Akademia Górniczo-Hutnicza im. Stanisława Staszica
w Krakowie
Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Katedra Informatyki

Rozprawa doktorska

Algorytmy symulacji płynów rzeczywistych bazujące na modelach DPD oraz SPH dla komputerów z pamięcią współdzieloną oraz rozproszoną

mgr inż. Paweł Wróblewski

Promotor: dr. hab inż. Krzysztof Boryczko

Styczeń 2009

Podziękowania

Chciałbym bardzo serdecznie podziękować wszystkim osobom, które przyczyniły się do powstania niniejszej pracy. Bez ich pomocy i wsparcia ukończenie jej nie byłoby możliwe. W szczególności pragnę podziękować:

- Panu Profesorowi dr hab. inż. Krzysztofowi Boryczko za merytoryczną opiekę, wyrozumiałość, jak również cenne wskazówki i zachęty, bez których niniejsza praca nie mogłaby powstać,
- Panu dr inż. Mariuszowi Kopciowi za liczne dyskusje i współpracę podczas implementacji nowoczesnych metod cząstek,
- Pani Jolancie Lepiarczyk za cenne rady, wsparcie oraz pomoc okazaną w ramach studiów doktoranckich,
- kolegom Tomaszowi Arodziowi oraz Marcinowi Kurdzielowi za liczne dyskusje, spotkania oraz za wspólny czas spędzony na studiach doktoranckich.

Przede wszystkim pragnę podziękować moim rodzicom. To dzięki ich wsparciu, okazywanej pomocy oraz cierpliwości mogłem skupić się na pracy nad niniejszą rozprawą.

Niniejsza rozprawa była częściowo finansowana przez grant Ministerstwa Nauki i Szkolnictwa Wyższego nr 3 T11F 010 30.

Spis treści

Spis rysunków

Rozdział 1

Wprowadzenie

1.1 Podstawy i zakres pracy

Niemal jednocześnie z powstaniem komputerów pojawiły się pierwsze modele i algorytmy pozwalające na ich wykorzystanie do symulowania zjawisk zachodzących w świecie rzeczywistym. Można uznać, że pierwsze symulacje wykonano już w latach czterdziestych XX wieku w ramach projektu Manhattan, kiedy Jonh von Neumann i Stanisław Ulam wykorzystali komputer do modelowania rozpraszania i absorbcji neutronów, tworząc w ten sposób metodę Monte Carlo [95, 43]. Na przestrzeni kilkudziesięciu lat stworzono metody symulacji modelujące zjawiska z ogromnego zakresu dziedzin, od skal astronomicznych [38] do skali kwantowej [67], od temperatur bliskich zeru bezwzględnemu [50] do osiągających dziesiątki i setki tysięcy kelwinów [12]. Dodatkowo, istnieją metody modelujące zachowanie zwierząt w grupie [40], działanie biologicznych układów nerwowych [6], mechanizmy giełdowe [7] i wiele innych złożonych systemów [120, 88], kończąc na możliwości modelowania tak tajemniczych zjawisk, jak powstanie we wszechświecie obserwowanej czterowymiarowej czasoprzestrzeni [3].

Wiele z wymienionych wyżej zjawisk, istotnych zarówno z naukowego punktu widzenia jak i dla zastosowań przemysłowych, dotyczy mechaniki płynów. Zjawiska te zachodzą w różnych skalach przestrzenno-czasowych, poczynając od skali mikro, poprzez skalę mezo, aż do skali makro [83]. Dodatkowo, wiele z tych interesujących zjawisk, takich jak np. powstawanie przepływu turbulentnego, oddziaływania pomiędzy płynem a ściankami naczynia, zachowanie się powierzchni swobodnej płynu czy przepływy wielofazowe, zachodzi na styku poszczególnych skal. Niestety nie są jeszcze dostępne poprawne i efektywne metody pozwalające na symulację wielu z tych zjawisk, zwłaszcza gdy nacisk jest kładziony na spójność termodynamiczną stosowanego modelu [125, 48].

Jedną z technik symulacji przepływów i zachowania się płynów w różnych warunkach są tzw. metody cząstek. Jest to opis płynu zgodnie z podejściem lagranżowskim, w którym zmienne opisujące płyn obliczane są w punktach poruszających się razem z przepływającym płynem [134]. Opis lagranżowski stoi w opozycji do opisu eulerowskiego, w którym zmienne opisujące płyn obliczane są w stałych, nie zmieniających położenia punktach. W metodach cząstek pod pojęciem „cząstki" rozumie się pewną objętość, ilość płynu, z której środkiem masy są skojarzone pewne parametry fizyczne. Zbiór cząstek reprezentujących interesujący fragment materii tworzy układ zamknięty w umownym, izolowanym od świata zewnętrznego pudle obliczeniowym. Symulacja polega na obserwacji parametrów ewoluującego układu cząstek od pewnego stanu początkowego przez zadaną z góry liczbę kroków czasowych. Modelowanie płynu odbywa się za pomocą obliczania trajektorii dużej liczby cząstek oddziałujących ze sobą.

W wielu metodach cząstek występuje siatka, która określa geometryczne własności są-

siedztwa pomiędzy cząstkami, bądź też narzuca więzy na ruch cząstek. Bezsiatkowe metody cząstek są tego pozbawione. W każdym kroku symulacji obliczane są siły oddziaływań i na ich podstawie rozwiązywane są równania ruchu, dzięki czemu wyznaczane są położenia i prędkości cząstek w kolejnym kroku symulacji. Nie istnieje żadna siatka, która definiowałaby relację sąsiedztwa pomiędzy cząstkami, lub zmuszała cząstki do poruszania się w wybranych, zadanych kierunkach.

Jednym z najbardziej złożonych obliczeniowo fragmentów symulacji jest obliczanie sił oddziałujących na każdą cząstkę, a pochodzących od pozostałych cząstek układu. W zależności od zastosowanego potencjału oddziaływania cząstek stosuje się kilka metod wyznaczania sąsiadów. W przypadku potencjału krótkozasięgowego stosuje się tak zwany promień obcięcia polegający na tym, iż uwzględnia się oddziaływania pochodzące jedynie od cząstek zawartych w kuli o promieniu równym promieniowi obcięcia i środku w danej cząstce. Dla tak wprowadzonego uproszczenia istnieje kilka efektywnych algorytmów wyszukiwania sąsiadów [118].

Symulowanie złożonych układów z wykorzystaniem metod cząstek wymaga stosowania systemów komputerowych o dużych mocach obliczeniowych. Dodatkowo, duża złożoność pamięciowa wymusza, aby systemy te posiadały odpowiednio duże zasoby pamięci operacyjnej. Fakt, że obliczenia składają się z przeprowadzania tych samych operacji dla dużej ilości identycznych obiektów, powoduje, że w celu efektywnej realizacji symulacji właściwa jest implementacja równoległa. Stąd powszechne wykorzystanie architektur wieloprocesorowych z pamięcią rozproszoną lub współdzieloną [81, 36]. Pierwsze charakteryzują się relatywnie niską ceną, lecz dominujący na nich paradygmat programowania oparty na przesyłaniu komunikatów wymaga dużego doświadczenia programistycznego i nakładu czasowego. Drugie z wymienionych architektur są obecnie stosunkowo drogie, lecz dostępne środowiska programistyczne pozwalają osiągać porównywalne wartości parametrów obliczenia równoległego (przyspieszenie, efektywność) przy znacznie mniejszym nakładzie pracy. W obydwu podejściach kluczowy jest efektywny algorytm symulacji, gdzie przez efektywność rozumie się wykorzystanie do modelowanego zjawiska jak najmniejszych zasobów obliczeniowych, co dotyczy zarówno wykorzystanej pamięci operacyjnej jak i czasu obliczeń.

Modelowanie płynów rzeczywistych za pomocą symulacji komputerowych jest bardzo użyteczne w wielu dziedzinach współczesnej nauki. Poczynając od symulacji przemysłowych [33], poprzez medycynę [20], na wybuchach termojądrowych skończywszy [89, 82]. Poznanie zachowania się płynu, zwłaszcza złożonego, jest pożądane z punktu widzenia wielu zastosowań technicznych, biologicznych, farmaceutycznych. Jako przykłady można podać modelowanie spalania gazu w silniku w warunkach wysokotemparaturowych [111], czy modelowanie rozprowadzania leków w organizmie ludzkim [127]. Są to dziedziny, w których trudno jest badać dokładnie ciecz na drodze empirycznej. Dlatego duży nacisk kładzie się na rozwój technik symulacji. W niniejszej pracy przedstawiono kilka możliwych zastosowań metod symulacji płynów przy pomocy cząstek. Wynika z nich, że jest to obiecująca metoda, dająca wymierne i wartościowe rezultaty.

1.2 Cele pracy

Nowoczesne modele płynów uwzględniają złożony charakter symulowanych układów. W szczególności dotyczy to problemów z właściwym symulowaniem wielkości termodynamicznych modelowanego układu, odpowiednią realizacją przepływów w naczyniach o złożonych kształtach, modelowania zjawisk zachodzących w różnych skalach przestrzenno-czasowych. Nowoczesne modele płynów, umożliwiające symulację tych zagadnień, charakteryzują się dużym stopniem skomplikowania, co przekłada się na trudniejszą realizację ich implementacji. Uzyskanie akceptowalnych czasów obliczeń wymaga użycia nowoczesnych architektur

komputerowych i stosowania na nich efektywnych, dopasowanych do nich, algorytmów. Dotyczy to w szczególności implementacji równoległej. Jednym z celów niniejszej pracy jest właśnie zaproponowanie efektywnych implementacji takich algorytmów.

