CHALMERS | GÖTEBORGS UNIVERSITET

Partikelformulering av Fluider

Flödessimulering utifrån en partikelbaserad beräkningsalgoritm

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Christian Gulliksson
Daniel Lindblad
Jian Tan

Institutionen för matematiska vetenskaper
Chalmers tekniska högskola
Göteborgs universitet
Göteborg 2012

Partikelformulering av Fluider

Flödessimulering utifrån en partikelbaserad beräkningsalgoritm

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk fysik vid Chalmers

Christian Gulliksson Jian Tan

Kandidatarbete i matematik inom civilingenjörsprogrammet Kemiteknik med Fysik vid Chalmers

Daniel Lindblad

Handledare:	Nils Svanstedt	Institutionen för Matematiska Vetenskaper, Chalmers
	Mohammad Asadzadeh	Institutionen för Matematiska Vetenskaper, Chalmers
	Sverker Edvardsson	Mittuniversitetet
Examinator:	Carl-Henrik Fant	

Institutionen för matematiska vetenskaper
Chalmers tekniska högskola
Göteborgs universitet
Göteborg 2012

Sammanfattning

I följande arbete undersöks en partikelbaserad simuleringsmetod som kallas Fluidpartikelmetoden, FPM. FPM är designad för att simulera komplexa fluiders beteenden på mesoskalan, vilken sammanlänkar kontinuumskalan med den molekylära skalan. Metoden utvecklades av Pep Español i slutet av 1990-talet och är en vidareutveckling och generallisering av "Dissipative Particle Dynamics" som togs fram av Hoogerbrugge och Koelman under tidigt 1990-tal. Fluiden betraktas som sammansatt av kluster av molekyler som kallas partiklar [1] [4]. Dessa partiklar interagerar med mjuka krafter med begränsad utsträckning.

Denna formulering gör det möjligt att simulera större system under längre tid än med modeller som bygger på modellering av enskiljda molekyler. Det är även möjligt att modellera fall som kontinuummodeller inte klarar av [2], [13].

Syftet med arbetet är att undersöka i vilka fall FPM är tillämpbar samt metodens för- och nackdelar. Målet är att utveckla ett programskal och ta fram den teori som krävs för att senare kunna simulera fluider med hjälp av FPM.

Den relevanta teorin som krävs för att tillämpa modellen i simuleringar presenteras. Detta innefattar de krafter som uppkommer mellan partiklarna och även teorin för att kunna kallibrera dessa utifrån makroskopiska egenskaper.

Ett program för tillämpning av FPM i två dimensioner redovisas. Denna implementering använder en indelning av beräkningsdomänen i ett rutnät av boxar. Boxindelningen är nödvändig för att begränsa mängden beräkningar och underlättar parallelliseringen.

Simuleringsresultaten visar att modellen återskapar samma typ av flöden som Navier-Stokes ekvationer i enklare fall. Indelningen av beräkningsdomänen i boxar tillsammans med parallellisering ger en avsevärd effektivisering av programmet.

Modellen har goda förutsättningar att ge goda simuleringsresultat men detta kräver kallibrering av modellens parametrar vilket visat sig mycket svårt. Utan effektivisering av algoritmen visar det sig att modellen har begränsad tillämpbarhet.

Abstract

In this project a particle based simulation-method called the Fluid Particle Method, FPM was examined. FPM is designed to simulate complex fluids and their behavior in the mesoscale which links the continuum scale with the molecular scale. The method was introduced by Pep Español in the late 1990s and is a continuation and generalization of the Dissipative Particle Method developed by Hoogenbrugge and Koelman during the early 1990s. The fluid is modeled as clusters of molecules called particles [1] [4]. These particles interact with soft forces with limited range.

This model makes it possible to simulate bigger systems during longer times than models based on interactions between single molecules. It is also possible to model cases where the continuum falters [2], [13].

The purpose of this project is to examine where FPM is applicable and also the methods advantages and disadvantages. The goal is to develop a program and find the theory required to simulate fluids with FPM.

The relevant theory required to implement the model is presented. This includes the forces between particles and the theory to calibrate these from macroscopic properties.

A implementation of FPM in two dimensions is also presented decomposing the simulation-domain into boxes. This domain decomposition is necessary in order to limit the amount of calculations and simplifies the parallelization of the program.

Simple simulations show that the model creates the same type of flows as Navier-Stokes. Results also show that the partition of the domain into boxes, together with the parallelization, greatly improves program performance.

The model has potential to give good simulation results however further calibration of the parameters is needed. It is also clear that the model is limited if proper optimization of the algorithm is not made.

Innehåll

Förord

Följande kandidatarbete är gjort av tre personer som studerar tredje året vid en civilingenjörsutbildning på Chalmes Tekniska Högskola. Arbetet genomfördes våren 2012 vid institutionen för Matematiska Vetenskaper.

Varje gruppmedlem har kontinuerligt fört en tidslogg där denne dokumenterat vad denne har gjort och hur lång tid detta har tagit. Tidsloggen har skrivits för varje tillfälle det jobbats med arbetet. Det har även förts en gemensam dagbok där varje veckas arbete och framsteg har sammanfattats.

För att få en så effektiv arbetsgång som möjligt har ansvaret för olika delar av arbetet delats upp mellan gruppens medlemmar. Alla gruppmedlemmar har dock varit engagerade och involverade i alla delar av projektet och bidragit med hjälp. En utförligare utläggning av de olika medlemmarnas ansvarsområden följer nedan.

Ansvarsområden

Planering

Arbetet var från början mycket öppet och det var därför svårt att göra upp en exakt plan på vad som skulle åstakommas. Ingen visste vad som var rimligt att uppnå och vilka problem som kunde dyka upp. Därför så planerades arbetet kontinuerligt och anpassades utefter hur det gick. Genom att föra en kontinuerlig diskussion inom gruppen sattes rimliga delmål upp för när olika saker skulle vara gjorda och detta fungerade mycket bra. Daniel tog ansvaret för att planeringen genomfördes.

Informationsinhämtning/inläsningsdel

Informationsinhämtningen var alla i gruppen delaktiga i. Beroende på ansvarsområde tog olika personer ansvar för informationshämtning inom olika områden. Daniel hade främsta ansvaret för inläsning och informationshämtning gällande den fysikaliska modellen, numerisk lösningsmetod och kalibrering av modellen. Christian tog ansvaret för att läsa in sig mer på C programmering och teorin bakom hur programmet skulle fungera. Handledarna har även bidragit med inhämtning och tolkning av information.

Metoder - Val/Utveckling

Utveckligen av ett program samt valet av en metod för att simulera fluider var det stora syftet med detta arbetet. Från början var det endast sagt att en mesh-fri, partikelbaserad flödesalgoritm skulle användas. Hela gruppen ansvarade för att välja ut den partikelmetod som sedan användes. Utveckligen av programmet gjordes gemensamt men huvudansvaret tog Christian ansvar för.

Genomförande

Hela gruppen hjälptes åt med alla delar av projektet och att dessa blev genomförda. Ansvaret för att fysiken undersöktes och att relevant teori togs fram för att möjligöra implementeringen av den hade Daniel och Tan. Christian med stöd av Tan hade huvudansvaret för att programmet skapades och fungerade samt att simuleringar genomfördes. Ansvaret för rapportens skrivande tog alla hand om. Daniel tog hand om delarna om fysiken och den numeriska lösaren, Christian och Tan om programdelen och Tan för alla figurer och illustrationer. Daniel tog även ansvar för att gruppen deltog i alla handledningstillfällen, generella kompetenser samt höll koll på att saker genomfördes.

Bidrag till problemlösning, syntes och analys

Problemlösning

En rad problem har dykt upp under projektets gång. Gruppen har gemensamt hjälpts åt att lösa de problem som uppkommit och alla har kommit med konstruktiva idéer för att komma fram till lösningar. Problemen som uppkom kring den fysikaliska modellen, hur denna skulle tolkas och implementeras samt ställas in, bidrog främst Daniel till att lösa. Problem med programeringen uppstod även. Dessa problem löstes gemensamt, även om Christian med hjälp av Tan bidrog mest till denna del.

Kreativitet, idérikedom

Mycket kreativitet och nya idéer krävdes för att utveckla programmet eftersom ingen klar mall för hur det skulle se ut fanns. Mycket av de nya idéerna diskuterades gemensamt fram, ibland även i samråd med handledarna. Allt från övergripande idéer till programmeringsdetaljer har diskuterats och gemensamt utveckalts. Tack vare att Christan med hjälp av Tan hade huvudansvaret för programmeringen tog de störst ansvar för att komma med idéer på hur många programmeringsdetaljer skulle genomföras.

Skapande av en modell

Arbetet har bestått av tvåhuvudsakliga modeller. Det första som togs fram var en modell för att beskriva fluiden. Hela gruppen arbetade gemensamt den första tiden med att undersöka olika fysikaliska modeller och tog sedan gemensamt beslutet om att välja Fluidpartikelmetoden. Daniel med hjälp av Tan tog sedan ansvaret för att ta fram den teori som krävdes för att få en fullständig beskrivning av modellen och kunna tillämpa den. Den andra modellen som krävdes var en modell för hur programmet skulle se ut och i vilket programmeringsspråk det skulle skrivas. Främst Christian mha Tan tog ansvaret för att undersöka detta och tog sedan beslutet att C var den bästa programmeringsmiljön för vårt syfte. Modellen för programmet har sedan kontinuerligt och gemensamt tagits fram, Christian och Tan har här tagit ansvar för att modellen implementerats och fungerat. Implementeringen av den fysikaliska modellen i programmet genomfördes sedan gemensamt.

Analys av Projektrelaterat material

I början av projektet så hjälptes gruppen åt att ta fram och läsa artiklar som behandlade partikelmetoder. När ett gemensamt beslut om partikelmetod var taget blev arbetet mer individuellt. Daniel med hjälp av Tan fokuserade på material som rörde teorin bakom metoden. Christian med hjälp av Tan jobbade med programmeringen och fick läsa på väldigt mycket om hur syntaxen i C fungerar samt hur effektiv programmvara skrevs. När programmet var skrivet och den fysikaliska modellen klarställd hjälptes hela gruppen åt att analysra programmet och leta fel och möjligheter till förbättringar. Christian tog ansvar för att analysera prestandan i programmet och alla hjälptes på slutet åt att analysera de simuleringsresultat som programmet gav.

Huvudansvarig författare för olika avsnitt

Sammanfattning/Abstract	Christian, Daniel, Tan
Förord	Daniel
1. Inledning	Christian, Daniel, Tan
2. Teoridel	Daniel
2.1 Fluid Partikel Metoden, FPM	Daniel
2.2 Inställning av modell	Daniel
3. Implementering av Fluidpartikel-metoden	Christian, Daniel, Tan
3.1 Initialisering	Christian, Tan
3.2 Indexering och sortering	Christian, Tan
3.3 Beräkning av krafter	Christian, Tan
3.4 Tidsutveckling av systemet	Daniel
3.5 Paralellisering	Christian, Tan
3.6 Metod, Genomförande	Christian, Tan
4 Resultat	Christian
4.1 Prestanda	Christian
4.1 Simuleringar	Christian
5. Diskussion	Christian, Daniel
5.1 Teori	Daniel
5.2 Programmet	Christian
6. Slutsatser	Christian
Källförteckning	Daniel
Bilagor	Christian

Det redaktionella ansvaret för att rapporten strukturerats och att alla bitar kommit med har Daniel tagit hand om. Christian tog ansvaret för att rapporten kompilerades som den skulle.

Tack till

Vi vill först och främst tacka vår handledare Nils E M Svanstedt för hans engagemang, intresse och hjälp med vårt arbete. Nils gick ur tiden 28 April 2012, två veckor innan detta kandidatarbete skulle vara klart. Vi är mycket tacksamma för allt Nils hjälpt oss med och tycker det är mycket tråkigt att vi inte kommer få dela slutresultatet med honom. Vi vill även tacka Mohammad Asadzadeh för att han ställt upp som handledare efter Nils bortgång.

Ett stort tack till Mårten Gulliksson vid Mittuniversitetet för hans hjälp under arbetets gång. Mårten har bidragit med många goda råd både rörande tillvägagångs-sätt och rapport skrivande.

Vi vill även ge ett stort tack till Jenny Persson för att hon bidragit med sin språkliga kunskap.

Till sist vill vi även tacka Thomas Ericsson vid Matematiska Vetenskaper. Thomas har vairt till stor hjälp med programmeringen i C samt testkörning av programmet på Chalmers beräkningsdatorer för att testa prestandan.

1 Inledning

Beksrivningar av flödens beteenden är en viktig del inom många teknikområden och det forskas flitigt på området både i industrin och inom den akademiska världen. Det vanligaste angreppssättet att matematiskt beskriva och modellera flöden är genom Navier-Stokes ekvationer. Navier-Stokes beskrivningen bygger på att fluiden betraktas som ett kontinuum och det finns fall då denna beskrivning riskerar att inte fungera, speciellt för flöden på mikro- och nanometer-skalan [13].

För de flesta fall av flöden existerar inga analytiska lösningar till Navier-Stokes ekvationer och metoder har därför utvecklats för att lösa dessa numeriskt. De numeriska lösningsmetoder som finns kan i vissa fall även vara svåra att tillämpa. Speciellt svåra fall är kolloid-suspensioner med fasta rörliga partiklar och simuleringar i krångliga geometrier. Speciella metoder har därför utvecklats för att komma runt denna problematik [13].

I slutet av 1950-talet beskrevs för första gången en fluid som enskilda molekyler som interagerar [14]. Metoden kallas *Molecular Dynamics*(MD) och kan i teorin beskriva vilken fluid som helst. Denna metod är endast tillämpbar på väldigt små system eftersom mängden beräkningskraft för att simulera större system blir för stor. Huggebrugge och Koelman utvecklade i början av 1990-talet metoden *Dissipative Particle Dynamics* (DPD) vilken beskriver fluiden som en samling kluster av molekyler även kallat partiklar. Simuleringar på större skala under längre tid blev nu möjliga då mängden partiklar att simulera minskade samtidigt som krafterna som uppkommer mellan dessa kluster var mjukare än i MD. Pep Español presenterade 1997 en generaliserad vidareutvecklig av Huggebrugge och Koelman's modell som fick namnet *Fluid Particle Model* (FPM). DPD och FPM är designade för simuleringar på mesoskalan som bryggar gapet mellan kontinuumskalan och den molekylära skalan [1] [4].

