastaviti sa svojim izvođenjem. Pri pokušaju spajanja čekajuće i radne dretve moguće su dvije situacije:

- Ako je radna dretva već završila s izvođenjem, čekajuća dretva odmah nastavlja sa svojim radom.
- Ako je radna dretva još aktivna, čekajuća dretva stavlja se na čekanje. Tek kada radna dretva završi s radom čekajuća dretva će se vratiti u aktivno stanje.

5.2 Međusobno isključivanje

Međusobno isključivanje (engl. *mutual exclusion*) je način sinkronizacije koji osigurava da određeni dio programa može istovremeno izvoditi samo jedna dretva. Takvi dijelovi programa nazivaju se kritični odsječci (engl. *critical section*), a mehanizam koji osigurava međusobno isključivanje naziva se ključ (engl. *lock*). Dretva koja želi izvršiti takav dio programa mora zatražiti ulazak u kritični odsječak. Ako nijedna druga dretva trenutno ne izvršava taj dio programa, dretvi će biti dopušten ulazak i kritični odsječak će se smatrati zauzetim, a ključ koji čuva ulazak u odsječak će se smatrati zaključanim. U protivnom dretva će biti stavljana na čekanje. Po završetku izvođenja dijela programa označenog kao kritični odsječak dretva treba zatražiti izlazak iz njega. Ako nema dretvi koje čekaju na ulazak u odsječak, on se označava kao slobodan. Inače se odabire jedna od dretvi koje čekaju, vraća se u aktivno stanje i dopušta joj se ulazak.

Iako jednostavno, korištenje međusobnog isključivanja podložno je greškama koje često nisu lake za otkriti. Ništa ne sprječava dretvu da zaobiđe traženje dopuštenja za ulazak u kritični odsječak i jednostavno započne s njegovim izvođenjem. Ako je druga dretva trenutno unutar kritičnog odsječka, može doći do nepredvidljivog ponašanja programa. Ako dretva ne zatraži izlazak kada završi s izvođenjem, odsječak će ostati označen kao zauzet i nijedna dretva više neće moći ući u njega.

Ako dretve trebaju zaključati više od jednog ključa vrlo je važno da ih sve dretve zaključavaju u istom redoslijedu. Inače se može dogoditi da jedna dretva drži jedan ključ, a druga drugi te obje čekaju onu drugu da otpusti svoj. Ovakvo uzajamno čekanje dovesti će do potpunog zastoja rada programa. Zbog toga je umjesto više ključeva bolje koristi neki sofisticiraniji način sinkronizacije. Do potpunog zastoja može doći i ako dretva pokuša zaključati ključ koji je već prije sama zaključala. U tom slučaju će dretva čekati samu sebe da otpusti taj ključ što je naravno nemoguće. Ovaj problem moguće je riješiti korištenjem ključeva koji podržavaju višestruko zaključavanje, ako njihovu implementaciju pruža programsko sučelje za višedretvenost.

Ključevi su idealni za čuvanje sredstava koja su zajednička za sve dretve, pod uvjetom da je jedan ključ dovoljan da osigura ispravan rad programa. Za kompliciraniju sinkronizaciju bolje je koristiti neki od naprednijih načina.

5.3 Semafor

Semafor je način sinkronizacije dretvi koji se zasniva na korištenju cjelobrojnog brojača. Semafor se najčešće koristi za uređivanje pristupa sredstvima kojih ima u ograničenim količinama. U svom radu semafor se oslanja na ključeve ili nedjeljive operacije kako bi osigurao konzistentnost svoj stanja tj. vrijednosti brojača.

Pri stvaranju semafora njegov brojač postavlja se na neku početnu vrijednost. Dretve mogu uvećavati taj brojač i mogu ga pokušati umanjiti. Semafor ne smije dozvoliti da vrijednost brojača padne ispod nule. Kada dretva želi rezervirati dio sredstava za sebe ona će od semafora zatražiti da umanji vrijednost svog brojača kako bi odrazio novu količinu raspoloživih sredstava. Ako je vrijednost brojača veća ili jednaka iznosu za koji ga treba umanjiti semafor će to i učiniti te će dretvi dozvoliti da nastavi s izvođenjem. Ako vrijednost brojača nije dovoljna semafor će dretvu staviti na čekanje. Kada dretva koja je preuzela sredstva završi s njihovim korištenjem treba zatražiti od semafora uvećanje brojača. Vrijednost uvećanja trebala bi biti jednaka vrijednosti za koju je brojač prethodno umanjen. Ova operacije će uvijek uspjeti, te će semafor uvećati brojač i dretvi dozvoliti da nastavi s izvođenjem.

Kada semafor uveća brojač postoji mogućnost da je nova vrijednost brojača dovoljno velika za neku od dretvi koje čekaju na semaforu. Postoje dva načina na koje semafor može dretvi koja čeka dopustiti pristup traženim sredstvima:

- Semafor može sve dretve koje čekaju vratiti u aktivno stanje. Dretve će, redoslijedom koji nije određen, pokušati ponovno smanjiti vrijednost brojača. Ako dretva ne uspije, jer nova vrijednost opet nije dovoljna ili je neka druga dretva prije rezervirala sredstva, dretva će se vratiti na čekanje.
- Semafor može sam izabrati jednu ili više dretvi kojima je novo stanje dovoljno te ih vratiti u aktivno stanje. Korištenjem ovog načina dretvama je moguće dodijeliti prioritete tako da će semafor nastojati dretve višeg prioriteta prije propustiti.

Potencijalni problem sa semaforima jest što ne reguliraju da dretve moraju uvećati brojač za istu vrijednost za koju su ga smanjile. Problem je moguće riješiti uvođenjem posebnih objekata koji pamte količinu uzetih sredstava. Dretve ne manipuliraju izravno s brojačem već preko tih posebnih objekata što garantira konzistentnu vrijednost brojača.

5.4 Monitor

Monitor je objekt za sinkronizaciju dretvi koji, zajedno s uvjetnim varijablama, pruža najviše mogućnosti za kontrolu izvršavanja dretvi. Za razliku od isključivog izvođenja i semafora, monitor omogućava postavljanje proizvoljnih uvjeta za nastavak rada i čekanje dretvi. Osnovno svojstvo monitora je da istovremeno samo jedna dretva može raditi s njime.

Uvjete varijable su posebni objekti za stavljanje dretvi na čekanje kada uvjet za njihov nastavak rada nije zadovoljen i vraćanje u aktivno stanje jednom kada uvjet postane zadovoljen. Za rad uvjetne varijable potrebne su barem dvije dretve – jedna koje ispituje uvjet i ne može nastaviti dok on ne bude zadovoljen i druga dretva čiji će rad rezultirati zadovoljenim uvjetom. Kako bi se osigurao ispravan rad uvjetnih varijabli, svaka je povezana s točno jednim monitorom te ju je moguće koristiti samo pomoću tog monitora.

Dretva koja treba raditi s monitorom to počinje pokušajem operacije ulaska u monitor. Ako neka druga dretva nije već zauzela monitor operacija će uspjeti te će se monitor smatrati zauzetim. Ako je monitor već zauzet dretva će se staviti na čekanje. Dretva će postati opet aktivna i može ponoviti pokušaj tek kada dretva koja je zauzela monitor njega napusti. Jednom kada uspije ući u monitor dretva provjerava uvjete koji su nužni za daljnji rad. Ako su uvjeti zadovoljeni dretva napušta monitor i nastavlja s radom, ali ako nisu dretva se stavlja na čekanje korištenjem uvjetne varijable. Stavljanjem na čekanje dretva privremeno napušta monitor tako da ga mogu koristiti druge dretve.

Dretva koja svojim radom ispuni uvjet za nastavak rada neke druge dretve mora tu dretvu vratiti u aktivno stanje ako ona trenutno čeka. Da bi to učinila dretva prvo mora ući u monitor, vratiti u aktivno stanje jednu ili više dretvi pomoću uvjetne varijable i napustiti monitor. Slično kao i kod semafora postoje dva načina za vraćanje dretvi u aktivno stanje:

- Uvjetna varijabla može u aktivno stanje vratiti sve dretve koje čekaju te će one sve pokušati ući u monitor i opet provjeriti svoje uvjete. Redoslijed vraćanja dretvi u aktivno stanje često je nasumičan.
- Uvjetna varijabla može u aktivno stanje vratiti samo jednu dretvu. Ovo se najčešće koristi kada je uvjet dretve za nastavak rada sigurno zadovoljen.

5.5 Ograda

Ograda (engl. *barrier*) je način sinkronizacije dretvi koji osigurava da skupina dretvi nastavi izvođenje od istog mjesta u programu. Najčešće se koristi kada skupina dretvi obavlja niz operacija nakon kojih sve dretve moraju privremeno stati s radom kako bi se međusobno uskladile i preuzele nove podatke ili zadatke koje treba obraditi. Kako je dretvama potrebna različita količina vremena za obavljanje pojedinih zadataka praktički je nemoguće da sve dretve završe u isto vrijeme. Stoga je potrebno implementira mehanizam koji će dretve koje prije završne s radom staviti na čekanje. Tek kada i posljednja dretva završi s radom sve dretve mogu nastaviti dalje. Ako je potrebno obaviti usklađivanje podataka to će biti prepušteno jednoj dretvi, obično onoj koja posljednja završi, i tek kada je usklađivanje obavljeno sve dretve će istovremeno nastaviti s izvođenjem.

Ograda je implementirana slično kao semafor – sadrži brojač čija je početna vrijednost jednaka broju dretvi koje trebaju proći kroz ogradu. Svaki put kada jedna od dretvi pokuša proći brojač će se umanjiti za jedan. Ako nakon toga brojač nije jednak nuli znači da još uvijek ima dretvi koje nisu završile s radom. U tom slučaju dretva koja je pokušala proći će biti stavljena na čekanje. Tek kada posljednja dretva pokuša proći vrijednost brojača će pasti na nulu. Tada se prvo izvodi usklađivanje dretvi i potom se sve dretve vraćaju u aktivno stanje i nastavljaju s radom. Ograda radi suprotno od semafora – ako vrijednost brojača nije jednaka nuli dretva se stavlja na čekanje, a kada postane jednaka nuli sve dretve su vraćaju u aktivno stanje.