Jednym z możliwych kryteriów, według których można podzielić architekturę komputerową, jest sposób dostępu pamięci operacyjnej. Współczesne wieloprocesorowe architektury komputerowe bazują na modelu z pamięcią współdzieloną lub na modelu z pamięcią rozproszoną. W pierwszym z tych modeli dominującym środowiskiem programowania jest środowisko OpenMP. Powszechnie uważa się, iż nakład pracy programistycznej koniecznej do uzyskania względnie efektywnego algorytmu jest w tym przypadku niewielki. Niestety, uzyskiwane wartości wskaźników obliczenia równoległego w wielu przypadkach nie pokrywają się z oczekiwaniami. W modelu z pamięcią rozproszoną dominuje paradygmat programowania oparty o przesyłanie komunikatów. Jest on realizowany w jednym z wielu dostępnych środowisk programowania (MPI, PVM). Znajduje się on w pewnej opozycji w stosunku do wspomnianego wyżej modelu z pamięcią współdzieloną. Wymaga dużego nakładu pracy programistycznej. Pozwala jednak na osiąganie wyśrubowanych wartości współczynników obliczenia równoległego [84, 77]. W wielu przypadkach stwierdzenie, który z modeli będzie właściwy dla rozwiązywanego problemu nie jest możliwe wprost. Stąd celem tej pracy jest również określenie, który z wspomnianych modeli jest bardziej efektywny w zastosowaniu do modelowania płynów złożonych przy pomocy metody cząstek.

Powszechnie implementowane dotychczas modele znajdują zastosowanie w wielu dziedzinach nauki i techniki. W wielu przypadkach są to modele uproszczone. Ich implementacja jest prosta, czas wykonania akceptowalny. Wzrost mocy obliczeniowej współczesnych architektur komputerowych umożliwił implementacje znacznie bardziej złożonych modeli uwzględniających istotnie więcej właściwości modelowanych układów. Umożliwia to modelowanie różnych modeli lepkości, uwzględnianie nieizotermicznego charakteru symulowanego układu, uwzglednianie bądź pomijanie efektów wynikających z mikroskopowych własności układów. Oczekuje się, że tak zmodyfikowane modele pozwolą rozwiązać istniejące trudności w problemach dotyczących symulacji przepływów płynów nie-newtownowskich bądź modelowania zachowania płynów uwzględniające różne skale przestrzenno-czasowe. W pracy szczególną uwagę zwrócono na zagadnienia implementacji nowoczesnych modeli i ich zastosowanie do wymienionych wyżej problemów.

Głównym celem niniejszej rozprawy jest zaproponowanie efektywnej implementacji równoległej nowoczesnych modeli symulacji płynów, takich jak SPH, DPD oraz SDPD. Przedstawione w niniejszej pracy rezultaty można podzielić na dwie grupy. W części algorytmicznej, należą do nich:

- zaproponowanie, w oparciu o istniejące metody, nowych algorytmów implementacji równoległej metody cząstek,
- zaproponowanie, na podstawie porównania wskaźników obliczenia równoległego dla implementacji z wykorzystaniem środowisk OpenMP oraz MPI, ogólnych przesłanek dotyczących metod i sposobów zrównoleglania dla algorytmów symulacji metodą cząstek wykorzystujących paradygmaty programowania dla komputerów z pamięcią współdzieloną i rozproszoną,
- zaproponowanie metod dynamicznego podziału pudła obliczeniowego w celu równoważenia obciążenia na poddomeny dla algorytmów równoległych wykorzystujących środowiska programowania pracujące z wykorzystaniem przesyłania komunikatów.

W części dotyczącej zastosowań w dziedzinie symulacji komputerowych należą do nich:

- efektywna realizacja teoretycznych modeli nieizotermicznych, a w szczególności modelu SDPD, dla symulacji metodą czastek,

- zaproponowanie oraz weryfikacja nowego modelu lepkości w metodzie SPH do modelowania płynów o charakterze nie-newtonowskim,
- zaproponowanie modyfikacji metody SPH umożliwiającej symulowanie napięcia powierzchniowego cieczy,
- wykorzystanie modeli nieizotermicznych do modelowania zjawisk nierównowagowych, takich jak na przykład przepływ ciepła.

Przedstawione powyżej zagadnienia oraz metody ich rozwiązania zostały tak sformułowane, aby wykazać ogólną tezę niniejszej rozprawy doktorskiej, która brzmi następująco:

Nowoczesne teoretyczne modele płynów mogą zostać efektywnie zaimplementowane z wykorzystaniem paradygmatów programowania dla komputerów z pamięcią współdzieloną oraz rozproszoną. Tak opracowane algorytmy mogą służyć symulacji zjawisk złożonych zachodzących w różnych skalach przestrzenno-czasowych.

1.3 Streszczenie

Niniejsza praca składa się z pięciu podstawowych rozdziałów. W rozdziale drugim przedstawione zostały podstawy algorytmiczne symulacji metodami cząstek. Omówiono w nim podstawowe pojęcia i definicje, strukturę algorytmu symulacji oraz kwestie dotyczące kształtu pudła obliczeniowego oraz warunków brzegowych.

W rozdziale trzecim zaprezentowane zostały metody cząstek powszechnie używane w symulacjach. Opis rozpoczyna się od zaprezentowania podstawowej metody wykorzystującej cząstki, to jest od dynamiki molekularnej, będącej podstawą dla innych metod. W dalszej kolejności omówiono metody operujące w skali mezoskopowej, jednak nie spełniające warunku spójności termodynamicznej. Następnie zaprezentowano model GENERIC, dzięki któremu możliwe jest spójne termodynamicznie ujęcie modelowanego płynu w metodach cząstek. Omówiono metodę SDPD. W końcowej części rozdziału dokonano przeglądu schematów całkownia równań ruchu oraz przedyskutowano zakresy ich zastosowań.

W rozdziale czwartym przedstawiono podstawowe aspekty implementacji sekwencyjnej dla wybranych metod cząstek. Zaprezentowano szczegółowy opis pętli głównej algorytmu. Dodatkowo, przedstawiono sposoby doboru funkcji ważącej obecnej w metodach mezo i makroskopowych, sposób wyznaczania wielkości kroku czasowego w symulacji, jak również kwestie związane z tworzeniem konfiguracji początkowej. W dalszej części rozdziału porównano dwie definicje sąsiedztwa pomiędzy cząstkami na przykładzie metody SPH oraz przeanalizowano ich przydatność pod kątem modelowania zjawisk i przepływów płynów nieściśliwych.

Rozdział piąty przedstawia aspekty związane z implementacją równoległą metod cząstek. W części pierwszej omówione zostały wskaźniki obliczeń równoległych wykorzystywane do oceny efektywności implementacji równoległych. W dalszej części szczegółowo opisane oraz porównane zostały implementacje przeznaczone na architektury z pamięcią współdzieloną (środowisko OpenMP) oraz na architektury z pamięcią rozproszoną (środowisko MPI). Kolejno, przedstawiony został sposób optymalizacji algorytmu wykorzystującego paradygmat przesyłania komunikatów oraz jego dostosowanie do metod cząstek, w których wartości sił oddziaływania zależą od zmiennych losowych. W końcowej części rozdziału zaprezentowano porównanie efektywności obydwu algorytmów.

Rozdział szósty zawiera przykłady zastosowań przedstawionych implementacji do modelowania płynów rzeczywistych. Zaprezentowano w nim także propozycje modyfikacji klasycznych modeli pozwalające na modelowanie zjawiska napięcia powierzchniowego oraz symulacji przepływów płynów nie-newtonowskich. W dalszej części zaprezentowano wyniki symulacji z wykorzystaniem metody SDPD.

Rozdział 2

Podstawy algorytmiczne

Algorytmy symulacji płynów prezentowane w niniejszej pracy opierają się na bezsiatkowych metodach cząstek. Oddziaływanie cząstek płynu w rozpatrywanych modelach ma charakter krótkozasięgowy, co determinuje algorytmy wyszukiwania sąsiadów. W rozdziale przedstawiono podstawowe zagadnienia symulacji metodą oddziałujących cząstek. Omówiono kolejne kroki algorytmu symulacji dla metody cząstek począwszy od budowy konfiguracji początkowej, przez obliczanie oddziaływań i rozwiązywanie równań ruchu, na wyznaczaniu wartości parametrów makroskopowych skończywszy.

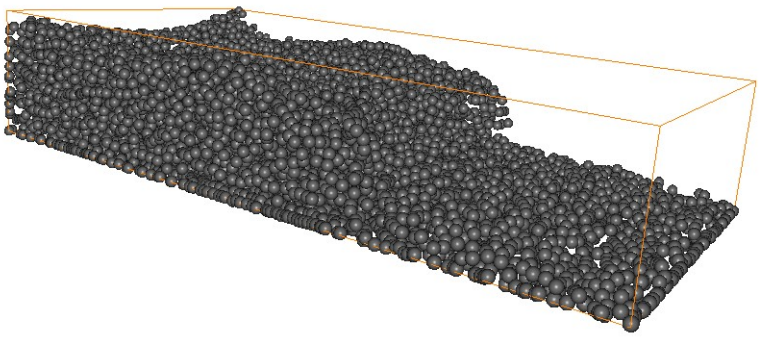
2.1 Metody cząstek

W metodach cząstek modelowany przez nie płyn reprezentowany jest przez zbiór punktów. Każdemu z nich przypisany jest zestaw parametrów, które określają jego właściwości w rozpatrywanym modelu. Jako przykład parametrów można wymienić: położenie, masę, objętość, czy prędkość. Punkt, wraz z opisującymi go parametrami, nazywany jest cząstką. Zestaw parametrów cząstek zależy od modelu wykorzystywanego w symulacji i może się zmieniać od kilku do kilkunastu elementów. Parametry cząstek wykorzystywane w symulacji określają przede wszystkim oddziaływania pomiędzy nimi.

Metody cząstek są typowym przykładem problemu oddziałujących N ciał. Modelowanie płynu polega na obliczaniu oddziaływań pomiędzy cząstkami i na tej podstawie na śledzeniu ich trajektorii oraz prędkości. Istnieje wiele metod cząstek, z których każdą wyróżnia pewien zestaw cech predestynujący ją do konkretnego typu zastosowań. Z oczywistych powodów trudno jest wymienić wszystkie istniejące metody.