På mesoskalan finns flera modeller utvecklade, två av dessa är *Lattice Gas Automata (LGA)* och *Lattice Boltzmann Automata (LBA)*. Båda metoderna bygger på att flödet beskrivs av ett antal diskreta punkter i ett gitter. Gittret har visat sig ge problem med randvärden och Galileisk invarians. Formuleringen av fluiden i termer av fria partiklar är en av styrkorna med DPD och FPM. Hantering av ränder, införandet av större partiklar i form av polymerer eller kolloider samt implementeringen av modellen blir avsevärt lättare med FPM än i andra typer av modeller [4].

Españols partikelmetod har i dagsläget inte tillämpats i någon större utsträckning och det är oklart vilka användningsområden modellen har. För att undersöka FPM metodens möjligheter och begränsningar krävs programvara för att utföra simuleringar.

1.1 Syfte

Arbetet syftar till att ge kunskap om Fluidpartikel-metodens styrkor och nackdelar vad gäller den bakomliggande teorin men även vad gäller metodens implementering.

Målet är att utveckla en programvara som kan användas för att simulera fluider med FPM.

1.2 Avgränsningar

Projektet är begränsat till utvecklingen av programvara för tillämpning av FPM. En teoretisk bakgrund samt en redogörelse för hur parametrarna kan bestämmas presenteras i syfte att göra detta möjligt. Parameterarna har endast uppskattats och testsimuleringar har gjorts.

1.3 Innehållet i rapporten

I avnitt 2.1 presenteras först den teori som Fluidpartikel-metoden bygger på. Efter detta presenteras i avsnitt 2.2 en metod för att kalibrera partikelmodeller utefter makroskopiska egenskaper. Programmet presenteras utförligt i avsnitt 3. Programmets funktion samt den numeriska lösare som använts presenteras.

I avsnitt 4 presenteras resultaten. Programmets prestanda och resultat från simuleringar av ett enkelt fall med vattenflöde mellan två plattor redovisas.

I avsnitt 5 diskuteras slutligen den fysikaliska modellen samt programmets struktur och funktion. Några sammanfattande slutsatser om modellen och programmet tillsammans med förslag på framtida arbeten med Fluidpartikel-metoden ges i avsnitt 6.

2 Teori

Teoridelen i denna rapport tar upp den nödvändiga teori som krävs för att implementera FPM. Detta innefattar en beskrivning av krafterna mellan partiklarna samt de skalära funktioner och parametrar som bestämmer dessa. En definition av partikelsystemet i två dimensioner samt en beskrivning av hur de kraftbestämmande parametrarna bestämms tas också upp.

2.1 Fluid Partikel Metoden, FPM

I Fluidpartikel-metoden beskrivs partiklarna som kluster eller droppar av molekyler. Det är dessa partiklar och inte de individuella molekylernas beteende som modellen beskriver.

I detta arbete kommer endast simuleringar i två dimensioner att utföras, vilket innebär att flödet antags vara konstant i den tredje dimensionen. I två dimensioner simuleras ett lager av partiklar likt ett stort antal kulor på ett golv.

Interaktionerna mellan partiklarna beskrivs av krafter vilket gör att Newtons mekanik kan användas för att beskriva tidsutvecklingen av systemet av partiklar och därmed också fluiden. På grund av detta är beskrivningen av fluiden helt oberoende av någon mesh som till exempel Finita Element Metoden baserade lösningar är. Partiklarnas läge, hastighet samt rotationshastighet bestämmer i FPM fluiden fullständigt.

2.1.1 Krafterna

FPM bygger på att partiklarna interagerar med fyra krafter. Krafterna uppkommer parvis mellan två partiklar i och j och berskriver de olika typer av interaktioner som uppkommer mellan de kluster av molekyler som partiklarna motsvarar [1].

Konservativ kraft Den första kraften är en konservativ och rent repusliv kraft. Den håller isär partiklarna och förebygger att de klumpar ihop sig. Kraften uppkommer på grund av att molekylerna som bygger upp klustrena repellerar varandra vilket gör att även klustrena gör det [10]. Denna kraft har formen

$$\vec{F}_{ij}^{K} = -\Lambda \cdot K(r_{ij}) \cdot \vec{e}_{ij}, \tag{1}$$

där Λ är en positiv skalfaktor som bestämmer storleken på kraften och $K(r)$ är en positiv skalär funktion som beskriver kraftens avståndsberoende [2]. Avståndet mellan partiklarna är $r_{ij} = |\vec{r}_{ij}|$, där $\vec{r}_{ij}$ är vektorn som går fran partikel i till j. Vektorn $\vec{e}_{ij} = \vec{r}_{ij}/r_{ij}$ är enhetsvektorn som pekar från partikel i till j och notationen ij betyder att kraften verkar på partikel i och orsakas av partikel j. Minustecknet gör att kraften beskriver repulsion [2].

Translationsberoende kraft Nästa kraft beror både på partiklarnas lägen och hastigheter. Denna beskriver friktionen som uppkommer då partiklar rör sig i förhållande till varandra. Kraften definieras enligt

$$\vec{F}_{ij}^{T} = -\gamma m \mathbf{M}_{ij}^{T} \cdot \vec{v}_{ij}, \tag{2}$$

där γ är en positiv skalfaktor och m är massan på en partikel [1]. Vektorn $\vec{v}_{ij} = \vec{v}_i - \vec{v}_j$ är den relativa hastigheten mellan partiklarna i och j. Har partiklarna samma hastighet blir den relativa hastigheten noll och ingen friktion uppkommer mellan partiklarna. Matrisen $\mathbf{M}_{ij}^{T}$ är definierad som

$$\mathbf{M}_{ij}^{T} = A(r_{ij})\mathbf{I} + B(r_{ij})\vec{e}_{ij}\vec{e}_{ij}, \tag{3}$$

där $A(r)$ och $B(r)$ är två positiva skalära funktioner som beskriver kraftens avståndsberoende. Matrisen $\mathbf{I}$ är en identitetsmatris av storlek $D \times D$, där D är systemets dimension. Den första termen i (2), med (3) insatt, (som innehåller $A(r)$) pekar i motsatt riktning till den relativa hastigheten och bromsar därför partiklarnas individuella rörelse. Den andra termen (som innehåller $B(r)$) pekar från partikel j till partikel i och syftar till att bromsa hastigheten partiklarna närmar sig varandra med. Båda delarna motsvarar därför friktion, den första när de rör sig i förhållande till varandra och den andra på grund av att de närmar sig varandra [1].

Rotationsberoende kraft Den tredje kraften uppkommer då partiklarna roterar i förhållande till varandra. Denna kraft beskriver också friktion och är definierad som

$$\vec{F}_{ij}^{R} = -\gamma m \mathbf{M}_{ij}^{R} \cdot (\frac{\vec{r}_{ij}}{2} \times [\vec{\omega}_i + \vec{\omega}_j]), \tag{4}$$

där γ och m är samma som i (2) och vektorn $\vec{\omega}$ är vinkelhastigheten hos ett kluster [1]. Matrisen $\mathbf{M}_{ij}^{R}$ är i sin tur definierad som

$$\mathbf{M}_{ij}^{R} = C(r_{ij})\mathbf{I} + D(r_{ij})\vec{e}_{ij}\vec{e}_{ij}, \tag{5}$$

där $C(r)$ och $D(r)$ är två positiva skalära funktioner som beskriver kraftens avståndsberoende. Den sista termen i (4), med (5) insatt, (som innehåller $D(r)$) är identiskt noll tack vare att skalärprodukten mellan vektorn $\vec{r}_{ij}/2 \times [\vec{\omega}_i + \vec{\omega}_j]$ och $\vec{e}_{ij}$ är noll. Anledningen till att $\mathbf{M}_{ij}^{R}$ är definierad med två termer är för att få en analogi med $\mathbf{M}_{ij}^{T}$ [1].

Den rotationsberoende kraften är analog med den translationsberoende kraften. I (2) är kraften proportionell mot den relativa hastigheten mellan partiklarna. I (4) är kraften istället proportionell mot kryssprodukten mellan halva vektorn som pekar från partikel i till j och summan av deras vinkelhastigheter. Om partiklarna tolkas som runda kluster av molekyler vilka precis kommer i kontakt med varandra är avståndet från partikel $i's$ centrum till kontaktytan $r_{ij}/2$. Den relativa hastigheten mellan ytorna på klustrena på grund av rotation är då precis kryssprodukten mellan $\vec{r}_{ij}/2$ och summan av de två partiklarnas vinkelhastigheter. Detta gör att kraften i (4) helt analogt med (2) är en friktionskraft som beror av den relativa hastighet som kommer av att partiklarna roterar [1].

Både $\vec{F}^{T}$ och $\vec{F}^{R}$ är icke centrala krafter som påverkar partiklarna med ett vridmoment. Detta gör att partiklarnas vinkelhastighet förändras i tiden [1].

Stokastisk kraft Den sista kraften tar hänsyn till de förlorade antalet frihetsgraderna som är resultatet av att behandla kluster av molekyler istället för varje individuell molekyl. Då klustrena interagerar via de tre första krafterna skapas en inre oreda i dem. Eftersom inte varje enskild molekyls påverkan tas hänsyn till i FPM modelleras summan av alla molekylers påverkan i klustret av en slumpmässig, eller stokastisk, kraft som uppkommer som ett resultat av de tre första krafterna. Den stokastiska kraften är definierad som

$$\vec{F}_{ij}^{S}\Delta t = \sigma m(\tilde{A}(r_{ij})d\mathbf{W}\text{'}_{ij}^{S} + \tilde{B}(r_{ij})\frac{1}{D}tr[d\mathbf{W}_{ij}]\mathbf{I} + \tilde{C}(r_{ij})d\mathbf{W}_{ij}^{A}) \cdot \vec{e}_{ij}, \tag{6}$$

där σ är en positiv skalfaktor, m en partikels massa och Δt längden på ett iterativt tidssteg som görs i simuleringen [1]. Funktionerna $\tilde{A}(r)$, $\tilde{B}(r)$ och $\tilde{C}(r)$ är tre positiva skalära funktioner som beskriver kraftens avståndsberoende. Kraften beror även av de tre infinitisimala matriserna $d\mathbf{W}\text{'}_{ij}^{S}$, $d\mathbf{W}_{ij}^{A}$ samt $d\mathbf{W}_{ij}$, där de två första matriserna är skapade utifrån den tredje. Matrisen $d\mathbf{W}_{ij}$ är en $D \times D$ matris som består av D^2 normalfördelade värden med väntevärde noll och varians Δt. De infinitisimala matriserna är definierade enligt

$$d\mathbf{W}_{ij}^{S\mu\nu} = \frac{1}{2}[d\mathbf{W}_{ij}^{\mu\nu} + d\mathbf{W}_{ij}^{\nu\mu}], \tag{7}$$

$$d\mathbf{W}_{ij}^{A\mu\nu} = \frac{1}{2}[d\mathbf{W}_{ij}^{\mu\nu} - d\mathbf{W}_{ij}^{\nu\mu}], \tag{8}$$

$$d\mathbf{W}\text{'}_{ij}^{S} = d\mathbf{W}_{ij}^{S} - \frac{1}{D}tr[d\mathbf{W}_{ij}^{S}]\mathbf{I}. \tag{9}$$

Den första matrisen i (8) är symetrisk, den andra i (9) är antisymetrisk och den tredje i (9) är en spårlös, symetrisk matris [1]. Här betecknar μ och ν en rad resp kolonn i matrisen. Det måste även gälla att $d\mathbf{W}_{ij} = d\mathbf{W}_{ji}$. Detta villkoret gör att den stokastiska kraften likt de tre föregående krafterna i FPM uppfyller Newtons tredje lag, $\vec{F}_{ij} = -\vec{F}_{ji}$. Newtons tredje lag gör att partiklarnas rörelsemängd är bevarad [1].

Nästa steg i att beskriva den fysikaliska modelllen är att definiera de skalära funktioner som beskriver krafternas avståndsberoende, detta görs i nästa avsnitt.

2.1.2 Skalära funktioner

De skalära funktionerna beskriver krafternas avståndsberoende. I FPM finns inga exakta ramar för hur dessa skall se ut, men det finns två villkor som de bör uppfylla. Det första villkoret är att kraften inte skall ha oändlig räckvidd, det skall därför finnas ett avstånd sådant att om partiklarna befinner sig längre ifrån varandra än detta blir kraften noll. Detta kan formuleras som en cut-off radie, r_{cut}, sådan att $\{r_{ij} > r_{cut} \implies W(r_{ij}) = 0\}$ [2]. Funktionen $W(r)$ är en godtycklig skalär funktion. Det andra villkoret är att funktionerna skall uppfylla ett normeringsvillkor som kan formuleras

$$\int W(r)d\mathbf{r} = \frac{1}{n}. \tag{10}$$

Denna integral är tagen över en cirkel eller sfär, med radie r_{cut}, beroende på om modellen tillämpas i två eller tre dimensioner. Här är n partikeltätheten som är definierad som antalet partiklar per yt- eller volymsenhet. Normeringsvillkoret betyder att de skalära funktionernas utbredning skall vara proportionellt mot inversen av partikeltätheten, vilken i sin tur är proportionell mot storleken på partiklarna [4].

För att modellen skall bli termodynamiskt stabilt och uppfylla *Fluktuation Dissipations Teoremet* måste $A(r) = C(r)$ [1]. Sista termen i (4) är noll och därför kan $D(r)$ sättas till noll utan att ändra något. Det är även smidigt att sätta $B(r) = A(r)$ [2] [3].