Ogradu je moguće koristiti u izmijenjenoj varijanti koja je korisnija za neke situacije. Veća fleksibilnost može se postići ako se odvoje operacije pokušaja prolaska kroz ogradu i umanjenja brojača. To omogućuje jednoj dretvi da umanji vrijednost brojača bez da bude stavljana na čekanje dok neka druga dretva može čekati na ogradi bez da utječe na vrijednost brojača. Ovo je korisno ako npr. jedna ili više dretvi moraju čekati da se obavi određeni broj zadataka jer one onda mogu čekati na ogradi. Vrijednost brojača ograde tada predstavlja broj zadataka koji moraju biti završeni. Dretve koje obavljaju zadatke za svaki završen zadatak umanje brojač bez da budu bez potrebe stavljene na čekanje. Jednom kada se obavi potreban broj zadataka ograda će propustiti sve dretve koje su bile na čekanju.

5.6 Nedjeljive operacije

Promjena vrijednosti varijable inače se mora odviti u najmanje tri koraka. U prvom koraku čita se vrijednost varijable iz memorije, u drugom se računa nova vrijednost, a u trećem zapisuje natrag u memoriju. S obzirom na to da dretva koja mijenja varijablu može biti prekinuta između bilo koja dva koraka od dretve koja također mijenja tu istu varijablu, može se dogoditi na jedna dretva nepredviđeno poremeti operaciju druge dretve. Jedan od načina osiguravanja nesmetane izmjene vrijednosti varijable jest korištenje jednog od načina sinkronizacije koji će osigurati da istovremeno samo jedna dretva mijenja varijablu. Nedostatak je velika cijena sinkronizacije – korištenje ključa, semafora ili monitora je za nekoliko redova veličine sporije od same operacije izmjene vrijednosti varijable.

Nedjeljive (engl. *atomic*) operacije su procesorske instrukcije koje omogućavaju čitanje i modificiranje memorije unutar jedne neprekinute operacije. Takve instrukcije su zapravo kombinacija instrukcije za čitanje, modificiranje i pisanje s razlikom da procesor garantira da trenutna dretva neće biti prekinuta tijekom izvođenja te kombinacije. Nedjeljive operacije su korisne jer omogućavaju dretvama da praktički istovremeno mijenjaju istu varijablu bez potrebe za korištenjem sporijih načina sinkronizacije.

Pri izvršavanju nedjeljive operacije procesor mora na neki način to signalizirati ostalim procesorima i procesorskim jezgrama. Ako neki drugi procesor tada pokuša izvršiti modificiranje memorije morati će privremeno zaustaviti svoje izvođenje dok prvi procesor ne završi započetu operaciju. Kako su nedjeljive operacije instrukcije tek malo dužeg trajanja od običnih instrukcija, to čekanje je uvijek vrlo kratko.

Tri najčešće nedjeljive operacije su *test and set* (TAS), *fetch and add* (FAA) i *compare and set* (CAS). TAS postavlja vrijednost varijable na jedan i vraća njezinu prijašnju vrijednost. FAA uvećava vrijednost za jedan i vraća staru vrijednost. CAS koristi dvije vrijednosti – očekivanu i novu. Ako je trenutna vrijednost varijable jednaka očekivanoj, varijabla se postavlja na novu vrijednost. Operacija kao rezultat vraća uspješnost pokušaja promjene vrijednosti.

Osim za brzu sinkroniziranu izmjenu vrijednosti varijabli, nedjeljive operacije vrlo su korisne i za implementaciju brzih sinkroniziranih struktura podataka za istovremeno korištenje od više dretvi.

6 Model paralelne simulacije

Paralelna simulacija gibanja nebeskih tijela sastoji se od skupa algoritama za rješavanje pojedinih problema. Od mogućih načina rješavanja izabrani su sljedeći algoritmi:

- algoritam Barnes-Hut za računanje rezultante sila koje djeluju na tijela,
- algoritam za dekompoziciju prostora pomoću strukture *quadtree* stabla,
- Eulerova metoda za numeričku integraciju diferencijalnih jednadžbi,
- slijedni algoritam radi ocjenjivanja učinkovitosti paralelizacije,
- metoda paralelnog računanja pomoću programskog sučelja za višedretvenost,
- statički i dinamički način raspodjele podataka po dretvama i
- metode semafora, ograde i nedjeljivih operacija za sinkronizaciju dretvi.

6.1 Model nebeskih tijela

Radi lakšeg računanja izabran je dvodimenzionalni prostor u kojemu se nalaze tijela. Samim time položaj tijela određen je s dva realna broja – njegovom x i y koordinatom. Osim položaja, potrebno je voditi računa i o trenutnoj brzini i ubrzanju tijela. Kako je za ta oba podatka potrebno znati njihov iznos i smjer najlakše ih je predstaviti kao vektore. Konačno, za svako tijelo potrebno je definirati i njegovu masu koja će ostati konstantna tijekom simulacije.

Kako se tijela privlače gravitacijskim silama, moguće je da se dva tijela približe toliko blizu da sile kojima djeluju jedno na drugo postanu tolikog iznosa da njihovo računanje unosi neprihvatljivo veliku pogrešku u račun. Zbog toga se izraz za računanje sila dopunjuje faktorom gušenja ε koji će oslabiti takve sile te povećati stabilnost sustava:

$$\ddot{r}_i = -\mathrm{G} \sum_{j=1;\, j\neq i}^{N} \frac{m_j (r_i - r_j)}{\left(\left| r_i - r_j \right|^2 + \varepsilon^2 \right)^{3/2}} \tag{6.1}$$

6.2 Dekompozicija prostora

Algoritam za dekompoziciju prostora pomoću strukture *quadtree* stabla počinje tako da cijeli prostor predstavi jednom ćelijom te redom dodaje tijela u strukturu. Svaka ćelija je predstavljena jednim čvorom stabla i sadrži sljedeće podatke:

- položaj geometrijskog centra ćelije,
- duljinu stranice ćelije,
- ukupnu masu svih tijela sadržanih unutar ćelije,
- položaj centra mase sadržanih tijela i
- popis čvorova djece.

Nakon stvaranja korijenskog čvora algoritam redom pokušava dodati sva tijela u taj korijenski čvor. Algoritam dodavanja tijela može se opisati sljedećim pseudo-kodom:

```
Ako trenutni čvor već sadrži tijelo
  Stvori čvorove djecu trenutnog čvora
  Odredi kojem čvoru djetetu pripada trenutno tijelo čvora
  Dodaj tijelo u taj čvor
  Odredi kojem čvoru djetetu pripada novo tijelo
  Dodaj novo tijelo u taj čvor
Inače
  Dodaj novo tijelo u trenutni čvor
```

Nakon što se završi s dodavanjem svih tijela, unutarnji čvorovi *quadtree* stabla neće sadržavati tijela već samo svoju djecu, dok će tijela biti sadržana u listovima stabla. Kako se djeca čvorova stvaraju samo pri pokušaju dodavanja tijela u čvor koji već sadrži tijelo, *quadtree* stablo neće sadržavati veliki broj nepotrebnih čvorova koji bi predstavljali prazan prostor. Isto tako raspodjela tijela u prostoru će se odraziti na strukturu stabla. Visina dijela stabla koji predstavlja prostor u kojemu se nalazi mali broj tijela će biti proporcionalno manja. Velika gustoća raspodjele tijela će uzrokovati veliki broj dodavanja u čvorove koji već imaju tijela zbog čega će algoritam morati stvoriti dodatne čvorove i samim time povećavati visinu stabla. Nakon završetka gradnje, algoritam obilazi stablo od listova prema korijenu te računa ukupnu masu svih tijela sadržanih u čvoru i njegovoj djeci te položaj centra mase tijela.

6.3 Paralelna verzija algoritma Barnes-Hut

Za svako tijelo u sustavu algoritam Barnes-Hut računa aproksimaciju rezultante sila kojima sva ostala tijela djeluju na njega. Algoritam obilazi stablo sagrađeno dekompozicijom prostora i nastoji pronaći ćeliju koja sadrži tijela koja su dovoljno udaljena da se mogu aproksimirati jednim tijelom. Da bi takva aproksimacija bila dobra ćelija mora zadovoljavati *kriterij otvaranja*: omjer između duljine stranice ćelije i udaljenost tijela do centa mase ćelije mora biti manji od unaprijed definiranog *kuta otvaranja*. Kao vrijednost kuta otvaranja često su uzima vrijednost 0,75 jer pruža dobar omjer između brzine računanja i točnosti simulacije (Salmon et al., 1994).

Algoritam počinje obilazak s korijenom stabla i provjerava zadovoljava li kriterij otvaranja. S obzirom na to da korijenski čvor predstavlja cijeli prostor, kriterij ne može biti zadovoljen te se algoritam rekurzivno spušta na čvorove djecu i na njima ispituje kriterij otvaranja. Algoritam nastavlja s rekurzivnom provjerom djece dok ne pronađe ćeliju koja ili zadovoljava kriterij otvaranja ili je predstavljana listom stabla. U tom slučaju algoritam sva tijela koja su sadržana u ćeliji aproksimira jednim tijelom koje je smješteno u centru mase ćelije i ima masu jednakoj ukupnoj masi svih tijela. Algoritam računa silu kojom to aproksimirano tijelo djeluje, dodaje je na rezultantu sila i završava s obradom tog čvora (djeca se ne ispituju). Kada obradi sve čvorove čije ćelije zadovoljavaju kriterij otvaranja, algoritam računa ubrzanje tijela uzrokovano rezultantom sila te ponavlja postupak računanja za sljedeće tijelo.

Jednom kada se izračunaju vrijednosti ubrzanja svih tijela za trenutni korak simulacije, postupkom numeričke integracije računaju se novi položaji i brzine tijela. Iz fizike je poznato da je brzina prva derivacija položaja, a ubrzanje druga derivacija položaja tj. prva derivacija brzine. Na temelju toga postavljaju se sljedeće dvije obične diferencijalne jednadžbe prvog reda:

$$x'(t) = v(t) \implies x_n = v_n \cdot h \tag{6.2}$$

$$v'(t) = a(t) \implies v_n = a_n \cdot h \tag{6.3}$$

Eulerovom metodom prvo se numerički rješava jednadžba (6.2) za položaj, a potom jednadžba (6.3) za brzinu.