Przeglądu metod cząstek można dokonać ze względu na skalę przestrzenno-czasową, w której znajdują one zastosowanie. W skali mikroskopowej dostępne są takie metody, jak dynamika molekularna (MD) [68], nierównowagowa dynamika molekularna (NEMD) [74], metoda Monte Carlo (MC) [51], bądź też bezpośrednie symulacje Monte Carlo (DSMC) [110]. W skali mezoskopowej dostępne są m.in. gaz siatkowy (LG) oraz gaz siatkowy Boltzmanna (LBG) [136], dynamika brownowska (BD) [119], dyssypatywna dynamika cząstek (DPD) [47], model cząstek płynu (FPM) [46], wygładzona dyssypatywna dynamika cząstek (SDPD) [48] czy spójna termodynamicznie dyssypatywna dynamika cząstek (TC-DPD) [125]. Z metod dostępnych w skali makroskopowej można wymienić metodę cząstki w komórce (PIC) [63], metody wierzchołkowe (Vortex Methods) [35] czy też wygładzoną hydrodynamikę cząstek (SPH) [99].

Podziału powyższych metod można dokonać również pod innym kątem. Część z nich wykorzystuje siatkę, która wypełnia modelowany obszar. Siatka ta służy bądź do wyznaczania kierunków, w których mogą poruszać się cząstki, albo jej węzły stanowią punkty, w których oblicza się wielkości związane z przepływem, bądź też dzieli ona zbiór cząstek



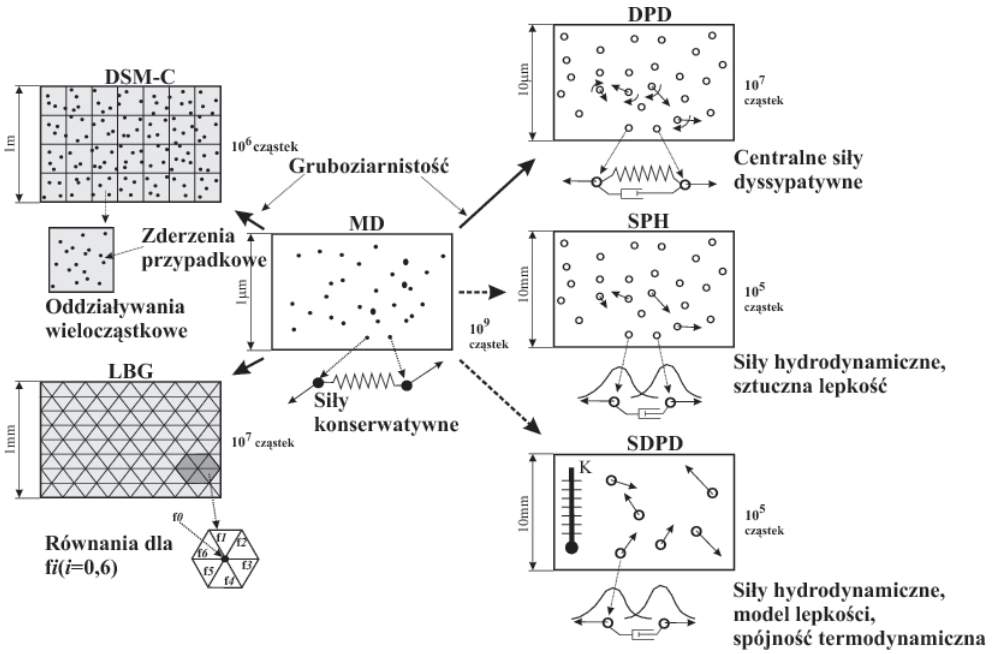
Rysunek 2.1: Modelowany płyn reprezentowany przez zbiór cząstek.

na podzbiory cząstek ze sobą oddziałujących. W każdym z tych trzech przypadków siatka narzuca pewne więzy na ruch, prędkości bądź oddziaływania cząstek. W opozycji do metod wykorzystujących siatki stoją bezsiatkowe metody cząstek. W metodach tych nie występują żadne narzucone ograniczenia ani żadne więzy ograniczające w układzie stopnie swobody. Cząstki w tych metodach mogą poruszać się w dowolnych kierunkach, a oddziaływania pomiędzy nimi zależą jedynie od ich parametrów i odległości pomiędzy nimi. Z wyżej wymienionych metod do zbioru bezsiatkowych metod cząstek zalicza się MD, NEMD, BD, DPD, FPM, SDPD, TC-DPD oraz SPH.

W niniejszej pracy rozpatrywane są jedynie bezsiatkowe metody cząstek. W metodach tych modelowany płyn reprezentowany jest tylko i wyłącznie przez cząstki. Przykładowy obraz modelowanego płynu przedstawiono na rysunku (2.1). Pomiędzy każdą parą cząstek działają siły określone przez przyjęty model oddziaływań. Symulacja jest ciągiem iteracji, gdzie w każdej z nich dla każdej pary cząstek obliczane są siły, z jakimi one oddziałują. Następnie dla każdej cząstki, na podstawie działającej na niej siły wypadkowej, obliczane jest przyspieszenie, jakiemu cząstka podlega w danej iteracji. Na tej podstawie obliczana jest poprawka do prędkości cząstki. Ostatnią czynnością w bieżącym kroku symulacji jest obliczenie nowego położenia cząstki na podstawie dotychczasowego położenia i obliczonej prędkości.

Historycznie pierwszą, bezsiatkową metodą cząstek była dynamika molekularna. Pomimo, iż kolejno powstałe metody różnią się od niej zarówno charakterem oddziaływań pomiędzy cząstkami jak i skalą przestrzenno-czasową, w której operują, koncepcyjnie są one do niej bardzo podobne. Dlatego też cała zgromadzona wiedza dotycząca szczegółów implementacyjnych dynamiki molekularnej mogła zostać wykorzystana przy opracowywaniu kolejnych metod. Schematycznie, zależność pomiędzy metodami przedstawiono na rysunku (2.2). Poszczególne bezsiatkowe metody cząstek zostały bliżej omówione w rozdziale 3.

Z punktu widzenia implementacji bezsiatkowe metody cząstek rozpatrywane w niniejszej pracy składają się z dwóch składników. Są to: model oddziaływania pomiędzy cząstkami, który określa charakter modelowanego płynu, oraz schemat całkowania, od którego zależy sposób obliczania nowych położeń i prędkości cząsteczek. Schemat całkowania powinien



Rysunek 2.2: Zależności i podstawowe różnice pomiędzy głównymi metodami cząstek.

zapewniać stabilność symulacji oraz uwzględniać deterministyczny bądź stochastyczny charakter oddziaływań międzycząstkowych.

2.2 Potencjały krótkozasięgowe

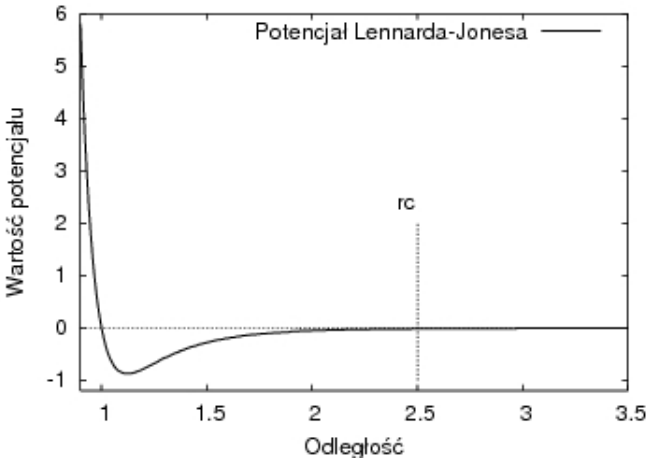
W metodach cząstek oddziaływania pomiędzy cząstkami najczęściej zadaje się w postaci potencjału. Potencjały wielociałowe, właściwe do modelowania zjawisk subatomowych i właściwości molekularnych [39, 117], nie znajdują zastosowania w symulacjach płynów w skalach mikroskopowej i od niej większych. W obszarze tym stosuje się najczęściej potencjały dwuciałowe.

W zależności od tego, jak potencjał zachowuje się wraz ze wzrostem odległości pomiędzy cząstkami r, rózróżniamy jego dwa rodzaje. Jeśli wartość potencjału maleje szybciej niż r^{-D}, gdzie D jest wymiarem problemu, to potencjał klasyfikuje się jako krótkozasięgowy. W przeciwnym przypadku potencjał określa się jako długozasięgowy [68].

Podział ten staje się jasny, jeśli rozpatrzy się całkę:

$$I = \int_{x_0>0}^{\infty} r^{-n} dr^D = \begin{cases} \infty, & \text{jeśli } n \leqslant D, \\ C < \infty, & \text{w przeciwnym przypadku.} \end{cases} \tag{2.1}$$

Wynika z niej, że w przypadku gdy $n \leqslant D$ do energii potencjalnej dowolnej cząstki dają wkład wszystkie cząstki z układu. W przeciwnym przypadku, ponieważ energia potencjalna danej cząstki jest skończona, można ograniczyć cząstki dające wkład do energii potencjalnej do tych znajdujących się w najbliższym sąsiedztwie rozpatrywanej cząstki. W ten sposób wprowadza się do symulacji promień obcięcia r_{cut}. Jeśli odległość pomiędzy dwoma cząstkami jest większa od r_{cut}, wówczas ich wzajemne oddziaływanie uznaje się za nieistotne. Jako przykład może posłużyć krótkozasięgowy potencjał Lennarda-Jonesa. Dla odległości większych niż zaznaczony promień obcięcia r_{cut} przyjmuje się, że potencjał jest zerowy, co zaznaczono na rysunku (2.3). Aby usunąć powstałą w ten sposób nieciągłość, potencjał poddaje się modyfikacji. Najczęściej polega ona na dodaniu do potencjału stałej wartości



Rysunek 2.3: Potencjał Lennarda-Jonesa wraz z zaznaczonym promieniem obcięcia.