Formen på de skalära funktionerna $K(r)$ och $A(r)$ väljs med ledning av [2] till

$$K(r) = \kappa \cdot (1 - \frac{r}{r_{cut}}) \tag{11}$$

respektive

$$A(r) = \alpha \cdot (1 - \frac{r}{r_{cut}})^2. \tag{12}$$

Konstanterna κ respektive α skall väljas så att (11) och (12) uppfyller (10). I två dimensioner blir dessa

$$\kappa = \frac{3}{n\pi r_{cut}^2} \tag{13}$$

och

$$\alpha = \frac{6}{n\pi r_{cut}^2}. \tag{14}$$

De skalära funktioner som ingår i (6) relaterar till de andra skalära funktionerna via två relationer [1]. Relationerna gör att modellen kan uppfylla *Fluktuations Dissipations Teoremet* och är

$$A(r) = \frac{1}{2}[\tilde{A}^2(r) + \tilde{C}^2(r)] \tag{15}$$

och

$$B(r) = \frac{1}{2}[\tilde{A}^2(r) - \tilde{C}^2(r)] + \frac{1}{D}[\tilde{B}^2(r) - \tilde{A}^2(r)]. \tag{16}$$

För att förenkla beräkningarna kan $\tilde{A}(r)$ väljas till noll [1]. Det gäller även att $A(r) = B(r) = C(r)$, vilket gör att (15) och (16) förenklas till

$$\tilde{B}(r) = \sqrt{2DA(r)} \tag{17}$$

och

$$\tilde{C}(r) = \sqrt{2A(r)}. \tag{18}$$

Nästa steg är att bestämma storleken på krafterna vilka ges av skalfaktorerna. För att bestämma dessa krävs att partikelsystemet är noga definierat.

2.1.3 Definition av partikelsystemet i 2D

Fluiden som FPM beskriver består av ett stort antal partiklar som alla bär på ett antal egenskaper såsom storlek och massa. Det är viktigt att kunna relatera dessa egenskaper till den verkliga fluid som skall simuleras för att vidare kunna beräkna storleken på de krafter som ingår.

Systemets totala massa måste vara konserverad, vilket kan uttryckas som

$$M = \sum_{i=1}^{N_{tot}} m_i, \tag{19}$$

där M är systemets totala massa, m_i betecknar en partikels massa och N_{tot} är totalt antal partiklar i systemet. Partiklarnas massa m_i beror på hur stora klustrena är samt fluidens densitet, ρ. Storleken av klustrena beror på hur tätt indelad fluiden är i kluster. Indelningens täthet beskrivs av δ, som definieras som medelavståndet mellan partiklarna i en dimension. Om fluiden befinner sig i en kubisk låda med sidan L, där N st partiklar får plats utmed en sida är $\delta = L/N$. Den endimensionella partikeltätheten, som beskriver antalet partiklar per längdenhet, definieras utifrån δ som

$$\eta = \frac{1}{\delta}. \tag{20}$$

Med hjälp av η kan partiklarnas massa beräknas. Låt åter fluiden befinna sig i en kubisk låda med sidan L och med N partiklar utmed en sida. Volymen blir då $V = L^3$ och totala antalet partiklar $N_{tot} = N^3$. Utifrån densiteten för fluiden blir den totala massan $M = \rho V$. Då varje partikel antas vara lika stor och densiteten antas konstant blir massan för en partikel

$$m_i = \frac{\rho V}{N^3} = \frac{\rho L^3}{N^3} = \frac{\rho}{\eta^3}. \tag{21}$$

Massan på en partikel är oberoende av om modellen tillämpas i två eller tre dimensioner, skillnaden är hur många dimensioner som partiklarna rör sig i.

Partikeltätheten, som i två dimensioner är definierad som antalet partiklar per ytenhet i två dimensioner, blir utifrån definitionen av N och L

$$n = \frac{N^2}{L^2} = \eta^2. \tag{22}$$

Partiklarna har även ett tröghetsmoment. I två dimensioner roterar de endast kring z axeln som är vinkelrät mot planet de rör sig i. Tröghetsmomentet för en partikel approximeras med det för ett klot, som runt z axeln är

$$I_{zz} = \frac{2}{5} m_i r_i^2, \tag{23}$$

där m_i är partikelns massa och r dess radie [11]. Radien på en partikel är halva medelavståndet mellan partiklarna, $r = \delta/2$.

Med systemet definierat är det nu möjligt att bestämma skalfaktorerna, grunden till detta introduceras i nästa del.

2.1.4 Skalfaktorernas relationer

Det som återstår för att erhålla en fullständig beskrivning av partikelfluiden är att bestämma skalfaktorerna som ingår i krafterna. För att göra detta finns teori framtagen som relaterar makroskopiska egenskaper till skalfaktorerna [1]. Dessa formler kan dock vara missvisande [2], och det är därför vanligt att ställa in partikelbaserade modeller genom att köra simuleringar för ett känt referensfall och ställa in parametrarna så att modellen ger samma resultat som i referensfallet [8], [7].

För att systemet skall bli termodynamiskt korrekt finns även ett antal relationer mellan skalfaktorerna som säkerställer att den relativa storleken på krafterna blir rätt.

Den första parametern att implementera är Λ i (1). Denna kan relateras till systemets tryck enligt

$$P = \frac{n}{2D} \int \Lambda K(r) r d\mathbf{r}. \tag{24}$$

Integralen i (24) går över en cirkel eller sfär med radien r_{cut} och P är trycket i systemet [2]. I två dimensioner ger (24)

$$P = \frac{\Lambda r_{cut}}{8}. \tag{25}$$

Genom att definiera trycket i systemet kan Λ besämmas. För att göra detta så behöver cut-off radien vara känd. Det finns ingen formel för denna men vanligt är att den väljs i storleksordningen 2δ [2].

Nästa storhet att definiera är ljudhastigheten c. Denna är definierad som

$$c^2 = \frac{k_B T}{m_i}, \tag{26}$$

där k_B är Boltzmanns konstant och T systemets temperatur [1].

För att få ett mått på hur trögflytande fluiden är kan en storhet som kallas för dimensionslös viskositet, Ω, införas [2], [9]. Denna definieras som

$$\Omega = \frac{\gamma r_{cut}}{Dc}. \tag{27}$$

Genom att variera värdet på Ω ändras även värdet på γ under antagandet att en cut-off radie definierats och c är beräknad enligt (26).

I FPM finns två formler härledda som relaterar modellens parametrar till fluidens kinematiska bulk respektive skjuvningsviskositet, ν_b och ν_s [1]. Dessa två formler är

$$\nu_b = \gamma n \left(\frac{A_2}{2D} + \frac{D+2}{2D} B_2 \right) + c^2 \frac{1}{\gamma D n (A_0 + B_0)} \tag{28}$$

och

$$\nu_s = \frac{1}{2} \gamma n \left(\frac{A_2}{2} + B_2 \right) + c^2 \frac{1}{2\gamma n (A_0 + B_0)}. \tag{29}$$

Dessa är härledda under antagandet att den konservativa kraften (1) är noll vilket endast är en bra appoximation för gaser under lågt tryck [1]. Parametrarna A_0, B_0, A_2 respektive B_2 är

$$A_0 = \frac{1}{D} \int A(r) d\mathbf{r}, \quad B_0 = \frac{1}{D(D+2)} \int B(r) d\mathbf{r} \tag{30}$$

och

$$A_2 = \frac{1}{D} \int A(r) r^2 d\mathbf{r}, \quad B_2 = \frac{1}{D(D+2)} \int B(r) r^2 d\mathbf{r}. \tag{31}$$

Integralerna går över en cirkel eller sfär med radie r_{cut}. Om cut-off radien är postulerad och partikeltätheten n definierad är de skalära funktionerna kända tack vare normeringsvillkoret (10). Detta leder till att ν_b och ν_s kan beräknas ur antingen (28) eller (29) med hjälp av de värdet på γ från (27). I två dimensioner blir värdena på parametrarna i (30) resp (31)

$$A_0 = \frac{1}{2n}, \quad B_0 = \frac{1}{8n} \tag{32}$$

och

$$A_2 = \frac{r_c^2}{10n}, \quad B_2 = \frac{r_c^2}{40n} \tag{33}$$

För att få en första uppskattning på γ går denna att lösa ut ur (28) resp (29) vilket ger

$$\gamma = \frac{10\nu_b}{r_c^2} \pm \sqrt{\left(\frac{10\nu_b}{r_c^2} \right)^2 - \frac{16c^2}{r_c^2}} \tag{34}$$

och

$$\gamma = \frac{40\nu_s}{3r_c^2} \pm \sqrt{\left(\frac{40\nu_s}{3r_c^2} \right)^2 - \frac{64c^2}{3r_c^2}} \tag{35}$$

Det framgår inte ur någon artikel om det är den positiva eller negativa roten som är fysikaliskt relevant. Därför måste båda varianterna testköras i programmet och slutsatser dras därifrån.

Den sista parametern som måste bestämmas är skalfaktorn σ i (6). Denna relaterar till massan, temperaturen och Boltzmanns konstant via den detaljerade balansekvationen

$$\sigma^2 = \frac{2k_B T \gamma}{m_i}. \tag{36}$$

Denna relation behövs för att relatera den stokastiska kraften till friktionskrafternas storlek som bestäms av γ [1].

Sammanfattning av teorin bakom Fluidpartikel-metoden FPM bygger på fyra krafter som alla representerar olika fysikaliska fenomen. Dessa skalas av tre typer av skalfaktorer, Λ, γ och σ och den stokastiska kraften är ett resultat av de tre första krafterna. För att simulera en viss fluid måste partikeltätheten n definieras, vilken bestämmer hur fin indelningen av fluiden i kluster är. Med denna definierad kan partikelmassa, tröghetsmoment och skalära funktioner bestämmas samt en lämplig cut-off radie definieras. När detta är gjort beräknas en första approximation av Λ, γ och σ enligt relationerna i 2.1.4. Dessa måste justeras bättre utefter ett referensfall, detta beskrivs i 2.2.1.

2.1.5 Samband i mekaniken

Med hjälp av de krafter och vridmoment som uppkommer i FPM så kan systemet uppdateras i tiden enligt sambanden i Newtons mekanik. Dessa samband relaterar krafter och vridmoment till accelleration och vinkelacceleration enligt

$$\frac{d\vec{v}_i}{dt} = \frac{1}{m_i}\sum_{i\neq j}\vec{F}_{ij} \tag{37}$$

och

$$\frac{d\vec{\omega}_i}{dt} = \frac{1}{I}\sum_{i\neq j}\vec{M}_{ij}. \tag{38}$$

Det gäller även att

$$\frac{d\vec{v}_i}{dt} = \vec{a}_i \tag{39}$$

och

$$\frac{d\vec{r}_i}{dt} = \vec{v}_i. \tag{40}$$

Här står $\vec{F}$ för kraft och $\vec{M}$ för vridmoment. Notationen ij står för att kraften verkar på partikel i och orsakas av partikel j. Summan går över alla partiklar j sådana att $|\vec{r}_i - \vec{r}_j| \leq r_{cut}$ [1]. Den totala kraften som verkar på en partikel definieras som

$$\vec{F}_{ij} = \vec{F}^K_{ij} + \vec{F}^T_{ij} + \vec{F}^R_{ij} + \vec{F}^S_{ij} \tag{41}$$

och vridmomentet som

$$\vec{M}_{ij} = -\frac{1}{2}\vec{r}_{ij} \times \vec{F}_{ij}. \tag{42}$$

Detta är ramverket i Fluid Partikel Metoden som utgör grunden för att kunna utföra simuleringar med den [1]. I nästa del så presenteras en metod för att ställa in modellen utifrån ett referensfall, vilket är det som är kvar för att ha en fullständig modell att kunna göra simuleringar med.

2.2 Inställning av modell

För att ställa in skalfaktorerna i partikelbaserade metoder är det vanligt att fluiden simuleras jämtemot ett referensfall för vilket flödet är väl känt [7], [8]. Anledningen till detta är att de teoretiska formler som finns ofta kan ge dåliga resultat främst på grund av att de är härledda utifrån förenklingar [1], [2]. Nedan presenteras en metod som är väl lämpad för partikelbaserade algoritmer.

2.2.1 Periodic Poiseuille Flow Method, PPFM

Metoden Periodic Poiseuille Flow Method är designad för att ställa in partikelbaserade modeller och presenteras utförligt i [7]. Denna metod har jämförts med andra metoder på partikelmodellen Dissipative Particle Dynamics och gett mycket bra resultat. Metoden bygger på att jämföra flödet som partikelmodellen ger med analytiska lösningar för Navier-Stokes modellen. Flödet som simuleras är det som uppkommer mellan två oändligt långa paralella plattor på avstånd d då fluiden strömmar utmed dessa plattor i en dimension, driven av gravitationsaccelerationen. I kontaktytan mellan fluiden och plattorna så sätts no-slip randvillkor, vilket betyder att fluides hastighet är noll här. Fluiden som simuleras måste vara Newtonsk, vilket de flesta reella fluider är.

För fallet med flöde mellan två plattor och no-slip randvillkor ger Navier-Stokes modellen följande funktion för hastighetsprofilen [7]

$$v_x(y) = \frac{\rho g_x}{2\mu}(yd - y^2). \tag{43}$$

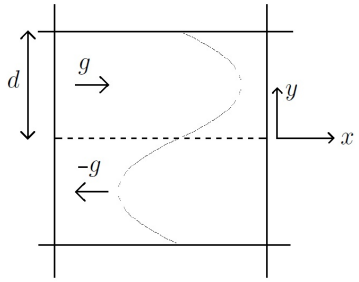
Här är ρ fluidens desitet, μ dess kinematiska viskositet och g_x är gravitationsaccelerationens x komposant. Medelhastigheten på flödet i x led gåt att ta fram som medelvärdet av $v_x(y)$ enligt [7]

$$\langle v_x \rangle = \frac{1}{d}\int_0^d v_x(y)dy = \frac{\rho g_x d^2}{12\mu}. \tag{44}$$

Reynolds-talet för detta flödet definieras som [7]

$$Re = \frac{\rho \langle v_x \rangle d}{\mu}. \tag{45}$$

För att kunna jämföra partikelmodellens med lösningarna på Navier-Stokes ekvationer måste den simuleras med no-slip randvillkor. En metod för att åstakomma detta föreslås i [7]. Metoden bygger på att dela in simuleringsområdet i två delar i vilka gravitationskraften tillåts verka i motsatt riktning, se figur 1. Genom att tillämpa periodiska randvillkor i y led simuleras i praktiken en oändlig serie av paralella, motriktade flöden. I gränsytan mellan områderna kommer fluiden att stå stilla, varpå no-slip randvillkor skapas här.