S obzirom na to da računanje sila koje djeluju na jedno tijelo ne utječe na to isto računanje za druga tijela, algoritam Barnes-Hut moguće je ubrzati korištenjem više dretvi koje će paralelno računati iznose sila za više tijela odjednom. Paralelni simulator sastoji se od jedne glavne dretve i više radnih dretvi. Glavna dretva prima zahtjev za računanjem sila te pokreće i koordinira radne dretve, dok su radnje dretve zadužene za računanje sila.

Pri inicijalizaciji simulatora osim glavne stvara se N radnih dretvi, gdje je N podesivi parametar, te jedan semafor i ograda za sinkronizaciju. Brojač ograde postavlja se na $N + 1$, a početna vrijednost semafora na nulu te sve dretve kreću s izvođenjem. Sve radne dretve u petlji izvode sljedeće naredbe zadane pseudo-kodom:

```
Pokušaj proći kroz ogradu
Izračunaj rezultante sila
Uvećaj brojač semafora za jedan
```

S obzirom na to da je brojač ograde postavljen na $N + 1$, a radnih dretvi ima samo N sve radne dretve će biti stavljene na čekanje odmah pri početku rada. Ovo osigurava da dretve počnu s računanjem sila tek kada glavna dretva primi zahtjev za računanjem. Kada dođe takav zahtjev glavna dretva izvršava sljedeće naredbe:

```
Prođi kroz ogradu
Pokušaj umanjiti brojač semafora za N
```

Kada glavna dretva pokuša proći kroz ogradu brojač ograde pasti će na nulu (zbog N radnih dretvi i jedne glavne) te će ograda vratiti sve radne dretve u aktivno stanje. S obzirom na to da je početna vrijednost semafora nula, glavna dretva neće moći njegov brojač umanjiti za N te će biti stavljena na čekanje. Kada radna dretva završi s računanjem sila uvećati će brojač semafora za 1 te će pokušati proći kroz ogradu zbog čega će biti opet stavljena na čekanje. Kada sve radne dretve završe s računanjem vrijednost brojača semafora biti će jednaka N te će glavna dretva biti vraćena u aktivno stanje. Glavna dretva će nakon toga umanjiti vrijednost brojača semafora na nulu te javiti dijelu programa koji je zatražio računanje da je ono obavljeno. Kako su sve radne dretve vraćene na čekanje na ogradu, a vrijednost semafora je opet jednaka nuli stanje je jednako onom koje je bilo prije počinjanja računanja te je simulator spreman za obradu sljedećeg zahtjeva za računanjem iznosa gravitacijskih sila.

6.4 Raspodjela podataka

Način raspodjele podataka određuje način kojim će se tijela dodjeljivati radnim dretvama na obradu. S obzirom na to da ne zahtijevaju sva tijela jednako vremena za računanje sila, jer je vrijeme računanja proporcionalno visini stabla na kojoj se tijelo nalazi, način raspodjele može značajno utjecati na ubrzanje dobiveno paralelnim računanjem.

Jednostavniji način je statička raspodjela podataka. Pri inicijalizaciji simulatora svako tijelo se dodjeli jednoj dretvi tako svaka dretva dobije jednak (ili približno jednak) broj tijela. S obzirom na to da svaka dretva unaprijed zna s kojim podacima treba raditi nema potrebe za međusobnom komunikacijom ni dodatnom sinkronizacijom dretvi. Loša strana ovog načina je što ne uzima u obzir vrijeme potrebno za računanje rezultante sila pojedinih tijela. Ako jednoj dretvi budu dodijeljena tijela koja zahtijevaju više vremena nego tijela drugih dretvi, tada može doći do situacije da druge dretve završe dosta prije i moraju čekati više opterećenu dretvu umjesto da obavljaju koristan rad.

Dinamička raspodjela podataka je način koji se može prilagoditi promjenjivim vremenima računanja rezultanti sila pojedinih tijela. Tijekom računanja sve dretve dijele jedan brojač koji sadrži indeks sljedećeg neobrađenog tijela. Dretva koja je slobodna za rad treba pročitati vrijednost brojača, uvećati ga čime će ostalim dretvama javiti da je tijelo s tim indeksom rezervirano te početi s računanjem sila koje djeluju na to tijelo. Ovime se osigurava da će sve dretve raditi dok god ima neobrađenih tijela te je izbjegnuta situacija da su neke dretve dosta više opterećene od drugih.

Vrijeme potrebno za sinkronizaciju zbog čitanja i mijenjanja vrijednosti brojača može se smanjiti korištenjem nedjeljivih operacija. Dretva treba pročitati trenutnu vrijednost brojača i izvršiti *compare and set* na vrijednost uvećanu za jedan. Ako uspije, može početi s obradom tijela čiji je indeks jednak pročitanoj vrijednosti. Ako je neka druga dretva prije rezervirala tijelo s tim indeksom, *compare and set* neće uspjeti pa dretva treba ponoviti pokušaj. Vrijeme utrošeno na sinkronizaciju moguće je dodatno smanjiti ako se poveća zrnatost raspodjele podataka. Umjesto rezerviranja tijela jedno po jedno, dretve mogu odjednom rezervirati veći broj tijela. U tom slučaju brojač se više ne uvećava za jedan već za veličinu zrnatosti. Povećanje zrnatosti smanjuje broj nedjeljivih operacija koje je potrebno izvesti, ali povećava vjerojatnost da će neke dretve biti više opterećene.

7 Ostvarenje paralelne simulacije

Programsko rješenje implementirano je u Javi, objektno-orijentiranom programskom jeziku čija standardna biblioteka sadrži potporu za upravljanje dretvama i sinkronizaciju dretvi ključevima, semaforima, ogradama, nedjeljivim operacijama i drugim načinima. Dretve i svi načini sinkronizacije predstavljeni su odgovarajućim razredima.

Sva logika simulacije, osim računanja sila, implementirana je u apstraktnom razredu *Simulator*. Razred *Simulator* može izgraditi *quadtree* stablo, izračunati ukupnu masu i položaj centra mase svakog čvora stabla te numerički integrirati nove položaje i brzine nebeskih tijela. Razredi koji nasljeđuju *Simulator* moraju implementiraju metodu računanja sila kako bi se simulator mogao koristiti. S obzirom na to da je *Simulator* ostvaren kao dretva, moguće je zatražiti prekid njegovog rada (engl. *interrupt*). Razred *Simulator* provjerava stanje zahtjeva za prekid tijekom svih koraka rada simulacije te ubrzo nakon primanja takvog zahtjeva prekida izvođenje simulacije i završava s radom.

Razred *SequentialSimulator* implementira slijedni algoritam računanja sila. Razred prolazi po svim tijelima te za svako tijelo izvodi algoritam Barnes-Hut kako bi izračunao aproksimaciju rezultante sila. Vrijeme potrebno slijednom algoritmu za računanje koristi se kao osnova za mjerenje ubrzanja paralelnih algoritama.

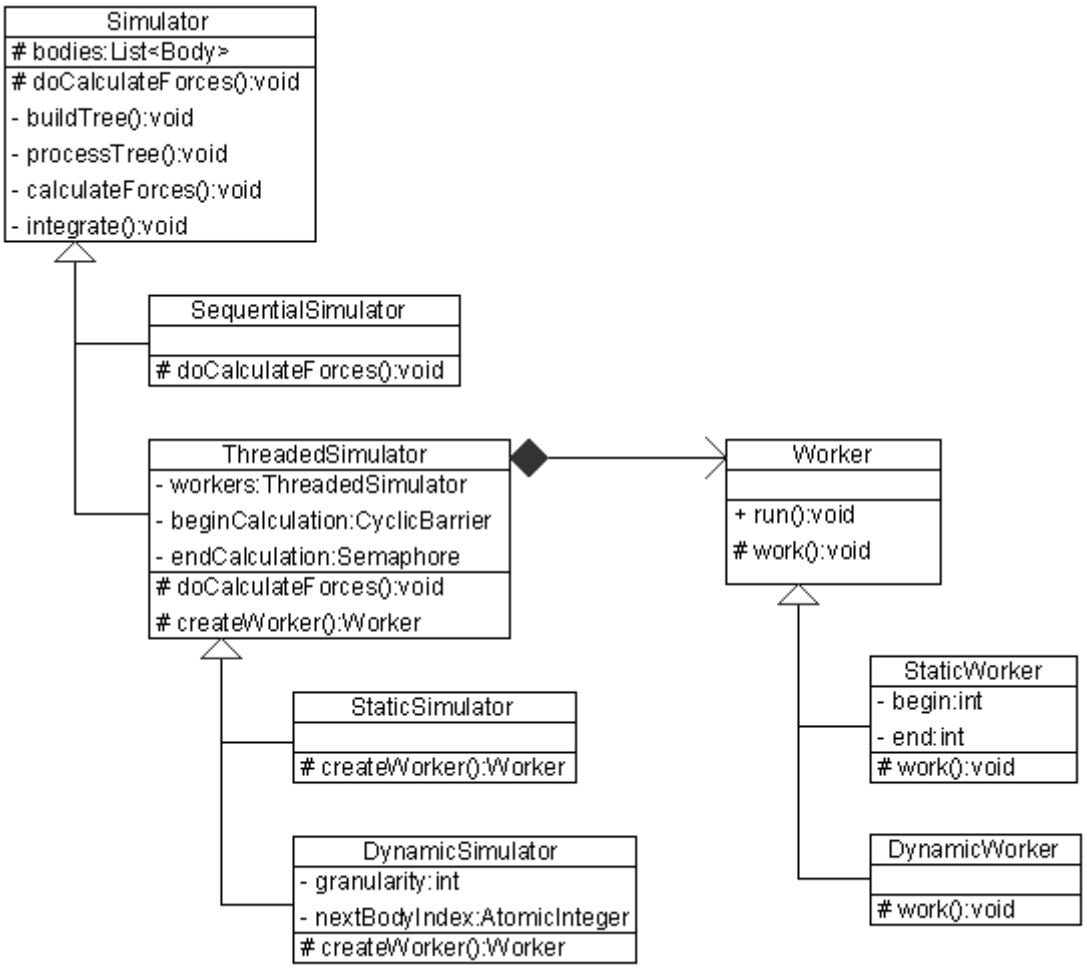
Razred *ThreadedSimulator* predstavlja osnovu za izvođenje paralelnih algoritama. *ThreadedSimulator* sadrži reference na instance radnih dretvi, objekt razreda *Semaphore* i razreda *CyclicBarrier*. Dodatno, *ThreadedSimulator* sadrži definiciju ugniježđenog apstraktnog razreda *Worker* koji predstavlja osnovu radne dretve. Razredi koji nasljeđuju *ThreadedSimulator* definiraju način stvaranja i korištenja radnih dretvi. Kako je razred radne dretve ugniježđen u razred simulatora, sve instance tih dretvi imaju izravan pristup objektima ograde i semafora koji se koriste za sinkronizaciju rada.