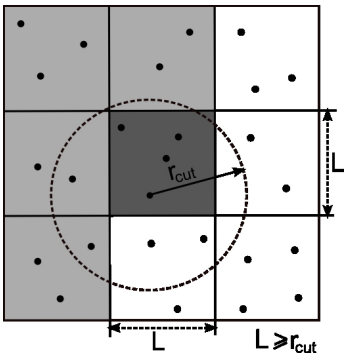
takiej, aby dla odległości równej promieniowi obcięcia wartość potencjału wynosiła zero. Można również zastosować tzw. poprawkę długozasięgową [85].

2.3 Organizacja pudła obliczeniowego

W bezsiatkowych metodach cząstek, w których oddziaływania mają charakter dwuciałowy, złożoność obliczeniowa algorytmu symulacji jest rzędu $O(N^2)$, gdzie N to ilość cząstek w układzie. Wynika to z liczby par cząstek w układzie, dla których trzeba przeprowadzić odpowiednie obliczenia. Przykładowo, dla każdej pary cząstek w układzie należy obliczyć odległość pomiędzy nimi, aby sprawdzić czy i ewentualnie z jaką siłą cząstki ze sobą oddziałują.

Wprowadzenie promienia obcięcia pozwala na zmniejszenie złożoności obliczeniowej algorytmu symulacji. Dokonuje się tego poprzez podział pudła obliczeniowego sześcienną (w przypadku trójwymiarowym) siatką. Długość boku komórki elementarnej siatki jest nie mniejsza niż największy promień obcięcia wystepujący w symulacji. Dzięki temu warunkowi znacząco ogranicza się liczbę potencjalnych sąsiadów (cząstek sąsiednich) dla każdej cząstki. Aby znaleźć sąsiadów dla cząstki znajdującej się w celi o numerze μ, wystarczy obliczyć jej odległość od innych cząstek znajdujących się w celi μ oraz od cząstek z cel sąsiednich do celi μ. Przedstawiono to na rysunku (2.4). Oddziaływania oblicza się jedynie dla sąsiadów znajdujących się w sferze o środku w danej cząstce i promieniu równym promieniowi obcięcia. Wprowadzenie podziału pudła obliczeniowego sześcienną siatką nie zmienia charakteru metody z bezsiatkowego na siatkowy. W przeciwieństwie do metod siatkowych wprowadzona siatka ma charakter wyłącznie implementacyjny i służy jedynie poprawieniu złożoności obliczeniowej algorytmu. W żadnym przypadku nie wpływa ona na ruch cząstek, ich oddziaływania czy też jakiekolwiek inne fizyczne aspekty symulacji.

Aby wykorzystać strukturę sześciennych cel należy w każdym kroku symulacji przyporządkować każdą cząstkę do celi, w której się ona fizycznie znajduje. Powszechnie wykorzystywanym w tym przypadku rozwiązaniem jest implementacja cel połączonych Hockneya opisana w [72]. Dzięki przyporządkowaniu, podczas znajdowania cząstek oddziałujących z konkretną cząstką i, wystarczy obliczyć odległości do cząstek znajdujących się jedynie w 27 sąsiadujących celach (przypadek trójwymiarowy), a nie w całym pudle obliczeniowym [21]. Dzięki temu złożoność algorytmu redukuje się do $O(N)$. W niniejszej pracy wykorzystano rozwiązane równoważne liście połączonych cel Hockneya, którego szczegóły implementa-



Rysunek 2.4: Cząstki oddziałujące z cząstką o numerze i na tle wprowadzonej siatki kwadratowej.

cyjne przedstawiono w podrozdziale 4.4.

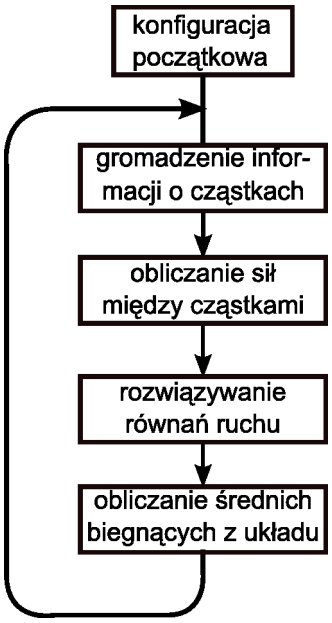
Z powodów implementacyjnych, jak również regularnego charakteru siatki, pudło obliczeniowe powinno mieć kształt prostopadłościanu. Z tego powodu pudło obliczeniowe nie może służyć do implementacji skomplikowanych kształtów naczyń, w których znajduje się modelowany płyn. Naczynia takie modeluje się, podobnie jak płyn, również z wykorzystaniem cząstek. Ścianki naczynia składają się z cząstek rozmieszczonych tak, aby oddawały ich kształt. Aby kształt naczynia nie zmieniał się podczas trwania symulacji, cząstkom ścianek nadaje się dużą (kilka rzędów wielkości większą niż dla cząstek płynu) masę, bądź pomija się rozwiązywanie dla nich równań ruchu. Ściany elastyczne symuluje się wykorzystując sąsiedztwo Moore'a, w którym cząstki oddziałuja potencjałem harmonicznym [18]. Oddziaływania pomiędzy cząstkami ścian a cząstkami płynu również opisuje się najczęściej przy pomocy potencjału. Istnieje wiele potencjałów modelujących te oddziaływania, każdy dla odpowiedniego charakteru oddziaływania pomiędzy ściankami naczynia a płynem. Przykłady tych potencjałów można znaleźć w [18, 99].

2.4 Schemat algorytmu

Struktura algorytmu symulacji jest wzglednie prosta. Na początku, przed rozpoczęciem symulacji należy wygenerować konfigurację początkową układu, który będziemy symulować. Oznacza to rozmieszczenie cząstek w odpowiednich miejscach w pudle obliczeniowym, nadanie im początkowych prędkości oraz innych wielkości je charakteryzujacych. Następnie rozpoczyna się wykonywanie zadanej liczby iteracji głównej pętli algorytmu.

Schemat algorytmu symulacji został przedstawiony na rysunku (2.5). Każda iteracja składa się z kilku kroków. W pierwszym z nich gromadzone są informacje o położeniach cząstek, na podstawie których są one przypisywane do odpowiednich cel. Następnie obliczane są oddziaływania pomiędzy cząstkami. Na podstawie obliczonych sił budowany jest układ równań ruchu Newtona, który rozwiązywany jest w kolejnym kroku algorytmu. Dzięki temu wyznaczone zostają położenia i prędkości cząstek w kolejnym kroku czasowym symulacji. Następnie wyznaczane są wartości średnich biegnących. Są one wykorzystane w odpowiednich statystykach i posłużą do obliczenia wartości parametrów makroskopowych symulowanego układu.

Kluczową kwestią jest wybór odpowiedniej metody rozwiązywania równań ruchu. Wybór ten determinuje stabilność, dokładność i efektywność całego algorytmu. Cechy charakterystyczne poszczególnych metod cząstek powodują, że dla każdej z nich właściwy może być inny schemat całkowania. Ich przegląd pod kątem wykorzystywanych w niniejszej pracy metod cząstek przeprowadzono w podrozdziale 3.6.



Rysunek 2.5: Schemat algorytmu symulacji.

2.5 Podsumowanie

W rozdziale przedstawiono podstawy algorytmiczne symulacji płynów metodami cząstek. Wprowadzono podstawowe pojęcia, podział systematyczny metod cząstek oraz przedstawiono podstawowe informacje dotyczące implementacji. Zaprezentowano kryterium pozwalające odróżnić potencjały długozasięgowe od potencjałów krótkozasięgowych. Potencjały krótkozasięgowe stanowią podstawę metod używanych w opisywanych symulacjach a ich cechy determinują metodykę implementacji algorytmów opisywanych w niniejszej pracy. Opis konkretnych metod oraz szczegóły ich implementacji zostaną rozwinięte w kolejnych rozdziałach.

Rozdział 3

Wybrane aspekty metod cząstek w symulacjach płynów

W rozdziale tym zaprezentowane zostały podstawowe bezsiatkowe metody cząstek ze szczególnym uwzględnieniem ich fizycznych podstaw. We wzorach opisujących poszczególne metody występują powtarzające się symbole i wielkości. Aby uniknąć ich wielokrotnego opisywania zostały one przedstawione poniżej:

m	masa cząstki
$\mathbf{r}_i$	wektor wodzący cząstki i
$\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$	
$r_{ij} = \lvert\mathbf{r}_{ij}\rvert$	
$\mathbf{e}_{ij} = \mathbf{r}_{ij}/r_{ij}$	wersor na osi pomiędzy cząstkami i oraz j
$\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$	różnica prędkości pomiędzy cząstkami i oraz j
$\mathrm{d}t$	krok czasowy

Opisując wybrane modele oddziaływań zwrócono uwagę na zakres ich zastosowań. Dokonano również przeglądu najczęściej stosowanych metod rozwiązywania równań ruchu oraz przeprowadzono dyskusję ich stosowalności w różnych skalach przestrzenno-czasowych.

3.1 Metoda MD

Metoda MD (Molecular Dynamics) jest najstarszą bezsiatkową metodą cząstek. Jej podstawowa wersja pojawiła się już pod koniec lat pięćdziesiątych [2]. Od tego czasu metoda była intensywnie rozwijana i znalazła szerokie zastosowanie w wielu dziedzinach nauki. Pojawiło się wiele jej wariantów dla konkretnych zastosowań. Wiedza zgromadzona podczas rozwijania tej metody posłużyła jako podstawa podczas rozwijania innych bezsiatkowych metod cząstek.

W metodzie MD cząstki odpowiadają rzeczywistym atomom lub molekułom modelowanego płynu. Ewolucja czasowa układu cząstek postępuje poprzez numeryczne rozwiązywanie równań ruchu cząstek. W dynamice molekularnej równaniem tym jest druga zasada dynamiki Newtona. Oddziaływania pomiędzy cząstkami wynikają tylko z ich własności i wzajemnych położeń, a nie zależą od żadnych zewnętrznych warunków bądź członów losowych. Dzięki temu metoda MD jest w pełni deterministyczna.