Figur 1: Skiss över Periodic Poiseuille Flow Method.

I figur 1 så är hastighetsprofilen enligt (43) inritad.

För att ställa in skalfaktorerna för partikelmodellen måste först modellen definieras enligt section 2.1. Efter detta behövs värden för fluidens dynamiska viskositet μ samt ett lämpligt avstånd mellan plattorna d definieras. När detta är klart utförs följande:

1. Definiera ett Reynoldstal, Re, lämpligen ett litet för att garantera laminärt flöde
2. Beräkna utifrån Re medelhastigheten $\langle v_x \rangle$ för flödet utifrån (45)
3. Beräkna utifrån $\langle v_x \rangle$ hur stor drivkraft g_x detta motsvarar utifrån (44)
4. Ställ in parametrarna Λ, γ samt σ med hjälp av P och Ω i (24) samt (27).
5. Kör simuleringen enligt beskrivningen ovan och bestäm utifrån detta $\langle v_x \rangle_{part}$ för partikelmetoden genom att medelvärdesbilda ett stort antal partiklars hastigheter
6. Jämför $\langle v_x \rangle_{part}$ med $\langle v_x \rangle$ i punkt 2. Om ej samma upprepa punkt 4. - 6.

Efter att skalfaktorerna är inställda är hela modellen definierad och redo att användas. Självklart krävs det ett datorprogram som klarar av att tillämpa metoden. Detta presenteras i nästa del.

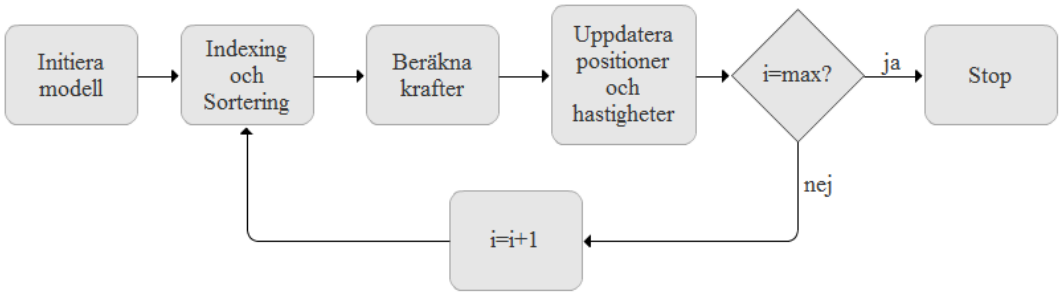
3 Implementering av Fluidpartikel-metoden

Implementeringen av FPM sker i form av ett program skrivet i programmeringsspråket C. Programmet är designat för att med så hög effektivitet som möjligt simulera en fluid med FPM.

Den enklaste typen av implementering är att låta varje partikel interagera med samtliga andra partiklar. Denna typ av implementering klarar dock inte av många partiklar eftersom det blir väldigt många interaktioner och detta resulterar i ett långsamt program. De interaktioner där avståndet mellan partiklarna är större än r_{cut} kan utelämnas helt. För att inte behöva beräkna avståndet mellan samtliga partiklar delas därför simuleringsområdet upp i ett rutmönster, där en ruta kallas för box. Endast avståndet mellan partiklar i angränsande boxar behöver nu beräknas om boxarnas storlek är tillräckligt stor.

Genom att tilldela varje partikel ett boxindex, givet av partikelns position, är det möjligt att endast låta partiklar i angränsande boxar interagera. Denna typ av indelning kan även tillämpas på ett tredimensionellt system utan några principiella skillnader [2].

I figur 2 visas en flödesschemat av hur programmet är strukturerat.



Figur 2: Ett flödesschemat visar hur programmet är strukurerat.

I programmets första del sätts de initiala parametrarna så som partikelmassa, antal partiklar, antal boxar och simuleringsområdets storlek. I denna del av programmet sätts även skalfaktorerna γ, σ och c.

Nästa steg är att tilldela varje partikel ett index utifrån partikelns position, detta index kallas för partikelns boxindex.

När tilldelningen är klar måste all data sorteras för att sedan kunna användas. Detta utförs med en sorteringsalgoritm som sorterar partiklarna utefter boxindex.

Kraftberäkningen stegar sig över alla boxar, för varje box ska krafter beräknas för partiklarna i boxen. Bara partiklar i samma eller angränsande box kommer påverka varandra. Kraftbidragen från de relevanta partiklarna summeras sedan.

När krafterna är kända uppdateras systemet med hjälp av Eulerlösaren, data sparas för analys och sedan kan nästa iterationssteg påbörjas.

3.1 Genomförande

I detta avsnitt beskrivs hur arbetet med implementeringen strukturerats under projektets gång. Även vilken hårdvara och mjukvara som använts beskrivs.

Från början utvecklades programvaran i Windows med en GCC kompilator. Vid ett senare tillfälle övergick utvecklingen till en Linuxmiljö där Intels parallellkompilator användes.

Skapandet av programmet skedde i flera stadier vilka finns listade i kronologisk ordning nedan.

- En testalgoritm utan parallellisering eller boxindelning skapades i Matlab för att ge en grundläggande förståelse för metodens implementering.
- Testalgoritmen implementerades i C.
- Boxindelningen utvecklades i C och testkördes.
- När programmet fungerade på en beräkningstråd paralleliserades koden för att köras på flera trådar.

Den dator som användes för körning av programmet använder en Intel i5 2500k processor med fyra trådar à 4600MHz. Operativsystemet som användes var Ubuntu version 11.10. Den kompilator som använts är Intels *Parallel Composer* som finns tillgänglig gratis för icke-komersiellt bruk[1].

3.2 Initialisering

I början av programmet ska alla begynnelsevillkor för systemet defineras, vilka är listade nedan.

- Antal partiklar, N_{tot}
- Antal boxar, M^2
- Startpositioner för partiklarna, $\vec{r}_{i,0}$
- Partiklarnas massa, m_i
- Partiklarnas initialhastighet, $\vec{v}_{i,0}$
- Partiklarnas initiala rotationshastighet, $\vec{\omega}_{i,0}$
- Krafternas skalfaktorer, $\gamma, \quad \sigma, \quad c$

Genom att variera dessa initialvillkor kan olika simuleringsfall betraktas.

[1] `http://software.intel.com/en-us/articles/intel-parallel-composer/`

3.3 Indexering och sortering

Det är nödvändigt att veta vilka partiklar som ska hanteras och i vilken ordning. För att hålla reda på vilka delar av systemet som ska interagera delas simuleringsområdet in i ett rutmönster. Genom att ge varje ruta ett index, så kallade boxindex, kan programmet navigera bland boxarna. Varje partikel tilldelas ett index utifrån vilken box den tillhör. Programmet sorterar sedan alla partiklar med avseende på boxindex. Exakt hur detta sker och och varför det är effektivt förklaras i följande avsnitt.

3.3.1 Boxindelning

Boxindelningen betyder att beräkningsdomänen delas upp i mindre delar och sedan utförs beräkningarna för varje liten del. Varje axel delas in i M delintervall vilket ger ett rutmönster med M^2 boxar. Varje box tilldelas nu ett index i stigande ordning från vänster till höger, rad för rad enligt figur 3. Varje partikel tilldelas sedan ett boxindex som bestäms av partikelns position. Index för partiklarna sparas i en lång vektor och kopplas till övrig data för partikeln.

Syftet med denna indelning är att begränsa antalet interaktioner till de som är relevanta. De relevanta interaktionerna är de där avståndet mellan partiklarna är mindre än cut-off radien. Genom att endast låta angränsande boxar interagera med varandra reduceras antalet beräkningar kraftigt.

Reduktion av antalet interaktioner En algoritm utan boxar som låter alla N_{tot} partiklar interagera med alla andra partiklar kommer utföra ett antal kraftberäkningar som är proportionellt mot N_{tot}^2. Genom att bara låta angränsande boxar interagera kan detta antal minskas drastiskt.

Låt medelantalet partiklar i en box vara antalet partiklar N_{tot} dividerat med antalet boxar M^2. Varje box interagerar med fyra andra boxar och sig själv, detta betyder att det antal interaktioner C som måste utföras kan uppskattas enligt

$$C = 5\left(\frac{N_{tot}}{M^2}\right)^2 M^2 = \mathcal{O}\left(\frac{N_{tot}^2}{M^2}\right) \tag{46}$$

Om antalet boxar är i samma storleksordning som antalet partiklar blir antalet beräkningar för denna typ av algoritmen proportionell mot N_{tot}.

3.3.2 Sortering

När partiklarna tilldelats boxindex sorteras de N_{tot} långa data-vektorerna efter detta index. Syftet med sorteringen är att programmet ska kunna hantera partiklarna boxvis och sedan hålla reda på vilka boxar som interagerar.

Sorteringen sker med Merge-Sort-metoden [12]. Den lista som ska sorteras innehåller boxindex och har N_{tot} element. Algoritmen delar in dessa index i två delar, detta sker rekursivt tills att hela listan delats in i N_{tot} listor. Då en lista med ett tal betraktas som sorterad är det nu möjligt att sammanfoga listorna genom att jämföra talen i dessa och lägga dem i rätt ordning. Listorna sammanfogas tills en lista med N_{tot} element har åstadkommits, denna är då sorterad och klar att användas.

Fördelen med just denna metod är att all data följer samma procedur vilket betyder att även partiklarnas data-vektorer blir sorterade på samma sätt som boxindex-vektorn. Detta är viktigt eftersom ingen partikeldata får skrivas över och ersättas med en annan partikels data. Detta beteende betyder att sorteringsalgoritmen är stabil. En förenklad matris som illustrerar resultatet av sorteringalgoritmen visas nedan.

Innan sortering:

$$P = \begin{bmatrix} \vec{r}_1 & \vec{r}_2 & \vec{r}_3 & \vec{r}_4 & \vec{r}_5 & \vec{r}_6 & \cdots \\ \vec{v}_1 & \vec{v}_2 & \vec{v}_3 & \vec{v}_4 & \vec{v}_5 & \vec{r}_6 & \cdots \\ bindex_4 & bindex_3 & bindex_1 & bindex_5 & bindex_2 & bindex_5 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Efter sortering:

$$\tilde{P} = \begin{bmatrix} \vec{r}_3 & \vec{r}_5 & \vec{r}_2 & \vec{r}_1 & \vec{r}_4 & \vec{r}_6 & \cdots \\ \vec{v}_3 & \vec{v}_5 & \vec{v}_2 & \vec{v}_1 & \vec{v}_4 & \vec{r}_6 & \cdots \\ bindex_1 & bindex_2 & bindex_3 & bindex_4 & bindex_5 & bindex_5 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Vektorerna $\vec{r}_i$ och $\vec{v}_i$ står för position och hastighet för partikel i. Index $bindex_n$ står för att partikeln tillhör box n. Efter sorteringen blir matrisen i rätt ordning med avseende på boxindex. Därmed kan programmet loopa över boxindex utan att ta hänsyn till hur partiklarna är ordnade i boxarna.

Antalet beräkningar i en Merge-Sort-metod är proportionellt mot $N_{tot} \log N_{tot}$ där N_{tot} är antalet element som ska sorteras. Detta är en förhållandevis snabb sorteringsalgoritm vilket är ytterligare en anledning att den används [12].

Att hitta partiklarna i en box För att hitta rätt bland partikeldatan måste start och slutindex i partikel-vektorn identifieras för varje box. När detta är känt identifieras de partiklar med index mellan start- och slut-index som partiklar i boxen.

Sorteringsalgoritmen tilldelar inte bara varje partikel ett index utan sparar också ner hur många partiklar som finns i varje box. Detta betyder att startindex för en box n kan bestämmas genom att summera antalet partiklar i alla boxar med index $m < n$. I figur 4 finns en förenklad bild av en sorterad partikelvektor och hur start och slutindex tolkas.

3.4 Beräkning av krafter

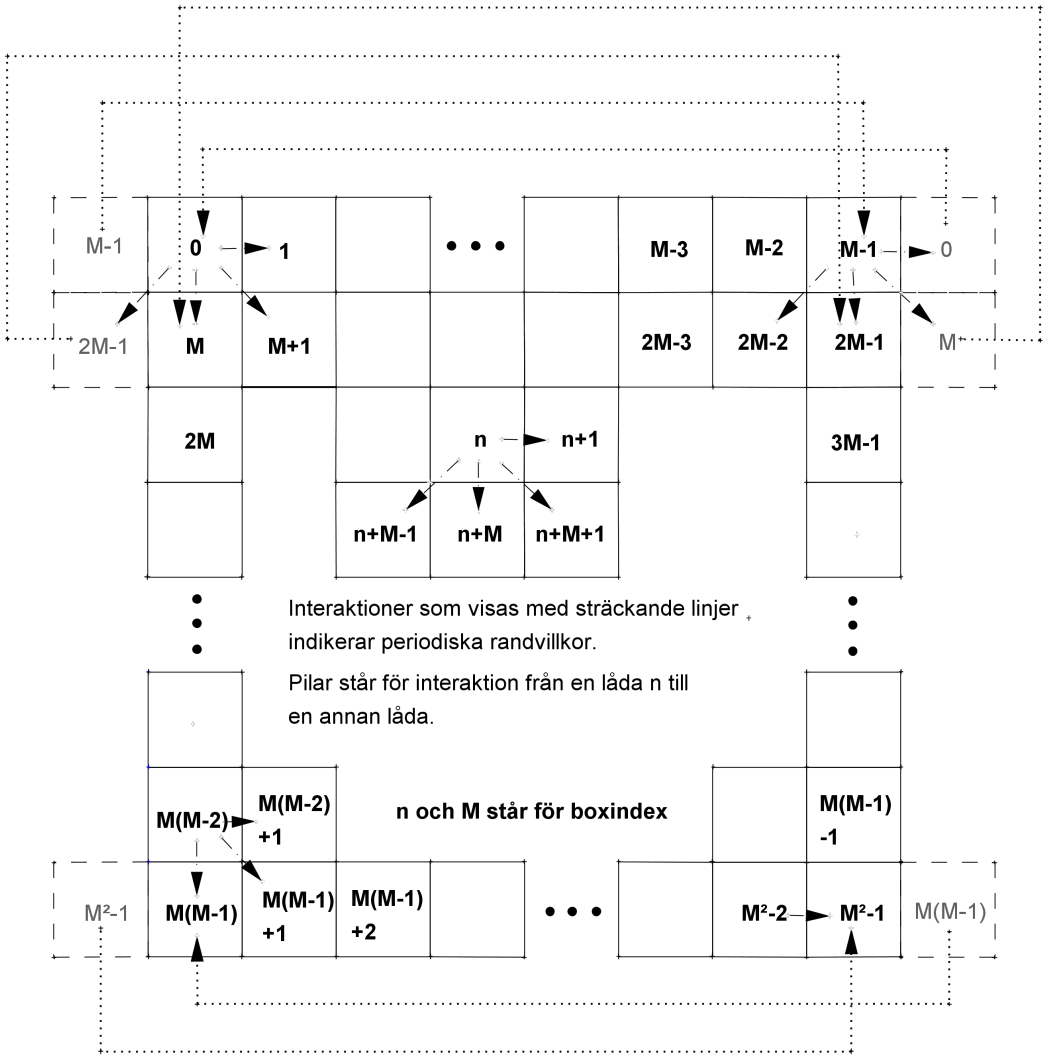
Efter att indexeringen och sorteringen för partiklarna är klar ska partiklarnas interaktioner med varandra beräknas. Detta sker i två steg, först med avseende på lådor och sedan med avseende på partiklar.