StaticSimulator predstavlja paralelni način računanja korištenjem statičke raspodjele podataka. Pri stvaranju radnih dretvi za svaku se određuje indeks prvog i zadnjeg tijela koje dretva treba obraditi.

DynamicSimulator predstavlja paralelni način računanja korištenjem dinamičke raspodjele podataka. Razred sadrži objekt tipa *AtomicInteger* koji predstavlja cijeli broj nad kojim je moguće izvršavati nedjeljive operacije. Osim izravnog korištenja metode *compare and set*, *AtomicInteger* pruža operacije koje se oslanjaju na tu metodu kao što su postavljanje nove vrijednosti te smanjenje ili povećanje za proizvoljnu vrijednost. Tijekom računanja radne dretve koriste taj brojač kako bi odredile indeks sljedećeg neobrađenog tijela. Radne dretve u koracima uvećavaju brojač za iznos granulacije ta računaju sile za sva tijela čiji se indeksi nalaze između vrijednosti brojača prije i poslije uvećanja.

7.1 UML dijagram razreda

Slika 7.1 prikazuje UML dijagram razreda koji su korišteni za ostvarenje paralelnog računanja gravitacijskih sila.



Slika 7.1: UML dijagram razreda

8 Rezultati

Za ispitivanje učinkovitosti paralelnog računanja korišteno je nekoliko različitih ispitnih slučajeva koji su testirani korištenjem redom: slijednog algoritama, paralelnog algoritma sa statičkom raspodjelom i paralelnog algoritma s dinamičkom raspodjelom uz granulaciju od 25%, 10%, 5% i 0%. Granulacija od 0% znači da dretve obrađuju tijela jedno po jedno. Svaki je test izveden deset puta te je kao konačan rezultat uzeta aritmetička sredina svih izmjerenih vremena u minutama.

Ispitane su tri vrste opterećenja: simetrično koje će dretve podjednako opteretiti, asimetrično koje će nastojati jednu dretvu opteretiti više nego ostale te promjenjivo opterećenje koje testira simuliranje sustava čija se struktura često mijenja.

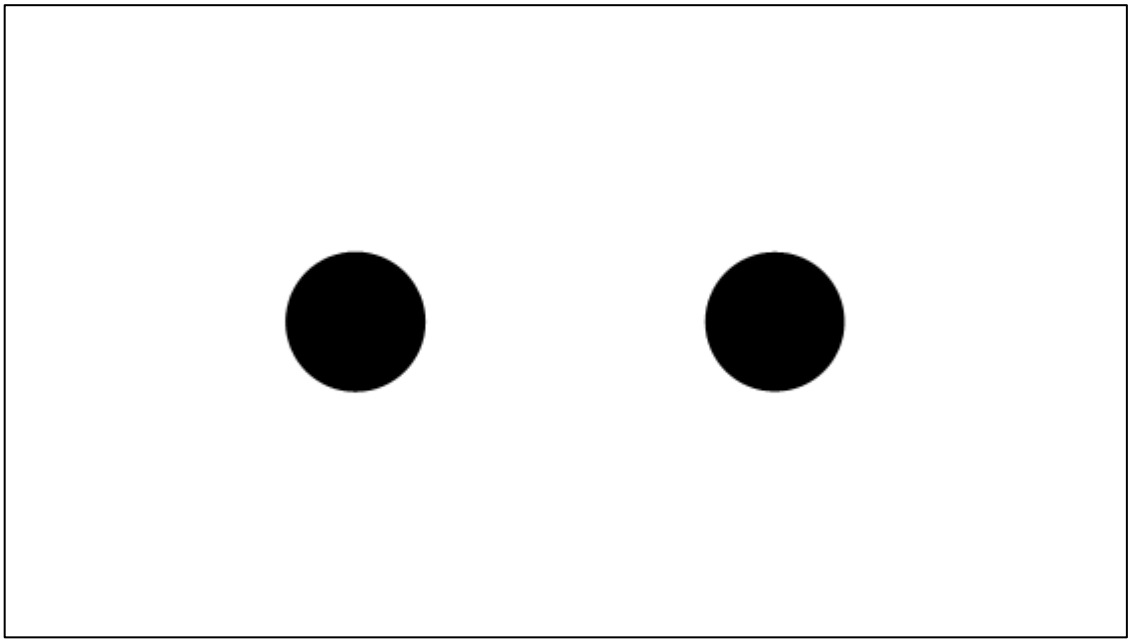
Prvi skup testova proveden je na procesoru s dvije jezgre, a za izvođenje paralelnih algoritama korištene su dvije dretve. Za svaki test prikazana je slika ispitnog slučaja te postotci vremena potrošeni na gradnju i obradu stabla kao i za računanje i integraciju sila. Dodatno, prikazano je i apsolutno ubrzanje u odnosu na slijedni algoritam.

Drugi skup testova proveden je na procesu s četiri jezgre, a za izvođenje paralelnih algoritama korištene su redom dvije, tri i četiri dretve. Za svaki je test, osim slike ispitnog slučaja, prikazan i graf apsolutnog ubrzanja paralelnih algoritama u ovisnosti o broju korištenih dretvi.

Ispitni slučajevi konzistentno pokazuju bolje rezultate pri korištenju dinamičke raspodjele podataka s malom zrnatošću. Statička raspodjela podataka daje ubrzanje slično dinamičkoj raspodjeli samo u slučaju idealnog simetričnog opterećenja, dok u svim ostalim slučajevima jednu dretvu optereti više nego ostale.

Paralelni algoritmi ne postižu idealno linearno ubrzanje. Razlog tomu je što se paralelno ne izvodi cijela simulacija već samo računanje sila koje u slijednom algoritmu zauzima približno 93% vremena. Drugi razlog je što dretve intenzivno koriste memoriju (zbog velike količine podataka koje treba obraditi) što ometa primjenu priručnih spremnika i samim time usporava čitanje memorije i izvođenje simulacije.

8.1 Simetrično opterećenje dvije jezgre



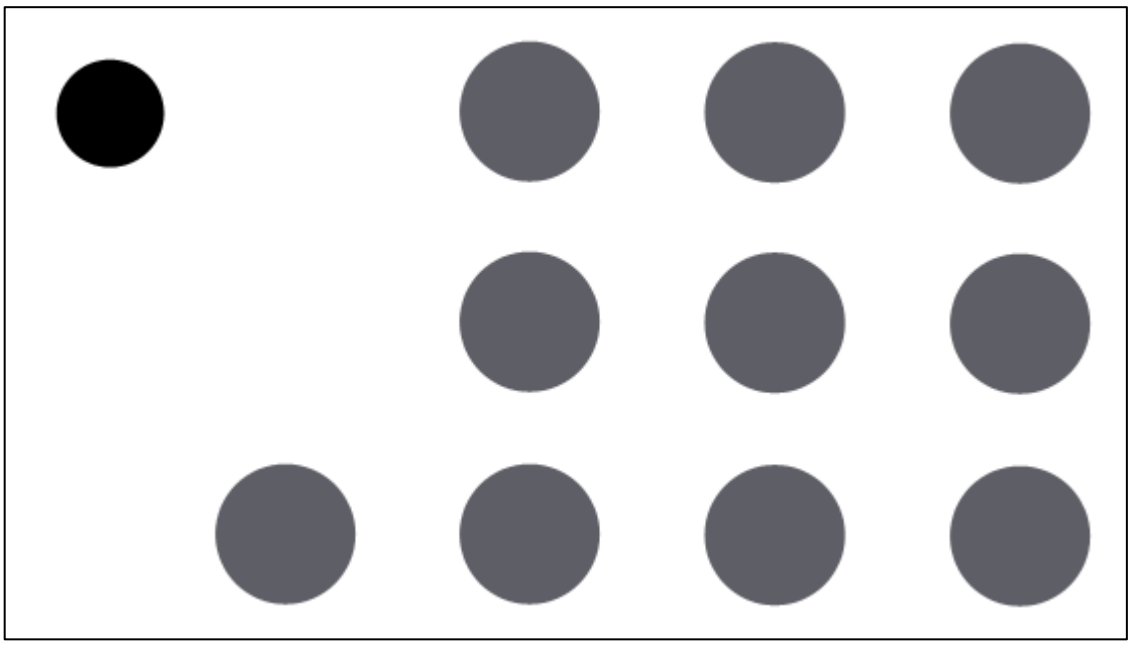
Slika 8.1: Ispitni slučaj simetričnog opterećenja dvije jezgre

Tablica 8.1: Rezultati ispitnog slučaja simetričnog opterećenja dvije jezgre

	Ukupno vrijeme	Gradnja stabla	Obrada stabla	Računanje sila	Integracija sila	Apsolutno ubrzanje
Slijedno	26:54	5,36	0,64	93,43	0,57	–
Statički	15:10	8,81	1,36	88,34	1,49	1,77
Dinamički (25%)	15:12	9,64	1,14	88,10	1,12	1,77
Dinamički (10%)	15:12	8,78	1,72	88,85	0,65	1,77
Dinamički (0%)	15:11	9,23	1,35	88,31	1,11	1,77

Ispitni slučaj simetričnog opterećenja sastoji od dvije galaksije jednakih masa, radijusa i broja zvijezda. Svaka galaksija ima 150.000 uniformno raspoređenih zvijezda. Kako će statička raspodjela dodijeliti po jednu galaksiju svakoj dretvi, a dinamička će također dodijeliti tijela iz sličnih dijelova galaksije opterećenja dretvi će biti podjednaka te samim time i apsolutna ubrzanja algoritama.