Postać oddziaływań międzycząstkowych zależy od modelowanego zjawiska. Potrzebne formuły opisujące siły dobierane są doświadczalnie bądź heurystycznie. Dość typowym oddziaływaniem wykorzystywanym w dynamice molekularnej jest potencjał Lennarda-Jonesa

6-12:

$$U_{ab}(r_{ij}) = 4\varepsilon_{ab}\left[\left(\frac{\sigma_{ab}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ab}}{r_{ij}}\right)^{6}\right], \tag{3.1}$$

gdzie indeksy a, b oznaczają typy cząstek i oraz j odpowiednio, σ_{ab} oznacza odległość pomiędzy cząstkami, dla której potencjał jest równy 0, a ε_{ab} głębokość studni potencjału. Wykres potencjału Lennarda-Jonesa został przedstawiony na rysunku (2.3). Potencjał Lennarda-Jonesa 6-12 składa sie z dwóch członów. Człon, w którym występuje zależność r^{-6}, jest członem przyciągającym wyprowadzonym dla dwóch wyindukowanych dipoli elektrycznych. Jego postać wynika z oddziaływania van der Waalsa. Natomiast człon, w którym występuje r^{-12}, został dobrany do członu r^{-6} tak, aby konieczne obliczenia analityczne miały wygodną postać. Człon ten opisuje krótkozasięgowe odpychanie pomiędzy rdzeniami (jądrami atomowymi) atomów lub molekuł [1]. Zgodnie z definicją (2.1) potencjał Lennarda-Jonesa 6-12 jest potencjałem krótkozasięgowym, dlatego też wygodnie jest w praktycznych zastosowaniach wprowadzić dla niego promień obcięcia r_{cut}. Przyjmuje się zwykle, że $r_{cut} = (2.5 \div 3.3)\sigma$ [1].

Metoda dynamiki molekularnej jest wciąż intensywnie wykorzystywaną metodą w symulacjach komputerowych. Znajduje zastosowanie w modelowaniu zjawisk związanych z lepkością i przepływem ciepła w płynach [32], powstawaniem defektów w kryształach [31, 32], modelowanie właściwości makromolekuł w systemach biologicznych (DNA, RNA, błony komórkowe) [15] lub zawiesin koloidów [78].

3.2 Metoda DPD

Metoda DPD (Dissipative Particle Dynamics) została zaproponowana przez Hoogerbrugge i Koelmana w pracy [73]. Impulsem do jej powstania była potrzeba poprawnego sformułowania modelu symulacji zjawisk zachodzących w płynach złożonych. Dotyczyło to między innymi: przepływów wielofazowych, przepływów przez media porowate, zawiesin koloidów i innych [104]. Wszystkie wymienione zjawiska zachodzą w skali mezoskopowej, czyli w wymiarach rzędu $10^{-8} \div 10^{-4}$ metra i skali czasowej rzędu $10^{-5} \div 10^{-2}$ sekundy. Skala ta nie jest dostępna dla dynamiki molekularnej, gdyż czas potrzebny na sumulację tak dużych układów przy pomocy metody MD na przestrzeni dużej liczby kroków czasowych powoduje, że czas obliczeń na dostępnych aktualnie superkomputerach jest nieakceptowalny. W skali tej natomiast operują metody bazujące na gazie siatkowym i są one wykorzystywane do symulacji wielu zjawisk [19, 128]. Metody te jednak posiadają pewne wady. Przykładowo, nie jest w nich zachowana niezmienniczość Galileusza [17] oraz złożone kształty naczynia komplikują implementację warunków brzegowych i mogą prowadzić do artefaktów numerycznych. Metoda DPD ma w założeniu łączyć zalety obydwu wspomnianych wyżej metod przy jednoczesnym pominięciu ich wad.

Pojęcia „cząstki" w metodzie DPD nie należy identyfikować z atomem bądź molekułą modelowanego płynu. Cząstka reprezentuje raczej pewien rozciągły obszar płynu lub jego kroplę (ang. „droplet"). Proces, w którym przechodzi się od obrazu mikroskopowego do obrazu właściwego skali mezo i w którym powstaje pojęcie cząstki DPD jest nazywany „coarse-graining". W procesie tym opis szczegółowy jest zastępowany przez opis statystyczny, a zmienne opisujące właściwości molekuł zastępowane są przez średnie dla coraz większych objętości [49].

Wersja DPD zaproponowana w [73] jest metodą izotermiczną, w której modelowany płyn znajduje się w stałej temperaturze. W wersji tej nie uwzględniono wszystkich wymaganych teoretycznych założeń, przez co musiała ona zostać poprawiona. Modyfikacja

została zaproponowana w pracy [47]. W wersji tej temperatura układu jest powiązana z fluktuacjami termicznymi i wystepuje jawnie w równaniach modelu.

W metodzie DPD siła pomiędzy cząstkami i oraz j składa się z trzech członów:

$$\mathbf{F}_{ij} = \mathbf{F}_{ij}^{C} + \mathbf{F}_{ij}^{D} + \mathbf{F}_{ij}^{B}, \tag{3.2}$$

gdzie $\mathbf{F}_{ij}^{C}$ oznacza człon konserwatywny, $\mathbf{F}_{ij}^{D}$ człon dyssypatywny oraz $\mathbf{F}_{ij}^{B}$ człon brownowski. Człony te mają następujące postaci:

$$\mathbf{F}_{ij}^{C}\mathrm{d}t = \pi\omega^{1/2}(r_{ij})\mathbf{e}_{ij}\mathrm{d}t, \tag{3.3}$$

$$\mathbf{F}_{ij}^{D}\mathrm{d}t = -\gamma m\omega(r_{ij})\left(\mathbf{e}_{ij}\cdot\mathbf{v}_{ij}\right)\mathbf{e}_{ij}\mathrm{d}t, \tag{3.4}$$

$$\mathbf{F}_{ij}^{B}\mathrm{d}t = \sigma\omega^{1/2}(r_{ij})\mathbf{e}_{ij}\theta_{ij}\sqrt{\mathrm{d}t}. \tag{3.5}$$

Funkcja ważąca $\omega(r)$ dla argumentów $r > r_{cut}$ przyjmuje wartość 0, natomiast dla argumentów $r < r_{cut}$ jest postaci $\omega(r) = A_D(1 - r/r_{cut})$. Z warunku normalizacyjnego

$$\int \mathrm{d}\mathbf{r}\omega(r) = \frac{1}{n} \tag{3.6}$$

można wyprowadzic wyrażenie na stałą A_D dla dowolnego wymiaru D problemu: $A_1 = 1/nr_{cut}$, $A_2 = 3/n\pi r_{cut}^2$ oraz $A_3 = 3/n\pi r_{cut}^3$, gdzie $n = N/V$ oznacza gęstość cząsteczkową. Pozostałe symbole występujące w powyższych równaniach oznaczają parametry modelu DPD: π, γ, σ oraz zmienną losową o rozkładzie normalnym θ_{ij}, dla której średnia jest równa 0, a wariancja 1.

Człon dyssypatywny odprowadza energię z układu. Aby nie doprowadziło to do „zamrożenia" cząstek płynu, wprowadzono człon brownowski F^B odpowiadający za szum termiczny. Na podstawie twierdzenia fluktuacyjno-dyssypacyjnego, wyprowadzono warunek jaki musi być spełniony, aby obydwie siły równoważyły się dla zadanej temperatury [47]:

$$\sigma = \left(2k_B T\gamma m\right)^{1/2}, \tag{3.7}$$

gdzie k_B oznacza stałą Boltzmanna, a T zadaną temperaturę. Warunek ten wiąże ze sobą człony brownowski i dyssypatywny tak, aby ustalał się pomiędzy nimi stan równowagi dla zadanej temperatury. Warto również zwrócić uwagę, że jeśli zwiększany jest krok czasowy, to wówczas człon brownowski traci na znaczeniu. Jest to spowodowane tym, że wartości zmian pędu wynikające z sił konserwatywnych i dyssypatywnych zachowują się jak $\sim \mathrm{d}t$, podczas gdy zmiany pędu od sił brownowskich zachowują się jak $\sim \sqrt{\mathrm{d}t}$ wraz ze wzrostem wartości $\mathrm{d}t$. Z wzoru (3.7) wynika także, że stałą fizyczną, która wiąże ze sobą odpowiednie oddziaływania na poziomie właściwej skali jest stała Boltzmanna k_B. To dzięki tej stałej człon brownowski jest zaniedbywalny w symulacjach zjawisk przebiegających w skali makro.

Metoda DPD znalazła szerokie zastosowania w modelowaniu zjawisk zachodzących w płynach złożonych. Z powodzeniem została wykorzystana do symulacji mieszanin koloidów [41, 16], roztworów polimerów [121] czy dwóch niemieszających się płynów [37].

3.3 Metoda SPH

Metoda SPH (Smoothed Particle Hydrodynamics) powstała pod koniec lat siedemdziesiątych w celu symulowania zjawisk astrofizycznych [90, 58]. Na przestrzeni kilku dekad została z powodzeniem zastosowana do modelowania zjawisk takich, jak np. formowanie się księżyca [14], powstawanie gwiazd [11, 10] powstawanie i ewolucja galaktyk [93, 28] i wiele

innych. Od początku lat dziewięćdziesiątych metoda znalazła również zastosowanie w modelowaniu zjawisk przebiegających w o wiele mniejszej skali, a dotyczących np. symulowania przepływu płynów w skali makroskopowej [99].