3.4.1 Interaktion mellan lådor

Det interaktionsmönster som valts bygger på att varje box ska interagera med sina grannar en gång. Interaktionerna mellan boxar finns listade i tabell 1, mönstret finns även illustrerat i figur 3.

Programmet är uppbyggt på ett sådant sätt att en interaktion mellan två lådor går båda vägarna. Detta betyder att om box n har interagerat med en box m måste inte box m interagera med box n.

I programmet implementeras dessa interaktioner genom en rad if-satser som utifrån boxindex använder villkoren i tabell 1 för att bestämma lådans position. Sedan utförs interaktionerna enligt figur 3.



Figur 3: Visualisering av hur boxarna interagerar med varandra.

Newtowns tredje lag I figur 3 framgår det att en låda ej behöver interagera med alla sina grannar. Anledningen är att krafterna mellan partiklarna följer Newtowns tredje lag. Detta innebär att den resulterande kraften för en partikelinteraktion är samma på båda partiklarna men har olika riktning. Detta betyder att interaktionskrafterna mellan två lådor endast behöver beräknas en gång för varje lådpar.

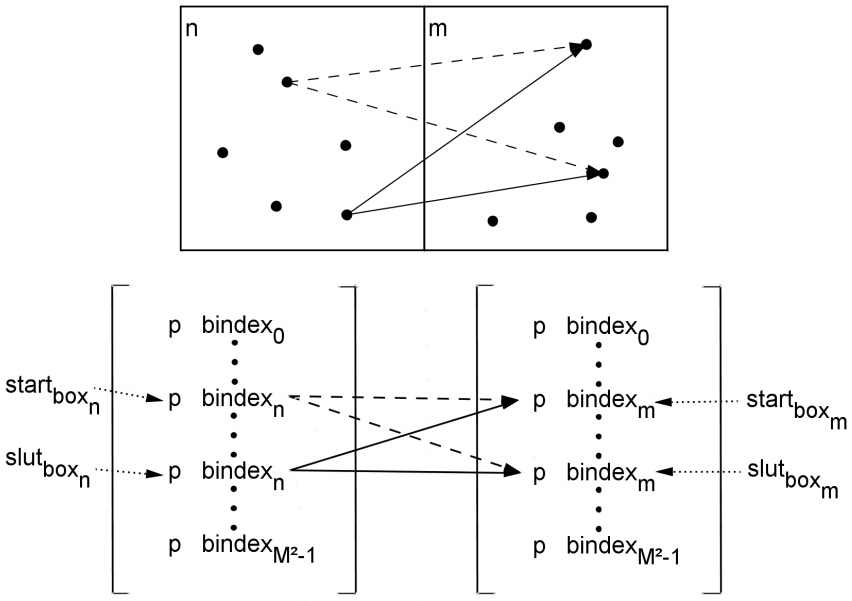
3.4.2 Interaktion mellan partiklar

För ett givet par av boxar ska partiklarna i boxarna interagera, dvs alla partiklar i box n ska interagera med alla partiklar i box m. Detta sker genom att stega över alla partiklar i box n och för varje steg beräkna krafterna från samtliga partiklar i box m. En illustration av detta finns i figur 4.

Den underliggande kraftberäkningen innehåller fyra komponenter som beräknas separat och adderas till den resulterande kraften på varje partikel. När det slutgiltiga kraften är beräknad på en partikel går algoritmen vidare till Eulerlösaren som uppdaterar systemet i tiden.

Tabell 1: Tabell över interaktioner mellan lådor där M är antalet boxar på en rad och $k, l \in \mathbb{N}$.

Lådindex n	Position	Interagerande lådor
$n = M \cdot k$ $\forall \quad 1 \leq k \leq M-2$	Vänstra randen	$n, \quad n+1, \quad n+M,$ $n+M+1, \quad n+2 \cdot M-1$
$n = k$ $\forall \quad 1 \leq k \leq M-2$	Övre randen	$n, \quad n+1,$ $n+M, \quad n+M \pm 1$
$n = M \cdot (M-1) + k$ $\forall \quad 1 \leq k \leq M-2$	Nedre randen	$n, \quad n+1$
$n = k \cdot M + (M-1)$ $\forall \quad 1 \leq k \leq M-1$	Högra randen	$n, \quad n+1-M, \quad n+1,$ $n+M, \quad n+M-1$
$n = k \cdot M + l$ $\forall \quad 1 \leq k, l \leq M-2$	Systemets inre	$n, \quad n+1, \quad n+M, \quad n+M \pm 1$
$n = 0$	Övre vänstra hörnet	$0, \quad 1, \quad M+1, \quad M$
$n = M-1$	Övre högra hörnet	$M-1, \quad 2M-1, \quad 2M-2, \quad M, \quad 0$
$n = M(M-1)$	Nedre vänstra hörnet	$M(M-1), \quad M(M-1)+1$
$n = M^2-1$	Nedre högra hörnet	$M^2-1, \quad M(M-1)$



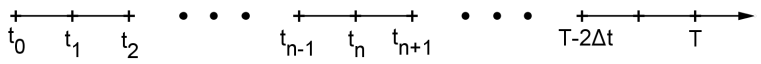
Figur 4: Visualisering av hur en partiklar i box n interagerar med partiklarna i en box m.

3.5 Tidsutveckling av systemet

I FPM hanteras stora mängder partiklar som alla uppdateras i tiden enligt sambanden i Newtons mekanik, se avsnitt 2.1.5. Partiklarna interagerar med hjälp av komplexa krafter som är både icke linjära och stokastiska. Detta leder till att en analytisk lösning till tidsutveckligen av $\vec{r_i}$, $\vec{v_i}$ samt $\vec{\omega_i}$ för alla partiklar $i \in [0, N_{tot}]$ är omöjlig att finna. Av dessa skäl måste numeriska metoder som kan implementeras i en dator användas.

För att lösa ett tidsberoende system numeriskt måste tidsintervallet diskretiseras så att t endast kan anta vissa diskreta värden t_n. Detta kan skrivas som $\{t = t_n \in [0, T], \quad n \in \mathbb{N}; \quad t_{n+1} - t_n = \Delta t\}$, se figur 5.

Det finns ett stort antal varianter av numeriska lösare och gemensamt för dessa är att de approximerar en storhets förändring över ett diskret tidssteg Δt. Eftersom endast storhetens utveckling approximeras introduceras generellt ett numeriskt fel för varje tidssteg. Valet av numerisk lösare är avgörande för att minimera detta fel.



Figur 5: Diskretiserat tidsintervall

Att uppdatera systemet i tiden kostar även beräkningskraft. I FPM måste varje partikel uppdateras individuellt i tiden vilket innebär att ett stort antal beräkningar måste utföras för varje tidssteg. För att få ett effektivt program bör därför den numeriska lösaren kräva minsta möjliga beräkningskraft.

Det är även vilktigt att den numeriska lösaren är stabil. Detta betyder att programmets funktion inte påverkas negativt eller att det i värsta fall slutar fungera. Utöver att minimera felet ϵ samt vara beräkningseffektiv och stabil är det även fördelaktigt om den numeriska lösaren är smidig och lätt att implementera.

3.5.1 Val av numerisk lösare

Låt vektorn $\vec{\xi}$ beteckna systemets storheter, då kan (37), (38) samt (40) skrivas som

$$\frac{d\vec{\xi}_i}{dt} = \vec{f}(\vec{\xi}(t)), \tag{47}$$

där förändringshastigheten per tidsenhet för systemets storheter är en funktion av alla systemets storheter. Genom att införa diskretiseringen kan (47) omformuleras som

$$\vec{\xi}(t_{n+1}) = \vec{\xi}(t_n) + \Delta t\vec{f}(\vec{\xi}(t_n)) + \mathcal{O}(\Delta t^2). \tag{48}$$

Här betecknar $\vec{\xi}(t_m)$ storheternas exakta värde i t_m. Genom att approximera $\mathcal{O}(\Delta t^2)$ till 0 erhålls

$$\vec{\xi}(t_{n+1}) \approx \vec{\xi}(t_n) + \Delta t\vec{f}(\vec{\xi}(t_n)). \tag{49}$$

Låt $\vec{\xi_n}$ beteckna det numeriskt beräknade värdet på $\vec{\xi}(t_n)$. Ur (49) definieras nu den numeriska lösaren Euler-Framåt som

$$\vec{\xi}_{n+1} = \vec{\xi}_n + \vec{f}(\vec{\xi}_n)\Delta t. \tag{50}$$

Funktionen $\vec{f}(\vec{\xi})$ kan även evalueres i t_{n+1}, detta ger den numeriska lösare som kallas för Euler-Bakåt, vilken definieras som

$$\vec{\xi}_{n+1} = \vec{\xi}_n + \vec{f}(\vec{\xi}_{n+1})\Delta t. \tag{51}$$

Varje partikel upplever i varje tidpunkt en acceleration $\vec{a}_i$ och en vinkelaccelleration $d\vec{\omega}_i/dt$. De har även en hastighet $\vec{v}_i$, ett läge $\vec{r}_i$ och en vinkelhastighet $\vec{\omega}_i$. Relationerna (39) samt (40) går med hjälp av (47) att identifiera i (50) vilket leder till

$$\vec{v}_{i,n+1} = \vec{v}_{i,n} + \vec{a}_{i,n}\Delta t, \tag{52}$$

$$\vec{r}_{i,n+1} = \vec{r}_{i,n} + \vec{v}_{i,n}\Delta t. \tag{53}$$

Här är $\vec{a}_{i,n}$ bestämd via (37) med hjälp av identiteten (39). Kraften i (37) är i sin tur en funktion av läge, hastighet och vinkelhastigheten för systemet av partiklar i en given tidpunkt. För vinkelhastgheten $\vec{\omega}_i$ gäller även att

$$\vec{\omega}_{i,n+1} = \vec{\omega}_{i,n} + \frac{d\vec{\omega}_{i,n}}{dt}\Delta t, \tag{54}$$

där $d\vec{\omega}_i/dt$ är bestämd av (38) via (42), vars högerled är en funktion av läge, hastighet och vinkelhastighet i en given tidpunkt.

Den numeriska algoritmen Euler-Bakåt är analog med Euler-Framåt, skillnaden är att $\vec{a}_i$, $\vec{v}_i$ samt $d\vec{\omega}_i/dt$ evalueras i t_{n+1} istället. I Euler-Framåt kan systemets tillstånd i t_{n+1} beräknas rakt fram om systemets tillstånd i t_n är känt. Givet att systemet simuleras utifrån kända initialvillkor är denna metod alltid möjlig att tillämpa. I Euler-Bakåt måste vissa storheters värde i t_{n+1} kunna beräknas för att systemet skall kunna uppdateras i tiden. För vissa system går det att använda denna metoden men för FPM blir det mycket svårt tack vare de komplexa krafterna som utgör högerledet i (37).

Euler-Framåt är därför den enda av de två lösarna som är smidig att implementera och som dessutom kan förväntas kräva lite beräkningskraft. När högerledet i (50) evalueras i t_n introduceras ett fel. I Euler-Framåt uppdateras systemet i varje tidssteg med hjälp av storheternas värde i t_n och därför introduceras ett systematiskt fel. För ett mekaniskt system innebär ett systematiskt fel hos till exempel hastigheten att denna antingen hela tiden höjs eller sänks mer än den borde från tidssteg till tidssteg. Detta resulterar i att energin (kinetisk) hos systemet hela tiden höjs eller sänks mer än den borde.

Genom att kombinera Euler-Framåt och Euler-Bakåt erhålls en lösare som kallas för Euler-Cromers metod [5], [6]. Euler-Cromers metod är en symplektisk Euler-lösare som definieras enligt

$$\vec{v}_{i,n+1} = \vec{v}_{i,n} + \vec{a}_{i,n}\Delta t, \tag{55}$$
$$\vec{r}_{i,n+1} = \vec{r}_{i,n} + \vec{v}_{i,n+1}\Delta t. \tag{56}$$

Första steget för att uppdatera hastigheten följer Euler-Framåt och andra steget för att uppdatera läget följer Euler-Bakåt. Denna lösare går att använda rätt fram likt Euler-Framåt [5] [6].