8.2 Asimetrično opterećenje dvije jezgre



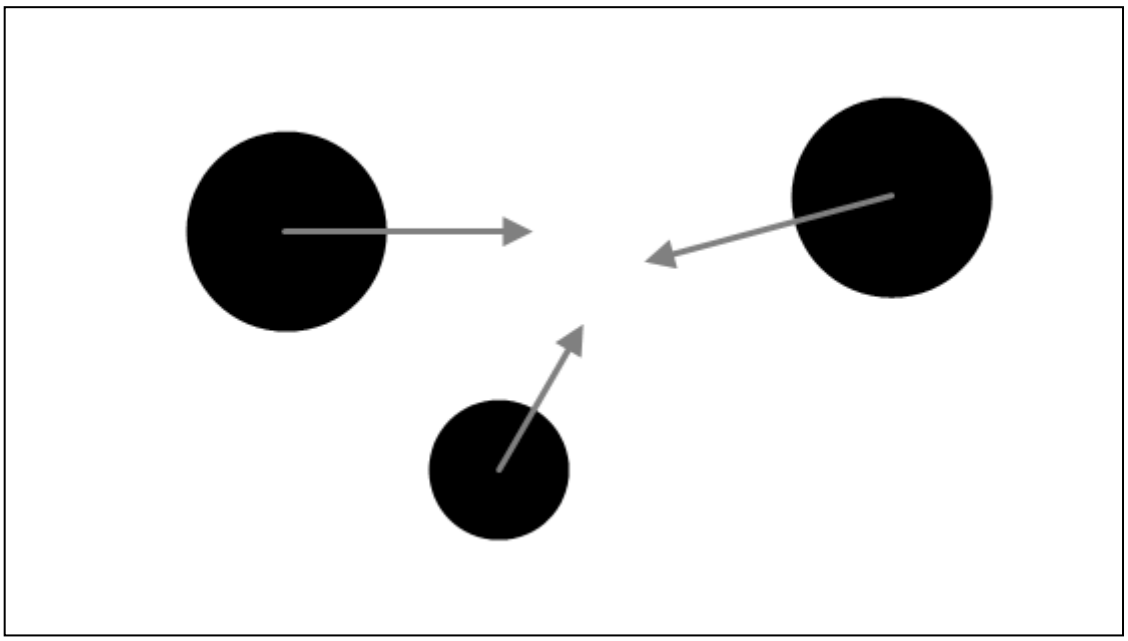
Slika 8.2: Ispitni slučaj asimetričnog opterećenja dvije jezgre

Tablica 8.2: Rezultati ispitnog slučaja asimetričnog opterećenja dvije jezgre

	Ukupno vrijeme	Gradnja stabla	Obrada stabla	Računanje sila	Integracija sila	Apsolutno ubrzanje
Slijedno	26:54	4,91	0,51	93,45	1,13	–
Statički	16:54	7,51	0,81	91,13	0,55	1,59
Dinamički (25%)	16:24	7,69	0,57	90,90	0,84	1,64
Dinamički (10%)	15:30	7,97	1,01	90,48	0,54	1,74
Dinamički (0%)	15:12	8,15	1,57	89,86	0,42	1,77

Ispitni slučaj asimetričnog opterećenja sastoji se od jedne galaksije sa 150.000 zvijezda (označene crnom bojom) i deset manjih galaksija od 15.000 zvijezda (označenih sivom bojom). Statička raspodjela će veću galaksiju dodijeliti prvoj dretvi, a male drugoj. Kako je jedna velika galaksija zahtjevnija, dretve neće biti jednako opterećene i ubrzanje će biti manje. Korištenjem dinamičke raspodjele i manje zrnatosti postiže se ujednačenije opterećenje dretvi.

8.3 Promjenjivo opterećenje dvije jezgre



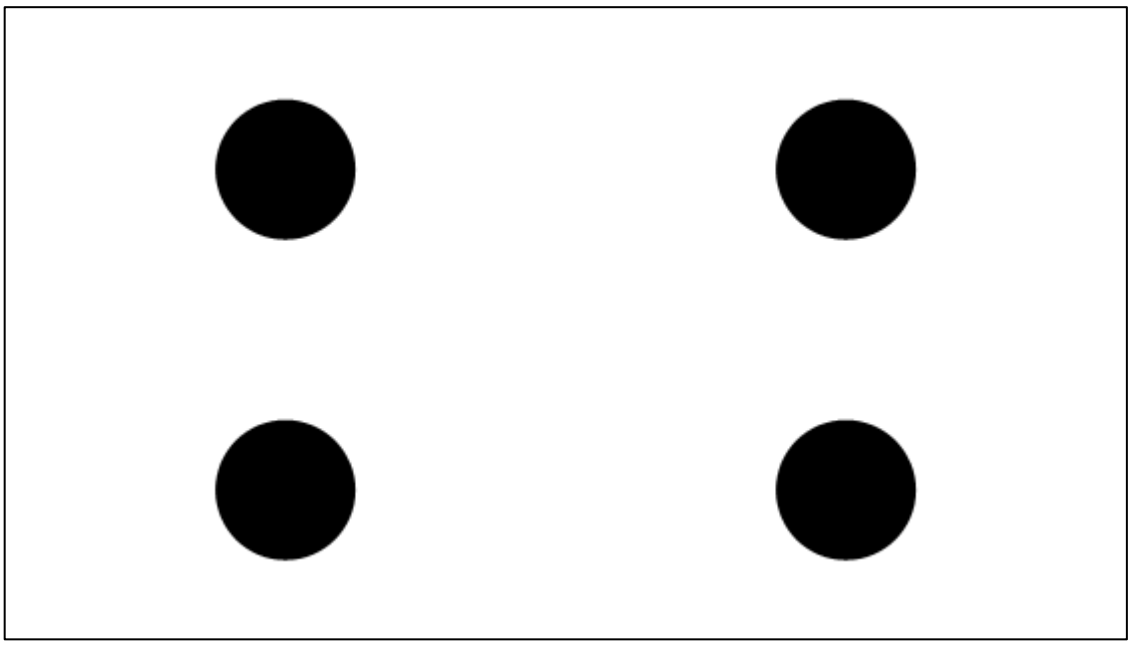
Slika 8.3: Ispitni slučaj promjenjivog opterećenja dvije jezgre

Tablica 8.3: Rezultati ispitnog slučaja promjenjivog opterećenja dvije jezgre

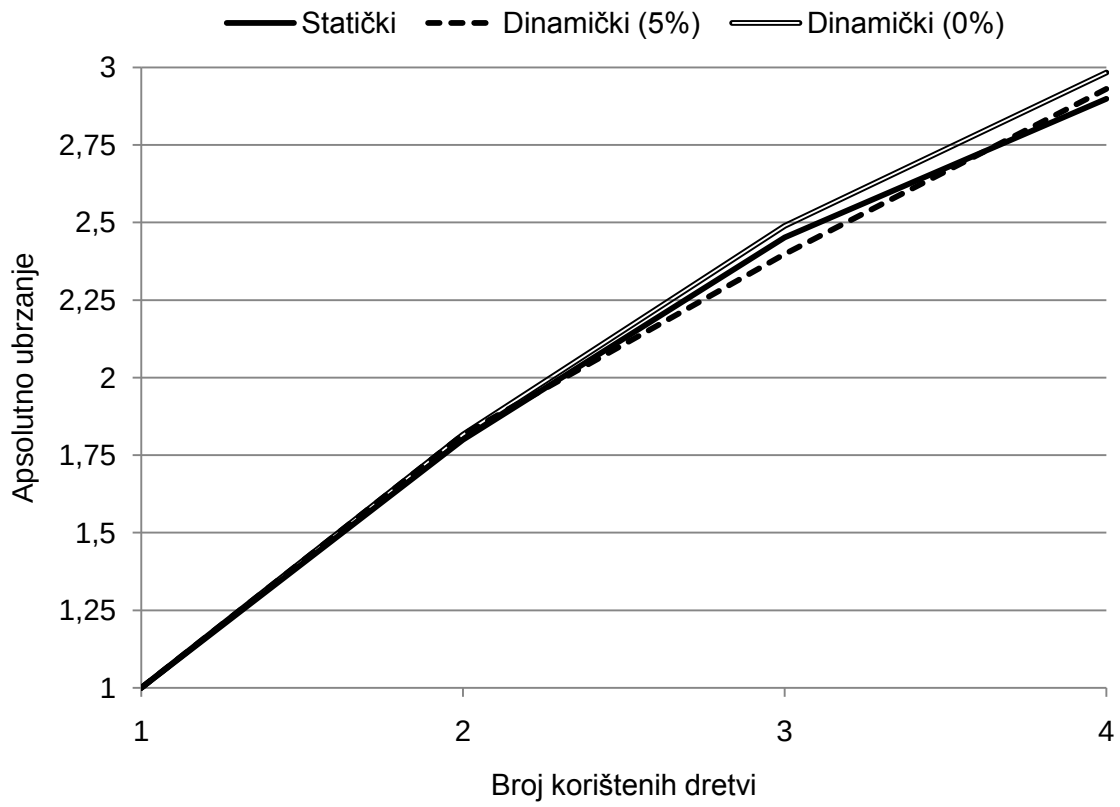
	Ukupno vrijeme	Gradnja stabla	Obrada stabla	Računanje sila	Integracija sila	Apsolutno ubrzanje
Slijedno	27:30	4,95	0,69	94,05	0,31	–
Statički	17:24	7,34	1,46	90,72	0,48	1,58
Dinamički (25%)	16:24	7,95	1,82	89,66	0,57	1,68
Dinamički (10%)	15:48	8,11	1,02	90,37	0,50	1,74
Dinamički (0%)	15:30	8,25	1,59	89,46	0,70	1,77

Ispitni slučaj promjenjivog raspoloženja sastoji se od tri galaksije koje sadrže svaka 150.000 zvijezda. Zbog velikih početnih brzina galaksija doći će do sudara što će uzrokovati velike promjene u raspodjeli tijela u prostoru. S obzirom na to da se raspodjela tijela mijenja često i naglo, tako se mijenjaju i vremena potrebna za obradu pojedinih tijela. Statička raspodjela postiže najmanje ubrzanje jer pretpostavlja da sva tijela zahtijevaju jednako vremena za obradu što ovdje često nije slučaj. Korištenje dinamičke raspodjele i manje zrnatosti daje veću fleksibilnost pri raspodjeli opterećenja i postiže bolje rezultate.