Główną ideą metody SPH jest aproksymacja pola dowolnej wielkości fizycznej na podstawie jej wartości w wybranych punktach przestrzeni. Aproksymacji dokonuje się wykorzystując funkcję jądra, nazywaną inaczej funkcją wagową, funkcją rozmycia bądź kernelem. Wartość funkcji dąży do zera w nieskończoności, a całka z funkcji jest równa jedności. Punkty, w których oblicza się wartości wielkości fizycznych to cząstki modelu SPH. Postać funkcji jądra powoduje rozmycie cząstek SPH w przestrzeni. Masa cząstek, wraz z innymi ich parametrami, nie jest zlokalizowana w jednym punkcie przestrzeni, lecz jest rozmyta w sąsiedztwie tego punktu. Podstawowym wzorem aproksymującym jest wzór:

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\mathbf{r} - \mathbf{r}_j|, h), \tag{3.8}$$

gdzie m oznacza masę cząstki, ρ gęstość, W funkcję jądra, a indeks j numer cząstki. Wielkość h to tzw. odległość rozmycia (ang. smoothing length); jej długość jest równa połowie promienia obcięcia stosowanego w symulacji. Równanie to opisuje, jak dowolna wielkość skalarna A w punkcie $\mathbf{r}$ jest obliczana na podstawie wartości A_j z punktów $\mathbf{r}_j$. Suma w powyższym wzorze przebiega po wszystkich cząstkach SPH modelowanego układu. Ponieważ funkcja ważąca W ma ograniczoną dziedzinę oraz dla punktów spoza dziedziny przyjmuje się, że funkcja ma wartość 0, to pod sumą w powyższym wzorze wystarczy uwzględnić cząstki SPH, do których odległość jest mniejsza od promienia obcięcia, czyli cząstki, dla których wartość funkcji jądra jest niezerowa. Szczegóły dotyczące teoretycznego wyprowadzenia tej zależności można znaleźć w pracy [99]. Po podstawieniu w równaniu (3.8) gęstości ρ jako wielkości skalarnej A otrzymuje się podstawowy wzór na gęstość w metodzie SPH [97] dla cząstki o numerze i:

$$\rho_i = \sum_j m_j W_{ij}, \tag{3.9}$$

gdzie $W_{ij} = W(r_{ij}, h)$ oznacza funkcję jądra obliczoną dla cząstek o numerach i oraz j.

Możliwych jest wiele postaci funkcji jądra, z których każda cechuje się odpowiednimi właściwościami. Najczęściej używane funkcje jądra zostały omówione w sekcji 4.2. Najprostszą i spełniającą wszystkie warunki teoretyczne funkcją jądra jest funkcja Gaussa. W praktyce jednak korzysta się z funkcji sklejanej, ze względu na jej zwarty nośnik, dzięki któremu można wprowadzić do symulacji promień obcięcia.

Metoda SPH jest dyskretną wersją równań Naviera-Stokesa [45]. Cząstki SPH oddziałują pomiędzy sobą siłami hydrodynamicznymi. Poprzez dyskretyzację równań Naviera-Stokesa otrzymuje się równania na wartości gradientu ciśnienia dla cząstek SPH, z których oblicza się działające na cząstki siły. W ten sposób otrzymuje się wielkości, dzięki którym można rozwiązać dla cząstek SPH równania ruchu w sposób analogiczny, jak w dynamice molekularnej.

Równanie ruchu opisujące ruch cząstki SPH [97] ma postać:

$$\frac{d\mathbf{v}_i}{dt} = -\sum_j m_j \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2} + \Pi_{ij} \right) \nabla_i W_{ij}, \tag{3.10}$$

gdzie P oznacza ciśnienie hydrostatyczne, Π_{ij} dodatkowy człon wprowadzający siły lepkości działajace pomiędzy cząstkami i oraz j, natomiast ∇_i oznacza gradient obliczony w punkcie $\mathbf{r}_i$. Dokładne, teoretyczne wyprowadzenie równania (3.10) można znaleźć w [87].

Sposób obliczania gęstości cząstek według wzoru (3.9) jest w pewnych przypadkach niewłaściwy. Dotyczy to modelowania makroskopowych płynów nieściśliwych z powierzchnią swobodną. Jest to spowodowane tym, że przy brzegu cieczy, bądź przy jej powierzchni swobodnej obliczana wg niego gęstość jest mniejsza niż w wewnątrz obszaru zajmowanego przez ciecz. Wynika to z faktu, że liczba cząstek przy powierzchni cieczy, które obejmuje suma we wzorze (3.9), jest mniejsza. Ze wzoru tego wynika również, że przy przechodzeniu przez granicę cieczy gęstość zachowuje się w sposób ciągły. W skali makroskopowej dla cieczy rzeczywistych gęstość zmienia się w sposób nieciągły. Z tych powodów, we wspomnianych symulacjach, korzysta się z wzoru [80, 91, 87]:

$$\frac{d\rho_i}{dt} = \sum_j m_j(\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla_i W_{ij}. \tag{3.11}$$

Dzięki zastosowaniu powyższego wzoru, w każdym kroku czasowym nie oblicza się na nowo gęstości cząstki, a obliczana jest jedynie poprawka jej gęstości. Wartość poprawki zależy od względnego ruchu oddziałujących cząstek. Ponieważ są to jedynie zmiany gęstości, konieczne jest więc określenie gęstości początkowej cząstek, jaka jest im przypisana podczas uruchamiania symulacji. Jest to część budowania konfiguracji początkowej symulacji, która została opisana w sekcji 4.3.

Równanie ruchu metody SPH powoduje, że w każdym kroku czasowym konieczne jest obliczenie wartości ciśnienia dla każdej cząstki. Wykorzystuje sie w tym celu równanie stanu. W zależności od docelowego zastosowania wykorzystuje się różne wersje tego równania. Jednym z najprostszych w implementacji jest równanie stanu dla gazu doskonałego. Jednak dla przypadków symulacji płynów nieściśliwych, takich jak np. woda, konieczne jest zastosowanie innego równania stanu. Odpowiednie równanie zostało zaproponowane w pracy [9]. Znalazło ono szereg zastosowań w modelowaniu płynów nieściśliwych. Równanie to jest postaci:

$$P_i = B\left(\left(\frac{\rho_i}{\rho_0}\right)^{\gamma} - 1\right), \tag{3.12}$$

gdzie ρ_0 jest ustaloną wartością gęstości. Powszechnie przyjmuje się, że $\gamma = 7$, a współczynnik B jest dobierany tak, aby względne zmiany gęstości były małe, tzn. $|\delta\rho|/\rho \sim 0.01$. Warunek ten jest spełniony, gdy $B = 100\rho_0 v_{\max}^2/\gamma$, gdzie $v_{\max}$ jest maksymalną prędkością pomiędzy cząstkami i jest ona przeważnie dość łatwa do oszacowania [99]. Przyjmuje się zwykle, że prędkość dźwięku jest równa $c = 10v_{\max}$, co pozwala od razu wyznaczyć prędkość maksymalną.

Lepkość w metodzie SPH wprowadza się poprzez dodatkowy człon Π_{ij} w równaniu ruchu (3.10). Już pierwsze symulacje wykonane przy pomocy metody SPH zawierały człon lepkości [90]. Początkowo lepkość w metodzie SPH została wprowadzona jako tzw. sztuczna lepkość, której zadaniem było umożliwienie symulacji zjawisk uderzeniowych. Było to konieczne, aby algorytm numeryczny zachowywał się stabilnie [99]. Na przestrzeni lat powstało wiele odmian sztucznej lepkości. Do najpopularniejszych należy model zaproponowany przez Monaghana oraz Gingolda [100], który wyrażony jest wzorem:

$$\Pi_{ij} = \begin{cases} \frac{-\alpha c_{ij}\mu_{ij} + \beta\mu_{ij}^2}{\rho_{ij}} & \text{, gdy } \mathbf{v}_{ij} \cdot \mathbf{r}_{ij} < 0, \\ 0 & \text{, gdy } \mathbf{v}_{ij} \cdot \mathbf{r}_{ij} \geqslant 0, \end{cases} \tag{3.13}$$

gdzie

$$\mu_{ij} = \frac{h\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}}{r_{ij}^2 + \eta^2}, \tag{3.14}$$

α oraz β to parametry modelu lekości, $c_{ij} = 1/2 \cdot (c_i + c_j)$ oznacza średnią prędkość dźwięku dla pary cząstek o numerach i oraz j, ρ_{ij} średnią gęstość, h odległość rozmycia. Czynnik $\eta^2 = (0,1h)^2$ został wprowadzony, aby uniknąć osobliwości w równaniu, gdy $r_{ij} \to 0$. Innym, dość często wykorzystywanym, modelem sztucznej lepkości w SPH jest model zaproponowany przez Hernquista i Katza [70], który został opisany zależnościami:

$$\Pi_{ij} = \frac{q_i}{\rho_i^2} + \frac{q_j}{\rho_j^2}, \tag{3.15}$$

gdzie

$$q_i = \begin{cases} \alpha h_i \rho_i c_i |\nabla \cdot \mathbf{v}|_i + \beta h_i^2 \rho_i |\nabla \cdot \mathbf{v}|_i^2 & \text{, gdy } \nabla \cdot \mathbf{v} < 0, \\ 0 & \text{, gdy } \nabla \cdot \mathbf{v} \geqslant 0, \end{cases} \tag{3.16}$$

$$\nabla \cdot \mathbf{v} = -\frac{1}{\rho_i} \sum_{j=1}^{N} m_j v_{ij} \cdot \frac{1}{2} \left[\nabla_i W(r_{ij}, h_i) + \nabla_i W(r_{ij}, h_j) \right]. \tag{3.17}$$

Głównym celem sztucznej lepkości było umożliwienie modelowania zjawisk, w których występują zderzenia lub fale uderzeniowe. Stąd też siły lepkości wprowadzono jako działające jedynie wtedy, gdy cząstki się do siebie zbliżały. Formalnie warunek ten oznacza, że sztuczna lepkość działa jedynie wówczas, kiedy dywergencja pola prędkości jest ujemna, $\nabla \cdot \mathbf{v} < 0$. W formalizmie SPH warunek ten wyraża się poprzez nierówność $\mathbf{v}_{ij} \cdot \mathbf{r}_{ij} < 0$.

Sztuczna lepkość w metodzie SPH została skonstruowana na podobieństwo lepkości występującej w gazach. Z tego powodu okazało się, że sztuczna lepkość dość dobrze modeluje lepkość rzeczywistą obecną w płynach. Stąd też powyższe wzory opisujące sztuczną lepkość stosuje się również do modelowania lepkości rzeczywistej z tą zmianą, że działają one zarówno dla dywergencji dodatniej, jak i ujemnej. Innymi słowy, siła lepkości działa pomiędzy cząstkami również wtedy, gdy cząstki się od siebie oddalają [99].