3.5.2 Jämförelse mellan Eulerlösare

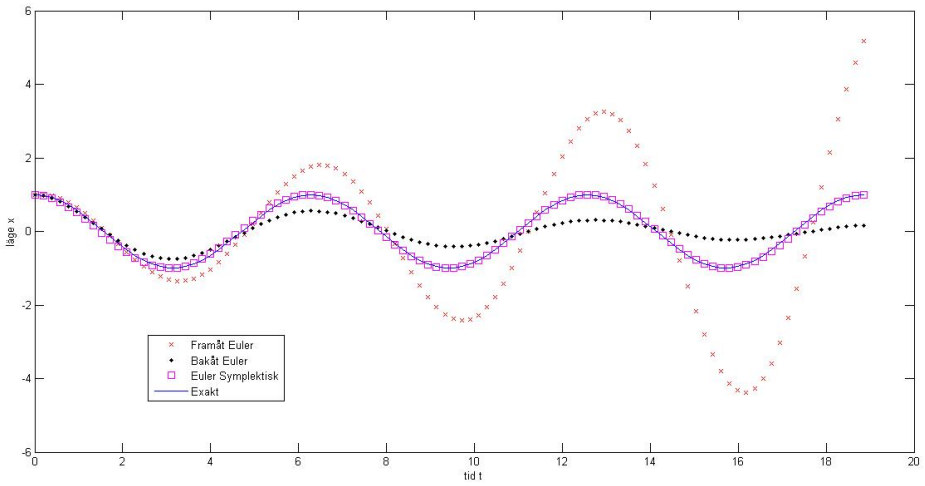
Euler-Framåt samt Euler-Cromers metod uppfyller kraven på att vara smidiga att implementera samt kräva så lite beräkningskraft som möjligt. Det sista kravet som skall uppfyllas är att lösaren genererar så små numeriska fel ϵ som möjligt. För att undersöka detta har jämförelser gjorts utefter vilka resultat de olika lösarna ger i jämförelse med analytisk härledda lösningar för några kända fysikaliska system. Jämförelserna har gjorts för den Harmoniska Oscillatorn samt den drivna och dämpade Harmoniska Oscillatorn. Differentialekvationerna samt de analytiska lösningarna till dessa är tagna från [11].

Harmoniska Oscillatorn, HO Den Harmoniska oscillatorn beskriver en vikt som hänger i en linjär fjäder med fjäderkonstant k. Denna dras ut från sitt jämviktsläge och släpps sedan. Systemet är helt idealt, vilket betyder att vikten inte dämpas av luftmotstånd eller liknande, och skall därför behålla en konstant svängningsamplitud. Differentialekvationen som beskriver systemet är

$$\frac{d^2x}{dt^2} + ax = 0, \tag{57}$$

där x är förskutningen från jämviktsläget (amplituden) och $a = k/m$ är kvoten av fjäderkonstanten och viktens massa. Simuleringsresultaten för tidsutvecklingen av förskutningen x, tillsammans med den analytiska lösningen, visas i figur 6.

Ur denna figur kan utläsas att amplituden för svängningen ständigt ökar resp minskar för Euler-Framåt resp Euler-Bakåt. Detta betyder att dessa två Eulerlösare introducerar ett systematiskt fel som gör att amplituden och därmed även energin ständigt ökar resp minskar.



Figur 6: Harmonisk Oscillator. $a = 1, t \in [0, 6\pi], \Delta t = 0.1885$

Euler-Cromers metod följer dock mycket väl den exakta lösningen och ger ett avsevärt bättre resultat.

Dämpad och Driven Harmonisk Oscillator, DDHO Det andra och fysikaliskt mest relevanta fallet som undersökts består av ett system med en vikt som hänger i en linjär fjäder samt utsätts för en hastighetsberoende dämpning och en drivkraft. Drivkraften i detta fallet är en sinusformad rörelse som svänger med en viss frekvens. Differentialekvationen som beskriver systemet är

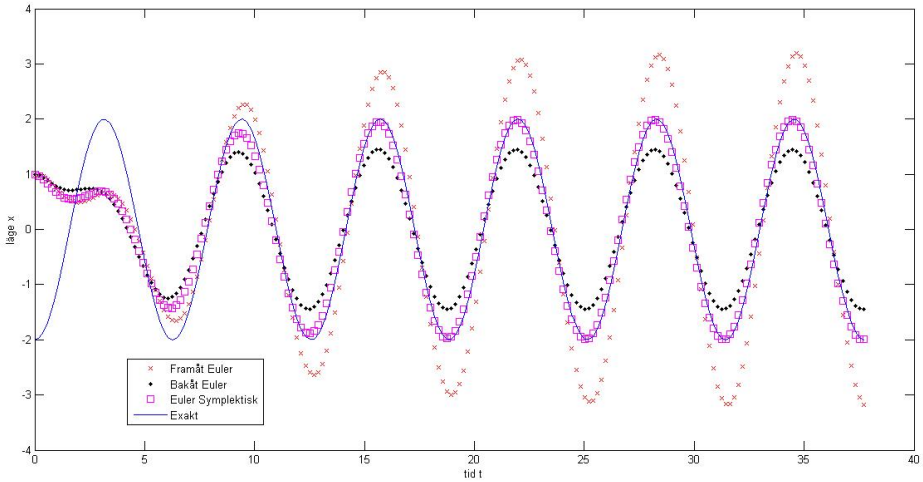
$$\frac{d^2x}{dt^2} + b\frac{dx}{dt} + ax = A\sin(\omega t), \tag{58}$$

där a är samma som i (57), $b = c/m$ är kvoten mellan dämpningskonstanten och viktens massa, $A = F_0/m$ är kvoten mellan den drivande kraftens amplitud och viktens massa och ω är den drivande kraftens vinkelfrekvens.

Denna modell kan användas för att undersöka hur en numerisk lösare påverkar det system som FPM beskriver. I FPM påverkas varje partikel av fyra olika krafter. Den första kraften är en konservativ och rent repulsiv kraft som uteslutande beror av avståndet mellan partiklarna. Denna kraften strävar efter att placera partikeln så långt från sina grannar som möjligt, vilket är i mitten av alla de partiklar som omger den. Detta är precis vad fjädern i DDHO modellen strävar efter, eftersom den ständigt vill återföra vikten till sitt jämvikstläge. I FPM ingår även två kraften som beror av hastighet och rotation och som ger upphov till friktion. Friktionen motverkar partiklarnas rörelse och rotation. Dessa krafter motsvaras av den dämpande kraften i DDHO modellen som beror av hastigheten. Slutligen finns det en stokastisk kraft i FPM som alltid uppkommer oberoende systemets rörelse. Den tillsammans med yttre krafter, till exempel gravitation, driver ständigt på systemet av partiklar. Dessa krafter motsvaras av den drivande kraften i DDHO modellen.

Resultatet av simuleringen av tidsutvecklingen som de tre olika lösarna ger i jämförelse med den analytiska lösningen för steady state visas i figur 7.

Steady state innebär att eftersom systemet både drivs på och dämpas kommer svängningen på vikten till slut att ställa in sig i ett jämviktsläge då energin som driver systemet precis är lika med energin som plockas ut ur systemet genom dämpning. figur 7 visar tydligt att Euler-Cromers metod är bäst på att uppskatta den exakta svängningen vid steady state. Euler-Framåt konvergerar mot en för stor amplitud och Euler-Bakåt konvergerar mot en en



Figur 7: Driven Dämpad Harmonisk Oscillator. $a = 1, b = 0.5, A = 1, \omega = 1, t \in [0, 12\pi], \Delta t = 0.1885$

för liten amplitud. Även flödet som FPM beskriver kommer konvergera mot steady state och det är därför viktigt att Eulerlösaren tillåter systemet att konvergera mot rätt sluttillstånd.

Valet av Eulerlösare avgör hur mycket fel som introduceras i systemet och för att få så bra simuleringsresultat som möjligt krävs att detta felet är så litet som möjligt. Resultaten av simuleringarna tillsammans med tidigare angivna fördelar är grunden till att Euler-Cromers metod enligt (56) valt att användas för att uppdatera hastighet och läge. Vinkelhastigheten $\vec{\omega}_i$ uppdateras med hjälp av Euler-Framåt enligt (54). Även partiklarnas fasvinkel ϕ kan uppdateras utifrån den beräknade vinkelhastigheten i t_{n+1} enligt Euler-Bakåt, vilket skulle Euler-Cromers metod för vinkelhastigheten och fasvinkeln. Men i FPM är endast vinkelhastigheten av intresse varpå detta utelämnas.

3.6 Parallellisering

Moderna datorer använder sig av processorer med flera kärnor vilket betyder att flera beräkningstrådar jobbar parallellt. För att kunna utnyttja prestandan hos flertrådade processorer måste koden anpassas för att sprida ut beräkningarna på dessa trådar. Algoritmen som delar upp simuleringsområdet i boxar lämpar sig speciellt bra för parallellisering då beräkningarna för boxarna är oberoende av varandra.

Det finns flera kompilatorer som automatiskt optimerar koden för flera trådar men dessa varierar kraftigt i kvalité och gränssnitt. Den kompilator som använts kan hantera OpenMP vilket låter användaren styra över parallellberäkningarna i större grad.

En parallelliserad kod skapas i OpenMP genom att först deklarera ett område av koden som ska paralleliseras, i ett sådant område kan sedan specifika loopar köras på flera trådar. Det är också möjligt att tala om för processorn hur många iterationssteg varje tråd ska utföra varje gång den tilldelas beräkningsjobb.

3.6.1 Parallellisering av indexering och sortering

Indexeringen är enkel att parallelisera då det endast är en for-loop med oberoende steg. Detta betyder att det inte spelar någon roll i vilken ordning stegen i loopen utförs vilket är en förutsättning för att loopen ska vara parallelliserbar.

Sorteringsalgoritmen som används är vald för att kunna parallelliseras på ett bra sätt. Detta är inte lika självklart som i fallet med indexeringen då algoritmen är rekursiv, den anropar alltså sig själv i en loop som avslutas när del-listorna endast har ett element. Detta löser parallelkompilatorn genom att fördela del-listorna till olika trådar. När alla trådar arbetar sorteras del-listorna separat i varje tråd för att sammanfogas när alla trådar är klara.

3.6.2 Parallellisering av kraftberäkningar

Då interaktionerna mellan boxar likt indexeringen är oberoende av ordning är det möjligt att utföra dessa beräkningar parallellt. Detta utförs genom att parallellisera loopen som låter alla boxar interagera.

I praktiken betyder det att iterations-stegen i loopen fördelas över trådarna och resulterar i att processorerna arbetar parallellt med olika boxar. Kraftbidraget på partiklarna i en box adderas till en, för tråden, individuell kraftvektor. Dessa summeras när alla iterationssteg är klara för att bestämma den totala resulterande kraften.

Problematik med reduktionsvariabler Boxformuleringen innebär problematik vad gäller parallelliseringen. Detta beror på att alla kraftbidrag ska summeras till samma kraftvektor vilket innebär att flera trådar riskerar att skriva till samma minnesplats. Denna typ av problem är inte omedelbart uppenbara då det kan resultera i en felaktig summa utan att programmet kraschar. Genom att specificera att de olika trådarna ska skriva till olika minnesplatser kan problem av denna typ undvikas, tre olika tillvägagångssätt beskrivs nedan.

- Deklarera nya variabler i parallellområdet. Varje tråd skapar en ny temporär variabel och adderar kraftresultanterna till denna. Sedan adderas denna privata kraftvektor till den gemensamma under kontrollerade former med hjälp av en sektion som endast utförs av master-tråden och på så sätt undviker minneskonflikter.
- Deklarera två variabler utanför parallellområdet och låt en av dessa vara privat. Detta resulterar i att varje tråd skapar en egen kopia av den privata variabeln som sedan försvinner när tråden har arbetat klart. De privata variablerna adderas innan parallellområdets slut till den gemensamma. Denna addition måste då ske i en sektion som endast utförs av en tråd precis som när reduktionsvariabeln deklareras i parallellområdet.
- Deklarera lika många reduktionsvariabler som det finns trådar men gör detta utanför parallellområdet. Genom att sedan identifiera vilken tråd som skriver i parallellområdet är det möjligt att skriva till olika reduktionsvariabler. På detta sätt skriver två trådar aldrig till samma minnesplats. När loopen är klar adderas reduktionsvariablerna under kontrollerade former. Vi valde detta angreppssätt eftersom det visade sig vara enklast att implementera.

4 Resultat

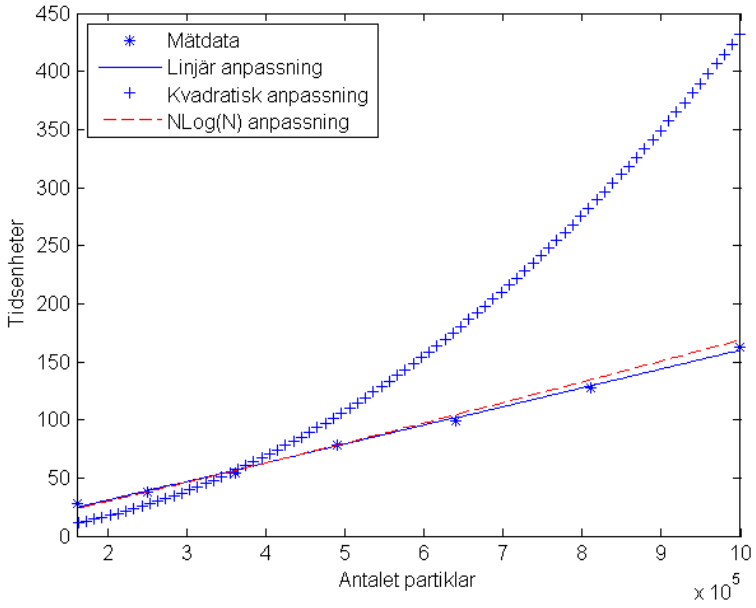
Resultaten presenteras i två delar, en som behandlar prestandan hos programmet och en del som behandlar resultaten från simuleringarna.

4.1 Prestanda

Att undersöka hastigheten på programmet är viktigt då snabbheten avgör programmets möjligheter att göra simuleringar. Ett snabbare program kan simulera ett längre tidsförlopp och fler partiklar vilket är bärande i denna metod. I följande avsnitt presenteras effektiviseringen till följd av boxindelningen och parallelliseringen.