8.4 Simetrično opterećenje četiri jezgre

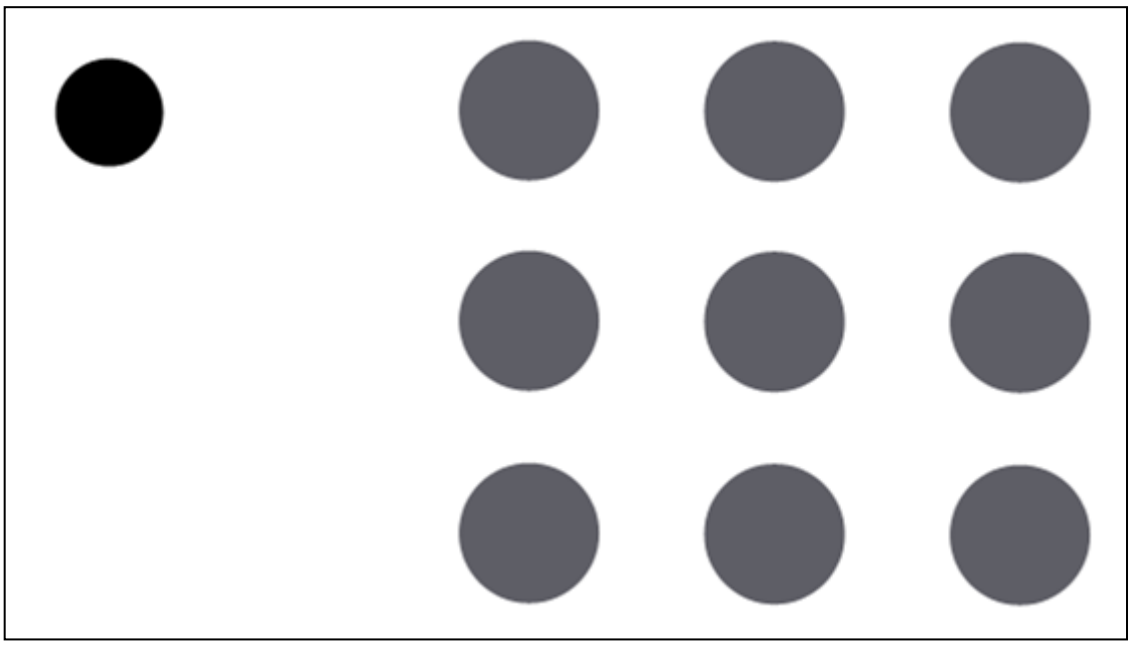


Slika 8.4: Ispitni slučaj simetričnog opterećenja četiri jezgre

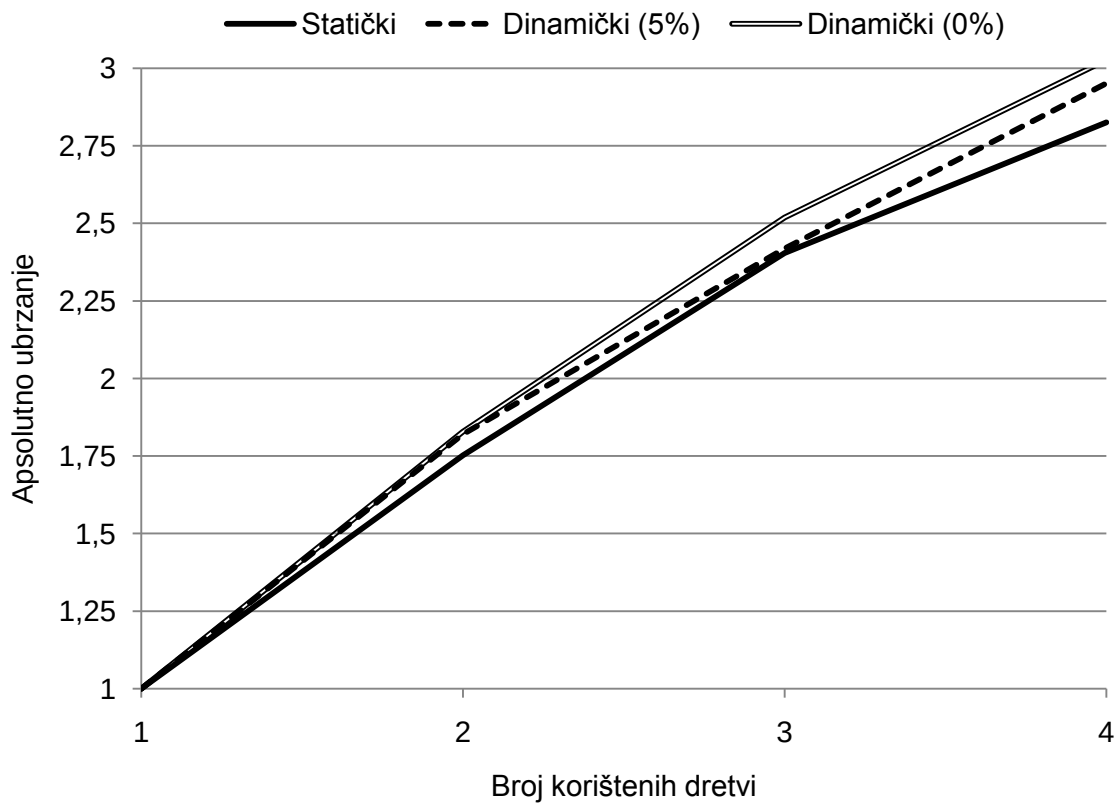


Slika 8.5: Graf apsolutnog ubrzanja za slučaj simetričnog opterećenja četiri jezgre