Metoda SPH została z powodzeniem zastosowana do modelowania zachowania się powierzchni swobodnych [34], uderzeń w powierzchnię i dalszego zachowania się ciał sztywnych [101] badź też przepływów wielofazowych i wieloskładnikowych [76].

3.4 Formalizm GENERIC

Formalizmem GENERIC nazywana jest struktura matematyczna równań termodynamicznych dla nierównowagowych układów ewoluujących do stanu równowagi. Nazwa GENERIC pochodzi od nazwy równania będącego podstawą teoretyczną całego formalizmu (Generic Equation for Non-Equilibrium Reversible-Irreversible Coupling). Jest to ogólna postać równania dynamiki dla układów nierównowagowych z dynamiką zarówno odwracalną, jak i nieodwracalną. Równanie GENERIC opisuje uniwersalne właściwości makroskopowej dynamiki układów ewoluujących do stanu równowagi. Prace nad opisem tych właściwości były prowadzone przez wielu badaczy, m.in. przez Onsagera [106, 107], Casimira [29], Ginzburga i Landaua [129] czy też innych [60, 59]. Wyprowadzone przez nich równania i teorie okazały się być przypadkiem szczególnym równania GENERIC, które zostało zaproponowane w pracach [64, 112]. Równanie to jest równaniem fenomenologicznym. Zostało zaproponowane na podstawie obserwacji dynamiki płynów złożonych podczas ich ewolucji do stanu równowagi. Oprócz warunku zgodności z danymi doświadczalnymi równanie to spełnia klika założeń teoretycznych, z których najbardziej istotnym jest forma równania. Miała ona być postaci podobnej do równania Ginzburga-Landaua, nazywanego inaczej równaniem relaksacji [64].

Niech x oznacza wektor zmiennych stanu opisujących modelowany układ. Równanie GENERIC określające ewolucję czasową wektora x jest postaci:

$$\frac{dx}{dt} = L\frac{\partial E}{\partial x} + M\frac{\partial S}{\partial x}, \tag{3.18}$$

gdzie $E \equiv E(x)$ oznacza energię, a $S \equiv S(x)$ entropię dla wektora x. Pierwszy człon prawej strony powyższego równania odpowiada odwracalnej części dynamiki układu, natomiast drugi człon części nieodwracalnej. Macierze L oraz M reprezentują operatory odpowiadające za odpowiednio odwracalną i nieodwraclną część dynamiki układu. Powyższa postać równania GENERIC jest jego uproszczoną wersją, w której pominięto fluktuacje termiczne.

Aby równanie (3.18) poprawnie opisywało termodynamikę układu rzeczywistego, model przez nie generowany musi spełniać pierwszą i drugą zasady termodynamiki. Narzuca to warunki na operatory L i M. Pierwsze dwa warunki zapewniają, że operatory te działają rozłącznie, to znaczy, że operator L nie wpływa na część nieodwracalną, a operator M na część odwracalną:

$$L\frac{\partial S}{\partial x} = 0, \qquad M\frac{\partial E}{\partial x} = 0. \tag{3.19}$$

Dodatkowo operatory te muszą spełniać odpowiednie warunki symetrii. Warunki te można wyrazić za pomocą nawiasów:

$$\{A, B\} = \left\langle \frac{\delta A}{\delta x} \middle| L\frac{\delta B}{\delta x} \right\rangle, \tag{3.20}$$

$$[A, B] = \left\langle \frac{\delta A}{\delta x} \middle| M\frac{\delta B}{\delta x} \right\rangle, \tag{3.21}$$

gdzie A i B są rzeczywistymi funkcjami określonymi na przestrzeni stanów $\{x\}$, a notacja $\langle|\rangle$ oznacza iloczyn skalarny. Operator L musi być antysymetryczny

$$\{A, B\} = -\{B, A\} \tag{3.22}$$

oraz spełniać tożsamość Jacobiego w poniższy sposób:

$$\{A, \{B, C\}\} + \{B, \{C, A\}\} + \{C, \{A, B\}\} = 0. \tag{3.23}$$

Operator M natomiast musi być symetryczny

$$[A, B] = [B, A] \tag{3.24}$$

oraz nieujemnie określony

$$[A, B] \geqslant 0. \tag{3.25}$$

Warunki (3.19) oraz (3.22)-(3.25) implikują zachowanie w układzie pierwszej i drugiej zasad termodynamiki, tzn. $\dot{E} = 0$ oraz $\dot{S} \geqslant 0$.

Deterministyczne równanie modelu GENERIC (3.18) jest w rzeczywistości przybliżeniem, w którym fluktuacje termiczne zostały pominięte. W przypadku uwzględnienia fluktuacji termicznych dynamika układu jest opisywana przez odpowiednie równanie Fokkera-Plancka [64]. Równoważne do niego równanie modelu GENERIC przybiera postać stochastycznego równania różniczkowego:

$$dx = \left[L\frac{\partial E}{\partial x} + M\frac{\partial S}{\partial x} + k_B\frac{\partial}{\partial x}M\right]dt + d\widetilde{x}, \tag{3.26}$$

gdzie $d\widetilde{x}$ oznacza człon stochastyczny. Człon ten jest kombinacją liniową niezależnych przyrostów procesu Wienera i spełnia zasadę Itô:

$$d\widetilde{x}d\widetilde{x}^T = 2k_B M dt, \tag{3.27}$$

co oznacza, że $d\widetilde{x}$ jest wielkością infinitezymalnie małą rzędu $1/2$ względem dt.

Przedstawione powyżej równania GENERIC, wraz z nałożonymi na nie warunkami, powodują, że każdy model dynamiczny, którego równania przyjmują postać GENERIC jest spójny termodynamicznie. Cała trudność polega na odpowiednim sformułowaniu modelu i sprowadzeniu jego równań do postaci zgodnej z równaniem GENERIC. Każdy model wyrażony w ramach formalizmu GENERIC jest modelem spójnym termodynamicznie [125].

3.5 Metoda SDPD

Metoda SDPD [48] (Smoothed Dissipative Particle Dynamics) powstała jako modyfikacja i rozszerzenie metody DPD. Główną motywacją, dla której metoda ta została zaproponowana była potrzeba rozszerzenia dotychczas istniejących metod o model spójny termodynamicznie, w którym wszystkie istotne parametry i skale przestrzenno-czasowe są określane jako parametry wejściowe do symulacji. Okazało się to możliwe poprzez zastosowanie formalizmu GENERIC. Ku pewnemu zaskoczeniu okazało się, że otrzymana forma sił zachowawczych jest taka sama jak w modelu SPH zaproponowanym przez Monaghana [48]. Z tego też powodu niektórzy nazywają metodę SDPD jako spójną termodynamicznie wersję SPH. W rzeczywistości jednak, SDPD jest uogólnieniem metody DPD.

Dzięki ujęciu równań metody SDPD w formalizmie GENERIC metoda ta jest metodą spójną termodynamicznie, co oznacza, że w prowadzonych przy jej pomocy symulacjach zachowane są pierwsza i druga zasady termodynamiki. Metoda ta przeznaczona jest do modelowania zjawisk zachodzących w mezoskali. Modelowany płyn wykazuje właściwości hydrodynamiczne i dodatkowo obecne są w nim fluktuacje termiczne. Równania metody zapewniają, że fluktuacje termiczne wraz z przechodzeniem od skali mezo do skali makroskopowej stają się zaniedbywalne. W mezoskali równania metody SDPD przechodzą w równania SPH, co oznacza, że są one dyskretną wersją równań hydrodynamiki.

Podstawowe równania zarządzające ewolucją układu przyjmują postać:

$$\begin{aligned}
\mathrm{d}\mathbf{r}_i =& \mathbf{v}_i \mathrm{d}t, \\
m\mathrm{d}\mathbf{v}_i =& \sum_j \left[\frac{P_i}{d_i^2} + \frac{P_j}{d_j^2}\right] F_{ij}\mathbf{r}_{ij}\mathrm{d}t - \sum_j (1-\xi_{ij}) a_{ij}\mathbf{v}_{ij}\mathrm{d}t \\
& - \sum_j (1-\xi_{ij}) \left(\frac{a_{ij}}{3} + b_{ij}\right) \mathbf{e}_{ij}\mathbf{e}_{ij}\cdot\mathbf{v}_{ij}\mathrm{d}t + m\mathrm{d}\widetilde{\mathbf{v}}_i, \\
T_i \mathrm{d}S_i =& \frac{1}{2}\sum_j \left(1 - \xi_{ij} - \frac{T_j}{T_i+T_j}\frac{k_B}{C_i}\right) \left[a_{ij}\mathbf{v}_{ij}^2 + \left(\frac{a_{ij}}{3} + b_{ij}\right)(\mathbf{e}_{ij}\cdot\mathbf{v}_{ij})^2\right]\mathrm{d}t \\
& - \frac{2k_B}{m}\sum_{j\neq i} \frac{T_i T_j}{T_i+T_j}\left(\frac{10}{3}a_{ij} + b_{ij}\right)\mathrm{d}t - \sum_j c_{ij}T_{ij}\mathrm{d}t \\
& - \frac{k_B}{C_i}\sum_{j\neq i} c_{ij}T_j \mathrm{d}t + T_i \mathrm{d}\widetilde{S}_i,
\end{aligned} \tag{3.28}$$

gdzie T oznacza temperaturę, S entropię a d odwrotność objętości V obliczaną na podstawie wzoru

$$\frac{1}{V_i} = d_i = \sum_j W_{ij}. \tag{3.29}$$

Ciśnienie cząstki o numerze i, oznaczane przez P_i, oraz jej temperaturę, T_i oblicza się korzystając z równań:

$$P_i = -\frac{\partial E^{eq}}{\partial V_i} \ , \ T_i = \frac{\partial E^{eq}}{\partial S_i}, \tag{3.30}$$

gdzie $E_i = E^{eq}(m, S_i, V_i)$ jest równowagowym równaniem stanu dla modelowanego płynu. Funkcja F_{ij} jest określona jako $F_{ij} = -\nabla W_{ij}/r_{ij}$, a T_{ij} oznacza różnicę temperatur cząstek o numerach i oraz j: $T_{ij} = T_i - T_j$. W równaniach (3.28) wielkość ξ_{ij} jest zdefiniowana jako:

$$\xi_{ij} = \frac{T_i T_j}{(T_i + T_j)^2} \left(\frac{k_B}{C_i} + \frac{k_B}{C_j} \right), \tag{3.31}$$

gdzie C_i jest ciepłem właściwym w stałej objętości dla cząstki i.