4.1.1 Resultat av boxindelningen

Genom att variera antalet partiklar och sedan optimera antalet boxar för antalet partiklar testades hur effektiv boxindelningen är. I figur 8 visas sambandet mellan antalet partiklar och tiden för beräkningarna.



Figur 8: Figuren visar tiden för ett beräkningssteg som funktion av antalet partiklar.

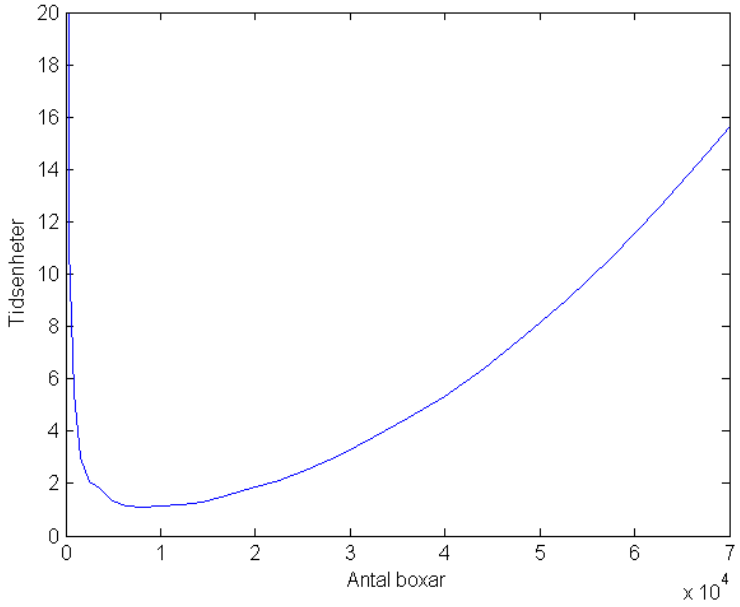
Antalet partiklar har varierats mellan 160 000 och en miljon med optimerat boxantal för varje körning. En jämförelse mellan felen visar att den linjära uppskattningen har ett 75% mindre fel än den logaritmiska. Detta betyder att algoritmen ligger närmast ett linjärt beroende.

Det är också relevant att visa hur hastigheten på beräkningarna skalar med antalet boxar. Grafen i figur 9 visar körningar med 250 000 partiklar och varierande boxantal. Speciellt viktig är formen på denna kurva. Eftersom det existerar ett minsta antal boxar skiljt från en box betyder det att boxindelningen effektiviserar beräkningarna jämfört med en algoritm som ej använder boxar.

4.1.2 Resultat av parallelliseringen

Resultatet av parallelliseringen redovisas i tabell 2. [2]

[2] Tack till Thomas Ericsson för testkörningen på Chalmers beräkningsserver.



Figur 9: Figuren visar tiden för att utföra ett tidssteg med 250 000 partiklar med varierande antal boxar.

Tabell 2: Körningstider med olika antal beräkningstrådar redovisas. Även effektiviteten för varje tråd redovisas där en ensam tråd arbetar med 100% effektivitet.

Antal trådar[st]	Tid[s]	Effektivitet [%]
1	57.00	100
2	31.80	89.6
4	18.00	79.2
8	8.70	81.2
16	5.00	71.3

Det är tydligt att parallelliseringen är effektiv. I tabell 2 finns även effektiviteten hos de individuella trådarna listade. Effektiviteten är definierad som

$$E = 100\frac{T_1}{k \cdot T_k} \tag{59}$$

där k är antalet trådar och T_k är tiden att utföra beräkningarna för k trådar.

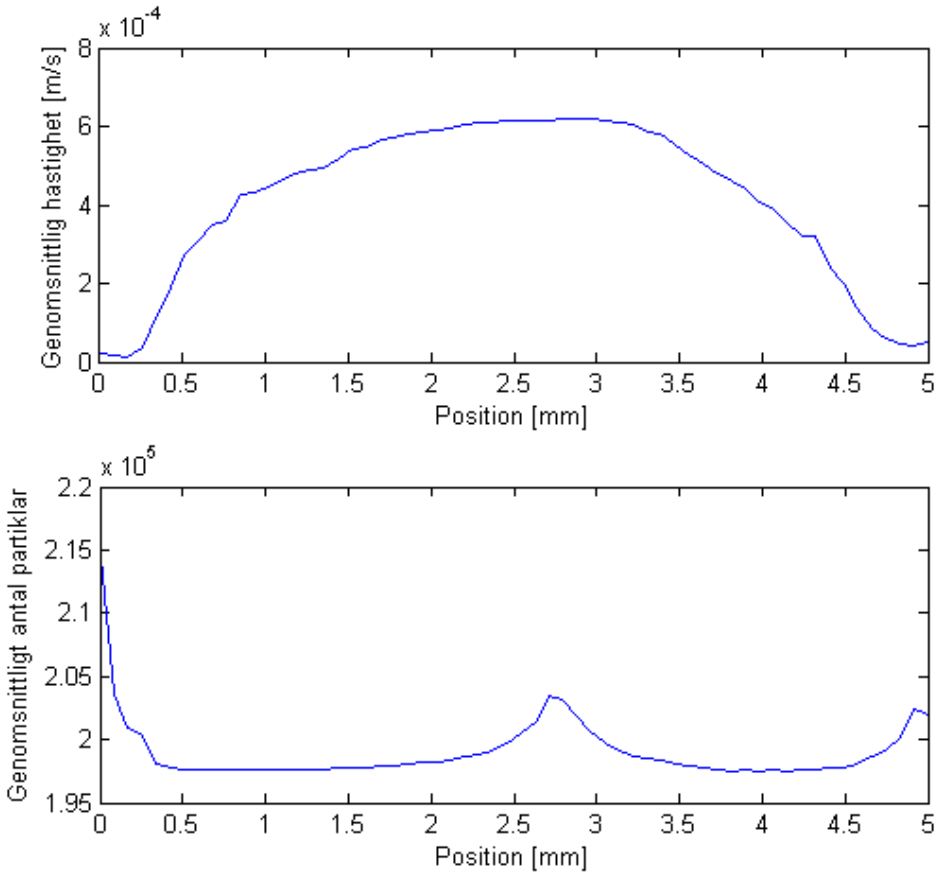
Resultaten visar att trådarna blir mindre effektiva när antalet går upp.

4.2 Simuleringar

Som ett första test simuleras vatten som flödar mellan två oändligt långa stillastående områden av partiklar. De nio första och sista boxarnas partiklar i vertikalt led står stilla och motsvarar det stillastående området.

Beräkningsdomänen har en kvadratisk form med sidan 0.5cm. För att åstadkomma jämvikt används ett periodiskt randvillkor. De partiklar som går utanför simuleringsområdet translateras tillbaka till början av området, en illustration av detta finns i figur 3. Detta periodiska randvillkor innebär i praktiken att ett oändligt långt flöde simuleras.

Figur 10 visar hastighetsprofilen när 250 000 partiklar flödat tills jämvikt inställt sig. Även



Figur 10: Den övre figuren visar hastighetsprofilen efter 4s. Den undre figuren visar fluidens densitet.

systemets densitet illustreras och som figur 10 visar fluktuerar den något. Testkörningar har visat att densiteten fluktuerar mindre när de konservativa krafterna ökar.

De skalfaktorer som använts är framtagna enligt avsnitt 2.1 och finns listade i tabell 3. Dessa är inte kalibrerade och är därför bara en första approximation. För att ge en mer illustrativ hastighetsprofil har den hastighetsberoende kraftens storlek tiodubblats.

Tabell 3: Skalfaktorer och simuleringsparametrar.

Λ	10^{-12} $[N]$
γ	10^{-2} $[1/s]$
σ	10 $[m/s^{(3/2)}]$
c	10^{-5} $[m/s]$
Δt	10^{-4} $[s]$
M^2	$8.1 \cdot 10^3$ $[st]$

Hastighetsprofilen i figur 10 är väldigt lik den som ges av den analytiska lösningen till Navier-Stokes i ekvationer, se (43). Hastigheten för partiklarna är lägre nära väggarna och har sin topp i mitten av området när jämvikt ställt in sig.

5 Diskussion

I denna del av rapporten diskuteras teorin och resultaten utifrån de mål och det syfte som projektet haft. Relationen mellan teorin och implementeringen diskuteras samt styrkor och svagheter i resonemang och implementering.

5.1 Teori

I denna del diskuteras den teori bakom fluidpartikel-metoden som presenterats i avsnitt 2 i förhållande till hur den kan tillämpas och vilka möjligheter/problem som finns med detta.

5.1.1 Krafterna i modellen

FPM är en vidareutveckling och generalisering av Dissipative Particle Dynamics [1]. Båda dessa algoritmer är designade för simuleringar på mesoskalan och utgår båda från att fluiden tolkas som kluster. En stor skillnad mellan DPD och FPM är att partiklarna kan rotera i FPM. Därför tillkommer även en rotationsberoende kraft.

Krafterna i FPM modellerar fler typer av interaktioner mellan partiklarna än i DPD. Det är framst friktion och den stokastiska kraften som är mer komplexa i FPM. Det är därför intressant att undersöka vilka konsekvenser detta får för att modellera komplexa flöden. Med komplexa flöden avses här flerfasflöde, flöde i krångliga geometrier, kolloid suspensioner, polymer suspensioner med mera.

5.1.2 Tolkning av en partikelfluid

Att tolka fluiden som uppbyggd av droppar eller kluster skall endast ses som en modell. Relevansen i tolkningen är inte det viktigaste utan att modellen ger bra resultat. Det finns problem med att tolka en fluid som uppbyggd av kluster. Problemet ligger i hur massa och tröghetsmoment för partiklarna skall bestämmas utifrån tolkningen som droppar av okänd form. Någon teori till detta finns inte att tillgå så det är upp till den som vill använda modellen att bestämma detta själv.

I detta arbete har alla systemets storheter beräknats i SI enheter. Anledningen till att inte reducerade enheter användts eftersom det var mer naturligt att beräkna systemets egenskaper i verkliga enheter. Dessutom kommer alla mätdata från simuleringarna att vara i rätt enheter och därför direkt kunna läsas av. Detta tillvägagångssätt har ej påträffats tidigare i simuleringar med FPM. Det är därför oklart om det är möjligt att implementera metoden i SI enheter. Skulle detta inte gå är det en klar nackdel med FPM.

5.1.3 Teorin till modellens parametrar

Skalfaktorerna och de skalära funktionernas relationer, se avsnitt 2.1.4 samt 2.1.2, behövs för att göra modellen termodynamiskt stabil. Endast de klara formlerna har utnyttjats och den bakomliggande teorin till dessa har utelämnats. Utan dessa relationer är modellen oanvändbar i det avseendet att den inte klarar av att reproducera fysikaliskt relevanta flöden.

Valet av de skalära funktionernas form är fri bortsett från att de begränsas av en cut-off radie och uppfyller ett normeringsvillkor (10). I detta arbete används ganska enkla skalära funktioner av samma typ som i [2]. För att förbättra modellen kan skalära funktioner som bättre beskriver avståndsberoendet användas. Förslag på en bättre form på $K(r)$ i (1) beräknad med hjälp av Molecular Dynamics föreslås i [10]. Det är dock osäkert om denna form ger ett avsevärt bättre resultat. Dessutom innebär mer komplexa uttryck att mer beräkningskraft krävs. Av dessa anledningar användes de enkla skalära funktionerna.

Utifrån de skalära funktioner som beskriver de två dissipativa krafterna bestäms de skalära funktionerna i den stokastiska kraften. Dessa relationer beskriver hur den stokastiska kraften beror av de dissipativa krafterna genom att ställa in deras relativa avståndsberoende. Ekvation (36) beskriver dessutom hur den stokastiska kraftens amplitud relaterar till de dissipativa krafterna. Dessa två relationer krävs för att modellen skall bli termodynamiskt stabil.

Teorin som beskriver hur skalfaktorerna relaterar till de makroskopiska egenskaperna tryck och viskositet, se (24) samt (28) (29), förväntas inte ge bra resultat utifrån fluidens makroskopiska egenskaper. Ekvationerna (28) och (29) är framtagna under antagandet att den konservativa kraften i (1) är noll och därmed systemet är väldigt kompressibelt. För en vätska är detta antagandet helt orimligt eftersom de flesta vätskor brukar vara praktiskt taget inkompressibla. Därför bör dessa relationer endast användas som en första approximation i en iterativ procedur för att bestämma modellens skalfaktorer.

5.2 Programmet

Vid utvärderingen av programmet är hastigheten på algoritmen, specifikt hur väl denna skalar mot antalet beräkningstrådar och partiklar en viktig aspekt. I denna del diskuteras programmets effektivitet utifrån resultaten samt svårigheter med implementeringen.

5.2.1 Diskussion om boxindelning

Boxindelningens syfte är att interaktioner mellan partiklar som inte känner av varandra ska ignoreras på ett så effektivt sätt som möjligt. I detta avsnitt diskuteras resultaten av boxindelningen.

Skalning mot antal partiklar En algoritm som inte tar hänsyn till distansen mellan partiklarna kommer i varje tidssteg behöva låta alla N_{tot} partiklar interagera med varandra vilket betyder att antalet beräkningar blir proportionellt mot N_{tot}^2. Tiden för att utföra ett sådant tidssteg kommer därför vara proportionellt mot antalet partiklar i kvadrat.

Antalet beräkningar mellan partiklar för ett system med boxar visar sig däremot bli proportionell mot N_{tot} gånger någon konstant. I (46) visas att antalet kraftberäkningar skalar linjärt mot antalet partiklar. Detta förklarar varför den $N_{tot} \log N_{tot}$ beroende anpassningen är sämre än den linjära. Sorteringsalgoritmen tar nämligen försumbart liten tid i förhållande till kraftberäkningarna och påverkar därför inte programmets skalning mot antalet partiklar.

Skalning mot antal boxar Ekvation (46) antyder att fler boxar alltid skulle resultera i lägre körtid. Det är dock tydligt från figur 9 att det finns ett optimalt antal boxar. Detta beror på att det även kostar beräkningskraft att tilldela boxindex och sortera all data. Ekvation (46) tar inte hänsyn till de beräkningar som krävs för att sortera, indexera och låta boxarna interagera. I figur 9 börjar beräkningstiden stiga när antalet boxar ökar mycket. När beräkningstiden börjar öka har en brytpunkt nåtts då det inte längre är effektivt att dela upp området i fler boxar.