8.5 Asimetrično opterećenje četiri jezgre

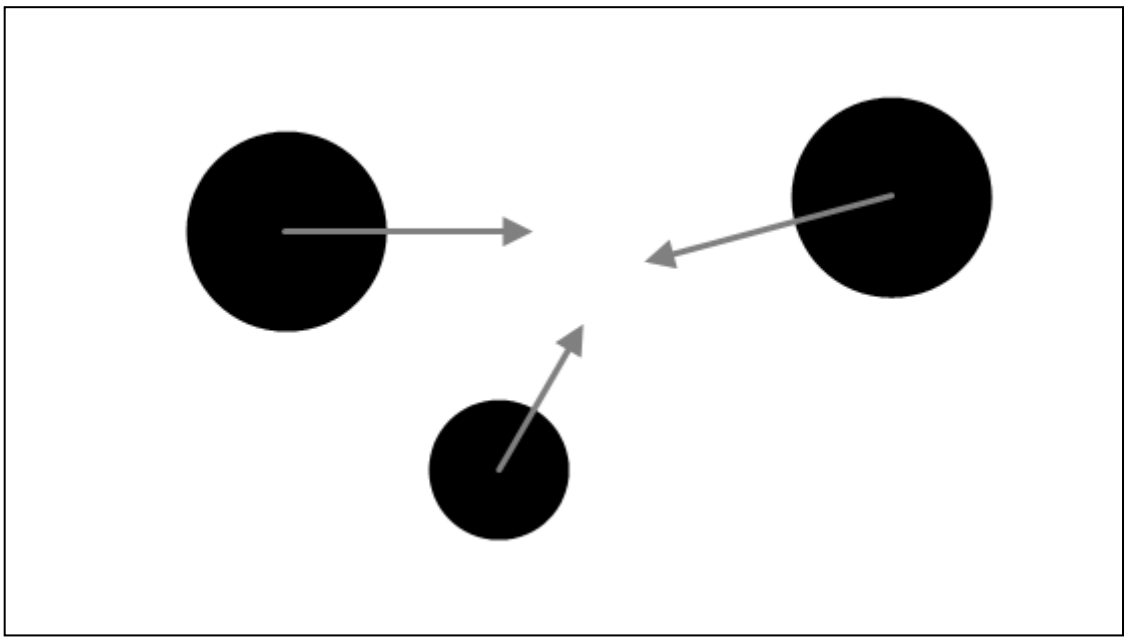


Slika 8.6: Ispitni slučaj asimetričnog opterećenja četiri jezgre

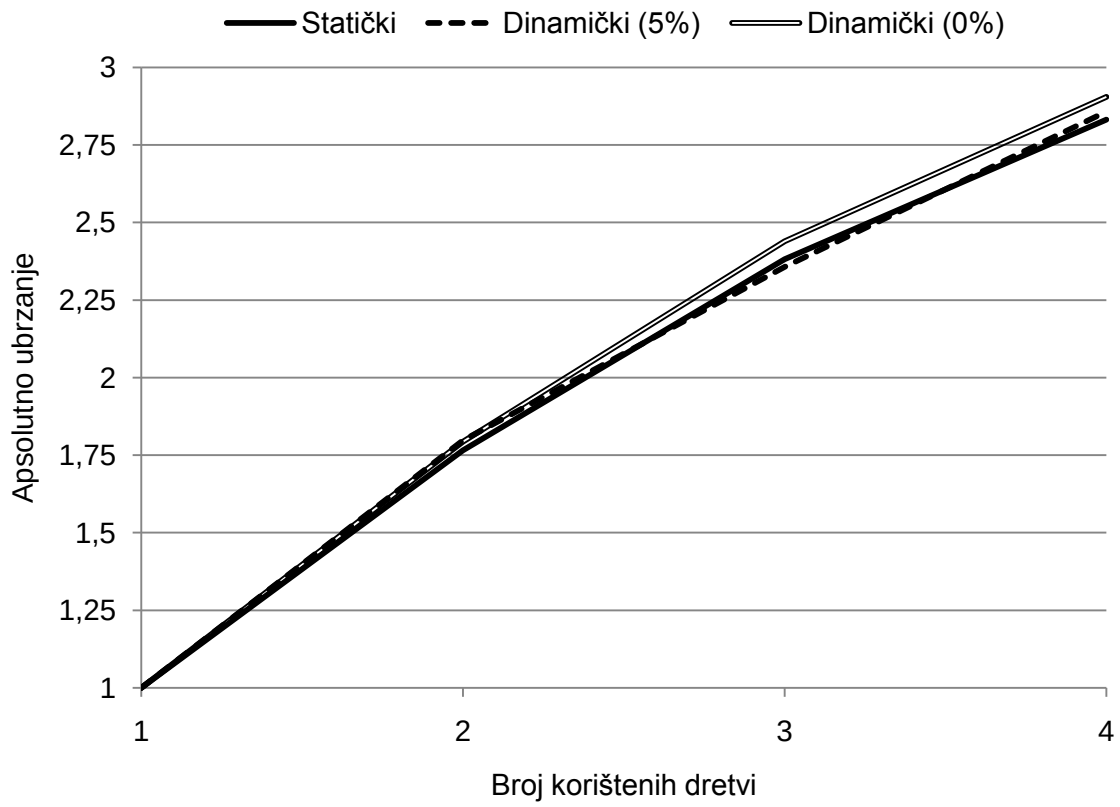


Slika 8.7: Graf apsolutnog ubrzanja za slučaj asimetričnog opterećenja četiri jezgre

8.6 Promjenjivo opterećenje četiri jezgre



Slika 8.8: Ispitni slučaj promjenjivog opterećenja četiri jezgre



Slika 8.9: Graf apsolutnog ubrzanja za slučaj promjenjivog opterećenja četiri jezgre

Zaključak

U ovom radu prikazane su metode i algoritmi paralelne simulacije gibanja nebeskih tijela. Objašnjena je implementacija Eulerove metode numeričke integracije diferencijalnih jednadžbi, metoda dekompozicije prostora pomoću strukture *quadtree* stabla, računanje utjecaja gravitacijskih sila primjenom algoritma Barnes-Hut te dva načina raspodjele podataka između više dretvi.

Razvijeno je programsko rješenje u jeziku Java koje omogućuje izvođenje simulacije korištenjem slijednog i paralelnog algoritma. Korištenjem razvijenog programa provedeno je nekoliko testova različitih sustava kako bi se ustanovio utjecaj načina raspodjele i veličine zrnatosti na vrijeme simulacije.

Pokazano je da ubrzanje dobiveno paralelnim računanjem uvelike ovisi o izboru načina raspodjela podataka kao i o ponašanju samog sustava koji se simulira. Rezultati su pokazali da je najbolje rješenje dinamička raspodjela podataka s malom zrnatošću jer se može najbolje prilagoditi dinamičnom sustavu nebeskih tijela. Veće granulacije smanjuju tu prilagodljivost, što za posljedicu ima neravnomjerno opterećenje dretvi i manje ubrzanje.

Paralelni algoritmi postižu poprilično ubrzanje u odnosu na slijedni algoritam, čime se značajno smanjuje vrijeme potrebno za izvođenje simulacije. S obzirom na to da moderni procesori često imaju više od jedne jezgre, paralelno računanje predstavlja dobar, iako složen, način ubrzanja izvođenja programa.

Literatura

[1] AARSETH, S.J. *Gravitational N-body Simulations: Tools and Algorithms*, New York: Cambridge University Press, 2003.

[2] BEUTLER, G. *Methods of Celestial Mechanics*, Berlin: Springer, 2005.

[3] DRMAČ, Z. et al. *Numerička analiza*, Zagreb: Sveučilište u Zagrebu, 2003.

[4] BERTSCHINGER, E. GELB, J.M. *N-body simulations*, Computers in Physics, 5, 2 (1991), 164–175.

[5] GREENGARD, L. ROKHLIN, V. *A fast algorithm for particle simulations*, Journal of Computational Physics, 73, 2 (1987), 325–348.

[6] HERNQUIST, L. OSTRIKER, J.P. *A self-consistent field method for galactic dynamics*, The Astrophysical Journal, 386, 1 (1992), 375–397.

[7] DUBINSKI, J. *A parallel tree code*, New Astronomy, 1, 2 (1996), 133–147.

[8] WARREN, S. *Astrophysical N-body simulations using hierarchical tree data structures*, Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing, Minneapolis (1992), 570–576.

[9] SALMON, J.K. WARREN, M.S. *Skeletons from the treecode closet*, Journal of Computational Physics, 111, 1 (1994), 136–155.

[10] MPI FORUM, *MPI Documents*, http://www.mpi-forum.org/docs, 22. 9. 2009.

[11] OPENMP ARCHITECTURE REVIEW BOARD (ARB), *OpenMP Specifications*, http://openmp.org/wp/openmp-specifications, 13. 1. 2009.

Sažetak

Paralelna simulacija gibanja nebeskih tijela

U ovom radu opisana je implementacija paralelne simulacije gibanja nebeskih tijela. Za dekompoziciju dvodimenzionalnog prostora korišten je algoritam gradnje *quadtree* stabla. Algoritam Barnes-Hut implementiran je za učinkovito računanje gravitacijskih sila koje utječu na gibanja nebeskih tijela, dok je za integraciju tih sila korištena Eulerova metoda za obične diferencijalne jednadžbe. Paralelno računanje ostvareno je pomoću sučelja za višedretvenost programskog jezika Java. Za koordinaciju rada dretvi korišteni su načini sinkronizacije semaforom, ogradom i nedjeljivim operacijama. Raspodjela podataka između dretvi ostvarena je na dva načina – jednostavnijom statičkom i fleksibilnijom dinamičkom raspodjelom uz podesivu zrnatost. Ispitan je utjecaj načina raspodjele podataka i veličine zrnatosti na ubrzanje dobiveno paralelnim računanjem.

Ključne riječi

paralelizacija, nebeska tijela, numerička integracija, dekompozicija prostora, quadtree, Barnes-Hut, višedretvenost, sinkronizacija, semafor, ograda, nedjeljive operacije

Summary

Parallel simulation of motion of celestial bodies

Implementation of parallel simulation of motion of celestial bodies is described in this thesis. Quadtree construction algorithm was used for decomposition of two-dimensional space. Barnes-Hut algorithm was used for efficient calculation of gravitational forces that affect motion of celestial bodies and Euler method for numerical integration of ordinary differential equations was used to integrate those forces. Parallel computing was achieved with Java multithreading programming interface. Thread synchronization with semaphore, barrier and atomic operations was used to coordinate multiple threads. Two methods for data distribution were implemented – simple static and more flexible dynamic distribution with configurable granularity. Influence of data distribution method and size of granularity on speedup archived with parallel computing was tested.

Keywords

parallelization, celestial bodies, numeric integration, decomposition of space, quadtree, Barnes-Hut, multithreading, synchronization, semaphore, barrier, atomic operations

Dodatak

Korištenje programskog rješenja

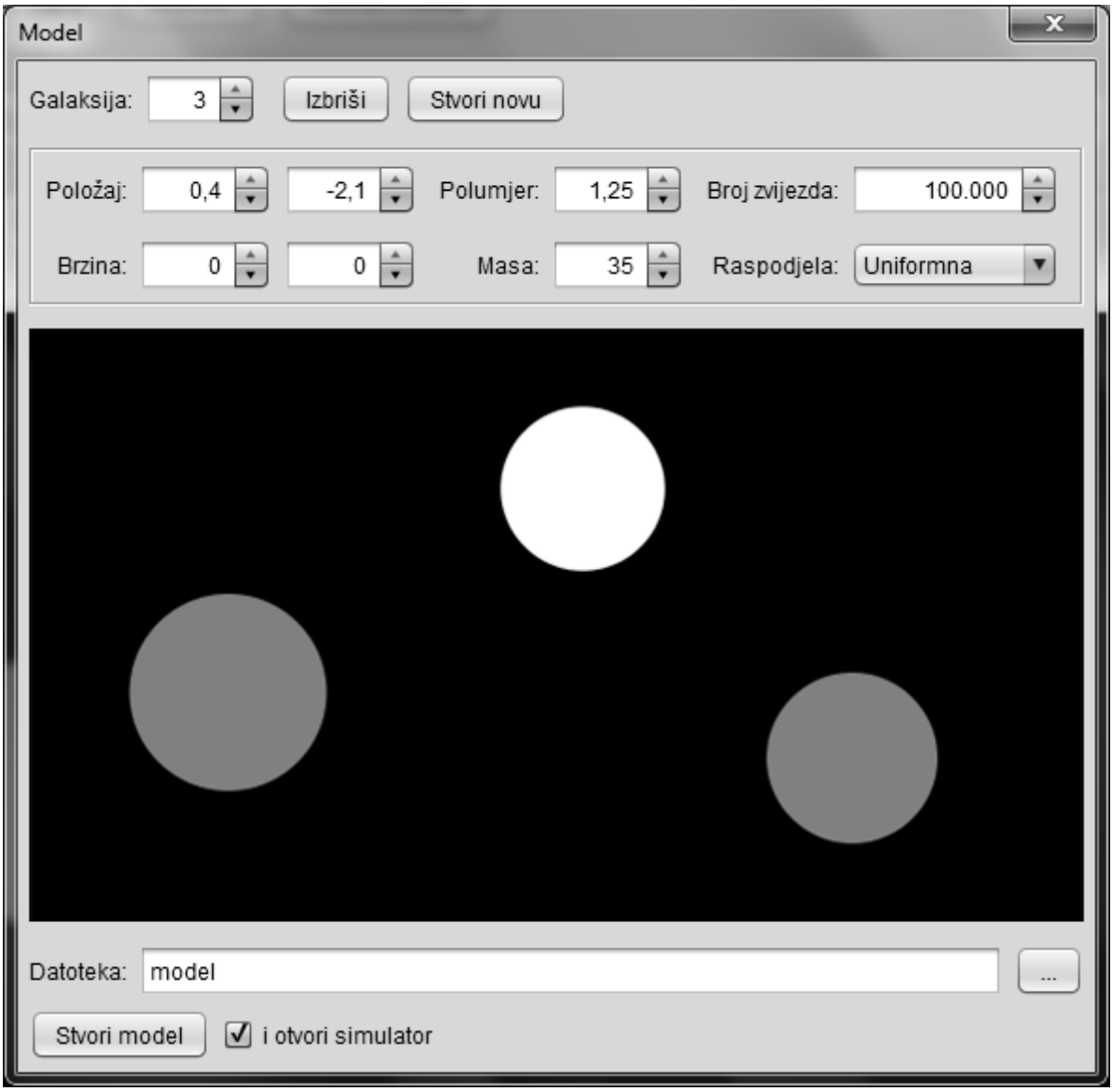
S obzirom na to da je program napisan korištenjem programskog jezika Java za njegovo izvođenje potrebno je na računalu imati instaliran *Java Runtime Environment* (JRE). JRE je moguće dohvatiti sa službenih stranica programskog jezika Java, a preporučljivo je koristiti zadnju verziju.