För att hitta ett optimalt förhållande mellan boxantal och antalet partiklar är det lättaste sättet att testa algoritmen över ett relativt lågt antal tidssteg och variera boxantalet tills en så kort körtid som möjligt uppnås.

Att det optimala antalet boxar är större än en enda box är ett viktigt resultat. Det betyder att beräkningskraften för att dela upp systemet i boxar är mindre än den kraft som krävs för att låta samtliga partiklar interagera med varandra.

5.2.2 Diskussion om parallelliseringen

Då det är kraftberäkningarna som tar i princip all beräkningstid är den mest intressanta delen att titta på just parallelliseringen av kraftberäkningarna. Parallelliseringen ligger över loopen som behandlar boxar, detta betyder att varje tråd tilldelas ett antal boxar och sedan utförs kraftberäkningarna för partiklarna i dessa boxar. Rent konkret delas alltså boxarna ut till trådarna och behandlas parallellt. Det är tydligt från tabell 2 att detta är en mycket effektiv loop att parallellisera även om den gemensamma kraftvektorn måste behandlas som en reduktionsvariabel.

Skalning mot antalet trådar I tabell 2 är det tydligt att programmet jobbar effektivt på flera trådar. Anledningen till att inte beräkningstiden halveras när antalet trådar fördubblas är att det kräver beräkningskraft att fördela arbetet över trådarna. Detta betyder att varje tråd blir mindre effektiv när antalet ökar.

Det visar sig att när sexton trådar arbetar parallellt är varje tråd trettio procent mindre effektiv än när endast en tråd används. Detta är dessvärre en av konsekvenserna med flertrådade beräkningar och effektiviteten är ändå förhållandevis bra.

En intressant observation är att åtta trådar arbetar effektivare än fyra, detta beror med största sannolikhet på hur processorn är uppbyggd. Troligtvis har varje processor åtta trådar och är designad för att effektivt arbeta med alla samtidigt.

Begränsningar med OpenMP OpenMP använder en typ av parallellisering som kräver delat minne för beräkningsenheterna, det betyder att alla trådar måste komma åt samma gemensamma minne. Så är inte fallet i moderna superdatorer. Minnet är istället gemensamt för processorer som ligger på samma moderkort och moderna kluster består ofta av flera hundra kort som är sammankopplade. En specifik, så kallad master-tråd används sedan för att sköta kommunikationen mellan processorerna.

Med andra ord är inte programmet direkt körbart på kluster med separat minne för processorerna. Detta kräver en omprogrammering med hjälp av MPI-syntax vilket betyder att all kommunikation trådar emellan sker manuellt. Denna typ av parallellprogrammering är långt mer omständigt än med OpenMP-syntax.

Det är möjligt att utveckla en MPI-baserad programvara med den Open-MP baserade koden som bas. Kommunikationen mellan trådar på samma moderkort skulle ske via det gemensamma minnet och på så sätt minska datamängden som master-tråden måste hantera.

Reduktionsvariabler i Fortran Algoritmen som används kräver att flera trådar måste komma åt samma minnesplats. Programmeringsspråket *C* saknar vektorreduktion, vilket betyder att en vektor som ska hanteras av flera trådar måste bahandlas manuellt. Detta leder till att parallell-loopen över boxinteraktionerna kräver speciellt anpassad kod för att inte skapa minneskonflikter mellan trådar. I Fortran är det möjligt att direkt deklarera en reduktionsvektor utan någon övrig modifikation av koden. Vid parallellisering av denna typ av algoritm är alltså Fortran att föredra framför *C* då koden blir enklare.

5.2.3 Tillämpning av programmet

Det är inte möjligt att säga något om vilken skala FPM är tillämpbar i utifrån de resultat vi fått. Det som däremot står klart är att det är möjligt att skapa en programstruktur som mycket effektivt kan implementera FPM och köra stora mängder partiklar under relativt många tidssteg.

Genom att kallibrera parametrarna så som finns beskrivet i kapitel 2.2.1 kan programmet även användas för att undersöka var FPM har sina användningsområden.

5.2.4 Tolkning av flödessimulering

Hastighetsprofilen i figur 10 har en kvadratisk form vilket även den analytiska lösningen av Navier-Stokes ekvationer har, se (43). Detta visar att implementeringen fungerar som tänkt och att modellen beskriver korrekt beteende hos fluiden. Simuleringen i avsnitt 4.2 är dock inte gjord med kalibrerade parametrar och beskriver därför inte ett flöde med korrekt hastighet.

Simuleringen visar även att de periodiska randvillkoren fungerar och att jämvikt kan uppnås. Detta är viktigt för att kunna kallibrera systemet.

Även om den dissipativa kraften är uppskalad visar simuleringsresultaten att den orsakar friktion mellan partiklarna. Detta är en viktig observation och en verifiering på att den är implementerad rätt.

Det visar sig också att partikeltätheten varierar med upp till 10%, vilket inte är ett korrekt beteende för en inkompressibel vätska där densiteten skall vara konstant. Det visar sig att om den konservativa, repulsiva, kraftens storlek ökas så blir tätheten jämnare. Detta indikerar att den konservativa kraften fyller sin funktion och orsakar repulsiva krafter mellan partiklarna. Densiteten finns illustrerad figur 10.

Den stokastiska kraftens funktion har verifierats genom att simulera ett initialt stillastående system och sedan observera att fluiden blandas om. Det är svårare att säga något om den rotationsberoende kraftens inverkan, detta lämnas åt framtida arbeten att undersöka.

6 Slutsatser

Slutsatserna innefattar för och nackdelar med FPM som framkommit under arbetets gång. Svårigheter i implementeringen och hur dessa kan lösas samt resultaten av våra lösningar summeras. Även ideér på framtida arbeten för att komma vidare i utvecklingen tas upp.

FPM Utifrån de resultat vi fått är det inte möjligt att säga något om skalan som FPM är tillämpbar på. För att ta reda på detta måste partikeltätheten som krävs för att simulera systemet bestämmas. Utifrån detta och den tillgängliga beräkningskraften kan sedan systemets storlek bestämmas. En sådan undersökning skulle ge en inblick i hur tolkningen av fluiden som kluster av molekyler speglar verkligheten.

Skalfaktorerna som är nödvändiga för att beskriva krafterna går inte att bestämma på ett korrekt sätt utifrån de makroskopiska tryck och viskositet. Detta innebär en klar nackdel då parametrarna istället måste bestämmas genom upprepade simuleringar som jämförs med ett referensfall.

Slutsatser från flödessimulering Utifrån det enkla fall som simulerats i avsnitt 4.2 kan flera slutsatser dras. Den kanske viktigaste är att programmet gör vad det är designat för, nämligen simulerar ett flöde av partiklar med friktions och täthetsbevarande krafter.

Det är också tydligt att det periodiska randvillkoret resulterar i en jämvikt. Detta innebär att programmet kan användas för att kallibrera parametrarna och sedan modifieras för att simulera mer komplicerade flöden.

Implementering Att det optimala antalet boxar är större än en enda box är ett viktigt resultat. Utifrån detta kan slutsatsen dras att den beräkningskraft som krävs för att dela upp systemet i boxar, indexera, sortera och styra interaktionerna mellan boxar är mindre än att kontrollera avståndsvillkoret mellan alla partiklar.

Implementeringen av FPM är inte trivial och kräver en väl strukturerad arbetsgång. Om ett stort (mer än en miljon) antal partiklar ska simuleras är det nödvändigt att på något sätt begränsa antalet interaktioner mellan partiklarna. Till exempel genom boxindelning av beräkningsdomänen.

Framtida arbeten med FPM Arkitekturen hos moderna superdatorer innebär att kommunikation mellan processorer måste ske manuellt, detta sker med MPI-syntax i de flesta fall. Framtida arbeten innefattar därför utveckligen av en MPI-baserad programvara för att använda FPM på större processorkluster.

Testkörningar för att bestämma skalfaktorerna måste även göras mot ett referensfall. Detta tillsammans med ytterligare simuleringar skulle ge en klarare bild över FPM-metodens fysikaliska relevans.

Boxindelningen men även parallelliseringen har inneburit svårigheter vad gäller implementeringen. Specifikt problematiken med reduktionsvariabler kräver tillförsikt och gör utvecklingen svårare. Genom att utveckla framtida programvara i Fortran skulle denna komplikation undvikas då Fortran stödjer vektorreduktion.

Referenser

[1] Español, P. (1998) Fluid particle model. *Physical Review E*, vol. 57, nr. 3, s. 2930 - 2934.

[2] Boryczko, K. Dzwinel, W. Yuen, D. A. (2002) Parallel Implementation of the Fluid Particle Model for Simulating Complex Fluids in the Mesoscale. *Concurrency and Computation: Practice and Experience*, vol. 14, nr. 2, s. 137 - 161.

[3] Dzwinel, W. Yuen, D. A. (2001) Dispersion of Colloidal Agglomerate in Mesoscale modelled by a hybrid Fluid Particle Model. *Task Quarterly*, vol. 5, nr. 3, s. 355 - 371.

[4] Hoogerbrugge, P. J. Koelman, J. M. V. A. (1992) Simulating Microscopic Hydrodynamic Phenomena with Dissipative Particle Dynamics. *Europhysics Letters*, vol. 19, nr. 3, s. 155 - 160.

[5] Hairer, E. Lubich, C. Wanner, G. (2003) Geometric numerical integration illustrated by the Störmer/Verlet method. *Acta Numerica*, vol. 12, s. 399 - 450.

[6] Timberlake, T. Hasbun, J. E. (2008) Computation in classical mechanics. *American Journal of Physics*, vol. 76, nr. 4, s. 334 - 339.

[7] Backer, J. A. Hoefsloot, C. J. Ledema, P. D. Lowe, C. P. (2005) Poiseuille flow to measure the viscosity of particle model fluids. *The Journal of Chemical Physics*, vol. 122, nr. 15.

[8] Karniadakis G. E. Fedosov D. A. (2009) Triple-Decker: Interfacing atomistic-mesoscopic-continuum flow regimes. *Journal of Computational Physics*, vol. 228, nr. 4, s. 1157-1171.

[9] Espanõl, P. Serrano, M. (1999) Dynamical regimes in DPD. *Physical Review E*, vol. 59, nr. 6, s. 6340 - 6347.

[10] Español, P. Serrano, M. Zuñiga, I. (1997) Coarse-Graining of a fliud and its relation with Dissipative Particle Dynamics and Smoothed Particle Dynamics. *International Journal of Modern Physics C*, vol. 8, nr. 4, s. 899 - 908.

[11] Meriam, J. L. Kraige, L. G. (2008) *Engineering Mechanics: Dynamics*. upplaga 6. New York: John Wiley and Sons Inc.

[12] Cole, R. (1988) Parallel Merge Sort. *Society for Industrial and Applied Mathematics*, vol. 17, nr. 4, s. 770 - 785.

[13] White, F. M. (2011) *Fluid Mechanics*. upplaga 7. New York: Mcgraw Hill Higher Education.

[14] Rapaport, D. C.(2004) *The Art of Molecular Dynamics Simulation*. [Elektronisk] Upplaga 2. Cambridge: Cambridge University Press.

7 Bilaga

Parallelisering och boxinteraktioner

```
omp_set_num_threads(nThr);
#pragma omp parallel shared(nThr, FX, FY, deltaRot) private(i_am)
{
for(i=0; i<N*N; i++){
*(FX+i)=0;
*(FY+i)=0;
*(deltaRot+i)=0;
}
#pragma omp for schedule(static, 10)
for(i=0;i<Msq;i++){         /* Loopar över alla beräkningsboxar */
if(i%M == 0){
if(i==0){           /* Första boxens, angränsar till tre andra boxar */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,0,0);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,0,1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,0,M);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,0,M+1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,xAxis,boxL,M,FX,FY,i,i+2*M-1);
}
else if(i == M*(M-1) ){ /* Boxen längst ner till vänster */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+1);
}
else{   /* Någon av de övriga boxarna på vänsterkanten */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M+1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,-xAxis,boxL,M,FX,FY,i,i+2*M-1);
}
}
else if(i% M == M-1){   /* Boxen ligger på högersidan */
if(i== (M-1) ){     /* Boxen ligger längst upp till höger */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,xAxis,boxL,M,FX,FY,i,0);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,xAxis,boxL,M,FX,FY,i,i+1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M);
}
else if(i == (Msq-1) ){  /* Boxen ligger längst ner till höger */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,xAxis,boxL,M,FX,FY,i,i-M+1);
}
else{   /* Någon av de övriga boxarna på högerkanten */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M-1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,xAxis,boxL,M,FX,FY,i,i-M+1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,xAxis,boxL,M,FX,FY,i,i+1);
}
}
else if(i< (M-1) && i>0 ){  /* Boxen ligger på övre raden men tillhör inte hörnen */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+1);
```

```
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M-1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M+1);

}
else if(i>(M-1)*M && i<(Msq-1) ){   /* Boxen ligger på nederkanten men tillhär inte hörnen */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+1);
}
else{       /* Boxen ligger inte på kanterna utan i mitten */
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M-1);
force2(Px,Py,Vx,Vy,Mass,Rot,N,deltaT,cutoff,0,boxL,M,FX,FY,i,i+M+1);
        }
    }
}
}
```

Euler-Cromers metod

```
for(i = 0; i < Nsq; i++){
    if(*(Mass + i) ==  mass1 ){
    *(Vx+i) = 0; *(Vy+i) = 0; *(Rot+i)=0;
    }
else{
            *(Vx+i) += (G+*(FX+i)/(*(Mass+i)))*deltaT;
            *(Vy+i) += (*(FY+i)/ (*(Mass+i)))*deltaT;
    }
            *(Px+i) +=  *(Vx+i) * deltaT;
            *(Py+i) +=  *(Vy+i) * deltaT;
            *(Rot+i) += *(deltaRot+i);
}
```