Grafičko sučelje programa prikazati će se nakon pokretanja:

Slika A-1: Početni ekran programa

Slika A-1 prikazuje početni ekran koji omogućava biranje između tri odvojena dijela programa:

1. Stvaranje modela simulacije tj. sustava galaksija nebeskih tijela kojima je moguće odrediti broj tijela od kojih se sastoje, početne položaje i brzine.
2. Simulacija stvorenog modela korištenjem slijednog ili paralelnog algoritma uz mogućnost određivanja parametara simulacije i snimanja video zapisa.
3. Pregled video snimke simulacije.

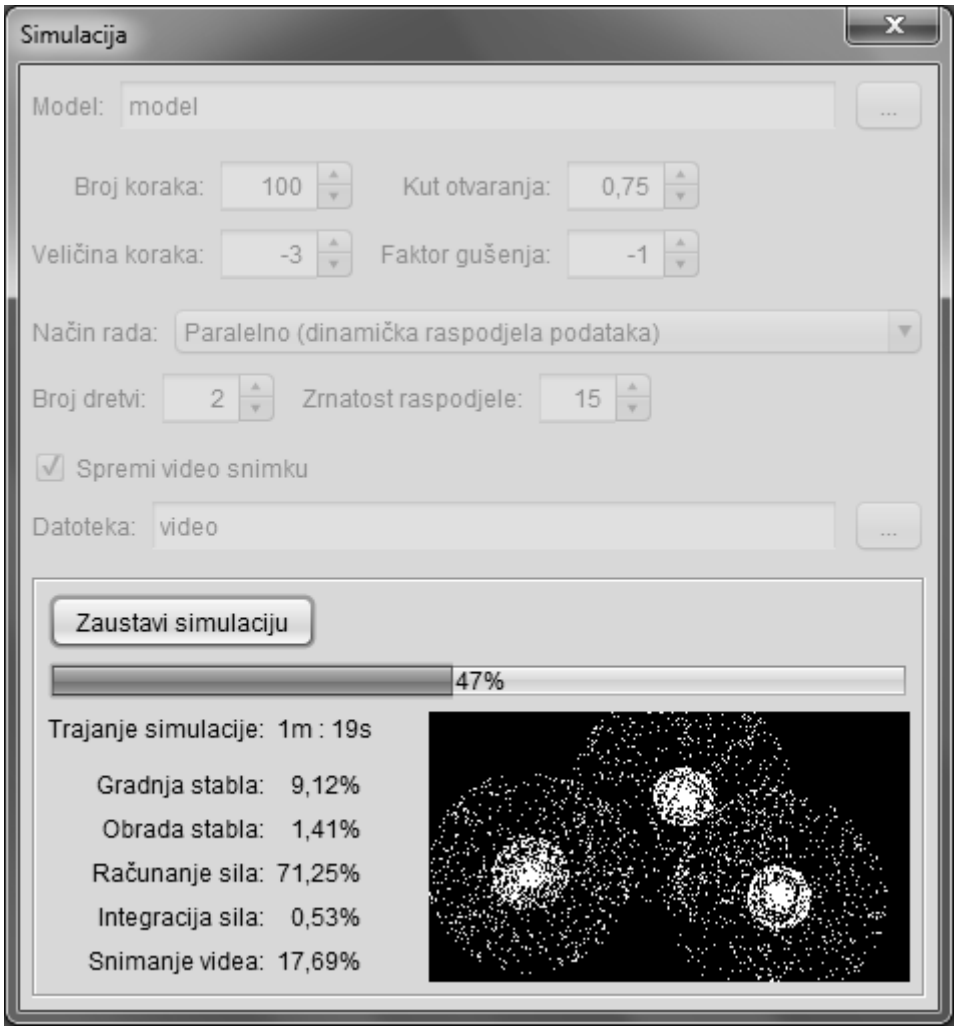


Slika A-2: Stvaranje modela simulacije

Slika A-2 prikazuje ekran modela. Moguće je stvarati nove, brisati stare i uređivati parametre postojećih galaksija. Za svaku galaksiju moguće je odrediti:

- položaj centra galaksije po x i y koordinati,
- početnu brzinu po x i y koordinati,
- polumjer,
- ukupnu masu svih zvijezda galaksije,
- broj zvijezda i
- raspodjelu zvijezda.

Moguće raspodjele su uniformna koja će uzrokovati jednolikom raspodjelom zvijezda i normalna koja će većinu zvijezda smjestiti bliže centru galaksije.



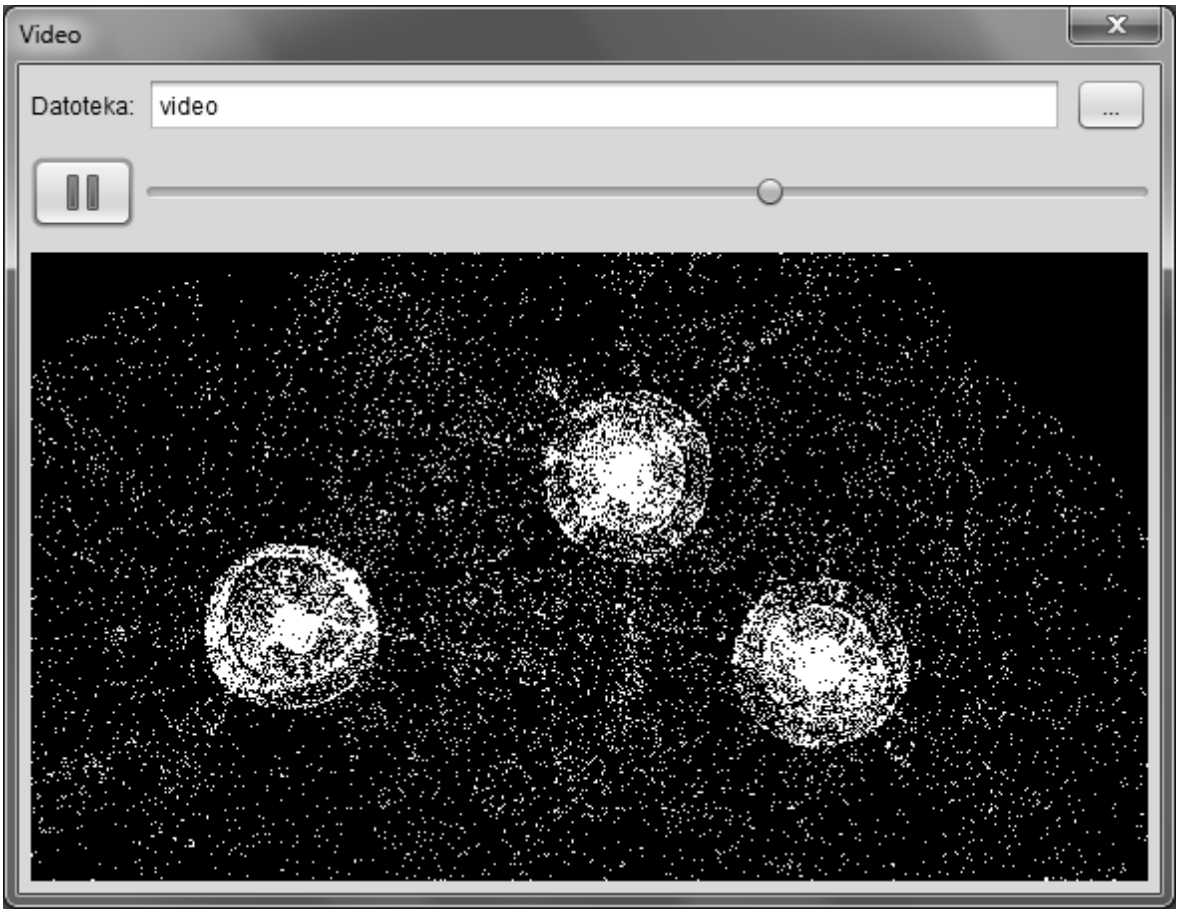
Slika A-3: Izvođenje simulacije

Slika A-3 prikazuje ekran simulacije. Za simulaciju je potrebno izabrati prethodno stvoren model te parametre simulacije:

- broj koraka,
- veličinu koraka zadanu kao potencija broja 10,
- kut otvaranja i
- faktor gušenja zadan kao potencija broja 10.

Od načina simulacije moguće je izabrati slijedni i paralelni algoritam sa statičkom ili dinamičkom raspodjelom podataka. U slučaju paralelne simulacije potrebno je izabrati broj dretvi koje će se koristiti, a u slučaju dinamičke raspodjele i zrnatost raspodjele kao postotak ukupnog broja tijela.

Tijekom simulacije prikazuje se postotak simuliranih koraka, trajanje simulacije, postotci vremena potrošeni na pojedine radnje te prikaz trenutnog stanja sustava.



Slika A-4: Pregled video snimke

Slika A-4 prikazuje ekran video snimke. Nakon izbora zapisa video snimke istu je moguće pregledati. Osim reprodukcije, video zapis moguće je privremeno zaustavi te se pozicionirati na proizvoljan trenutak unutar video snimke.