Metoda SDPD jest metodą, w której makroskopowe parametry transportu wystepują jawnie w parametrach równań modelu. Są one obecne we wzorach na współczynniki a_{ij}, b_{ij} oraz c_{ij}:

$$a_{ij} = \left(\frac{5\eta}{3} - \zeta \right) \frac{F_{ij}}{d_i d_j}, \tag{3.32}$$

$$b_{ij} = 5 \left(\frac{\eta}{3} + \zeta \right) \frac{F_{ij}}{d_i d_j} - \frac{a_{ij}}{3}, \tag{3.33}$$

$$c_{ij} = 2\kappa \frac{F_{ij}}{d_i d_j}, \tag{3.34}$$

gdzie η oznacza lepkość ścinania (ang. shear viscosity), ζ lepkość objętościową (ang. bulk viscosity), a κ przewodność cieplną.

Metoda SDPD jest metodą wyrażoną w formalizmie GENERIC. Ponieważ jest to metoda opisująca oddziaływania w skali mezoskopowej, konieczne jest ujecie w jej ramach fluktuacji termicznych. Fluktuacje termiczne są obecne we wzorach (3.28) jako $m\mathrm{d}\widetilde{\mathbf{v}}_i$ oraz $T_i \mathrm{d}\widetilde{S}_i$. Jednak formalizm GENERIC wymusza na nich serię warunków, które wyrażone są poniższymi wzorami:

$$m\mathrm{d}\widetilde{\mathbf{v}}_i = \sum_j \left(A_{ij} \overline{\mathrm{d}\mathbf{W}}^S_{ij} + B_{ij} \frac{\mathbf{1}}{3} tr[\mathrm{d}\mathbf{W}_{ij}] \right) \cdot \mathbf{e}_{ij}, \tag{3.35}$$

$$T_i \mathrm{d}\widetilde{S}_i = -\frac{1}{2} \sum_j \left(A_{ij} \overline{\mathrm{d}\mathbf{W}}^S_{ij} + B_{ij} \frac{\mathbf{1}}{3} tr[\mathrm{d}\mathbf{W}_{ij}] \right) : \mathbf{e}_{ij}\mathbf{v}_{ij} + \sum_j C_{ij} \mathrm{d}V_{ij}, \tag{3.36}$$

gdzie amplitudy A_{ij}, B_{ij} oraz C_{ij} są określone jako:

$$A_{ij} = \sqrt{8 k_B a_{ij} \frac{T_i T_j}{T_i + T_j}}, \tag{3.37}$$

$$B_{ij} = \sqrt{12 k_B b_{ij} \frac{T_i T_j}{T_i + T_j}}, \tag{3.38}$$

$$C_{ij} = \sqrt{2 k_B c_{ij} T_i T_j}. \tag{3.39}$$

Występująca w równaniu 3.36 wielkość $\mathrm{d}V_{ij}$ jest niezależnym przyrostem procesu Wienera, natomiast $\mathrm{d}\mathbf{W}_{ij}$ jest macierzą takich przyrostów. $\overline{\mathrm{d}\mathbf{W}}^S_{ij}$ jest jej symetryczną, bezśladową częścią:

$$\overline{\mathrm{d}\mathbf{W}}^S_{ij} = \frac{1}{2} \left(\mathrm{d}\mathbf{W}_{ij} + \mathrm{d}\mathbf{W}^T_{ij} \right) - \frac{\mathbf{1}}{D} tr[\mathrm{d}\mathbf{W}_{ij}]. \tag{3.40}$$

Niezależne przyrosty procesu Wienera spełniają poniższą mnemotechniczną regułę Itô:

$$\mathrm{d}\mathbf{W}_{ii'}^{\alpha\alpha'}\mathrm{d}\mathbf{W}_{jj'}^{\beta\beta'} = [\delta_{ij}\delta_{i'j'} + \delta_{ij'}\delta_{i'j}]\delta^{\alpha\beta}\delta^{\alpha'\beta'}\mathrm{d}t, \tag{3.41}$$

$$\mathrm{d}V_{ii'}\mathrm{d}V_{jj'} = [\delta_{ij}\delta_{i'j'} - \delta_{ij'}\delta_{i'j}]\mathrm{d}t, \tag{3.42}$$

$$\mathrm{d}\mathbf{W}_{ii'}^{\alpha\alpha'}\mathrm{d}V_{ii'} = 0, \tag{3.43}$$

co oznacza, że są one wielkościami infinitezymalnymi rzędu 1/2 względen dt. Opisują one biały szum gaussowski taki, że $\langle \mathrm{d}W_{ij}(t)\rangle = 0$.

Metoda SDPD jest metodą skomplikowaną matematycznie. W porównaniu do metod SPH i DPD występuje w niej o wiele więcej parametrów oraz równania ją opisujące są w znacznym stopniu skomplikowane. Z tego powodu metoda ta jest dość trudna do zaimplementowania. Biorąc pod uwagę również to, że jest to stosunkowo nowa metoda, powstała w 2003 roku, łatwo zrozumieć, że nie znalazła ona jeszcze szerokiego zastosowania. Dotychczas ukazało się około 50 prac cytujących oryginalną pracę dotyczącą SDPD. Jednak jedynie w kilku z nich zaprezentowano wyniki uzyskane przy jej pomocy. Dotyczy to modelowania zawiesin polimerów [86], lepkich przepływów [124], dyfuzji w układzie dwóch niemieszających się płynów [130] czy też przepływów wielofazowych [76].

3.6 Schematy całkowania równań ruchu

Same równania ruchu metody nie stanowią kompletnych narzędzi do uruchomienia symulacji. Stanowią one jedynie opis zmian, jakie zachodzą w modelowanym układzie. Konieczny jest jeszcze wybór i zastosowanie odpowiedniego algorytmu numerycznego rozwiązującego równania różniczkowe, jakimi są równania ruchu. Dostępnych jest wiele schematów, z których każdy posiada cechy charakterystyczne. Przy wyborze odpowiedniego schematu należy brać pod uwagę m.in. jego złożoność obliczeniową, rząd dokładności, jego stabilność czy wszechstronność jak również wielkość kroku czasowego, jaka może być zastosowana w ramach schematu.

Najstarszym sposobem całkowania równań różniczkowych jest metoda Eulera stworzona już w 1768 roku [24]. Jej równania w zastosowaniu do symulacji numerycznych są przedstawione poniżej:

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \mathbf{v}_i^n\mathrm{d}t + O(\mathrm{d}t^2), \tag{3.44}$$

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \mathbf{a}_i^n\mathrm{d}t + O(\mathrm{d}t^2). \tag{3.45}$$

Metoda ta jest metodą pierwszego rzędu. Z tego powodu jest ona stosowana dość sporadycznie i jedynie w celach edukacyjnych.

Z metod drugiego rzędu należy wymienić schemat żabiego skoku (ang. leap-frog) [118] opisany równaniami:

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \mathbf{v}_i^{n+\frac{1}{2}}\mathrm{d}t + O(\mathrm{d}t^3), \tag{3.46}$$

$$\mathbf{v}_i^{n+\frac{1}{2}} = \mathbf{v}_i^{n-\frac{1}{2}} + \mathbf{a}_i^n\mathrm{d}t + O(\mathrm{d}t^3). \tag{3.47}$$

W metodzie tej prędkość w kroku n oblicza sie jako średnią

$$\mathbf{v}_i^n = \frac{1}{2}\left(\mathbf{v}_i^{n+\frac{1}{2}} + \mathbf{v}_i^{n-\frac{1}{2}}\right).$$

Podobnym schematem jest algorytm Verleta [133]:

$$\mathbf{r}_i^{n+1} = 2\mathbf{r}_i^n - \mathbf{r}_i^{n-1} + \mathbf{a}_i^n \mathrm{d}t^2 + O(\mathrm{d}t^4), \tag{3.48}$$

$$\mathbf{v}_i^n = \frac{\mathbf{r}_i^{n+1} - \mathbf{r}_i^{n-1}}{2\mathrm{d}t} + O(\mathrm{d}t^2), \tag{3.49}$$

bądź jego wariant, tzw. prędkościowy schemat Verleta (ang. velocity Verlet):

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \mathbf{v}_i^n \mathrm{d}t + \frac{1}{2}\mathbf{a}_i^n \mathrm{d}t^2, \tag{3.50}$$

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \frac{\mathbf{a}_i^n + \mathbf{a}_i^{n+1}}{2}\mathrm{d}t. \tag{3.51}$$

Zarówno algorytm żabiego skoku, jak i algorytmy Verleta mają swoje ograniczenia. Najbardziej kłopotliwym jest to, że aby obliczyć prędkość w kroku n należy najpierw znać wartość siły w tym kroku. Oznacza to, że nie można stosować tych schematów w sytuacjach, gdy wartość siły zależy w jakikolwiek sposób od prędkości cząstek. Ma to miejsce chociażby w metodzie DPD przy obliczaniu członu dyssypatynego, czy w metodzie SPH, kiedy obecny jest człon odpowiadający za lepkość. Okazało się jednak, że schemat Verleta można bardzo łatwo zmodyfikować, aby usunąć tą niedogodność. Odpowiednie modyfikacje przefdstawiono głównie w pracach [65, 56].

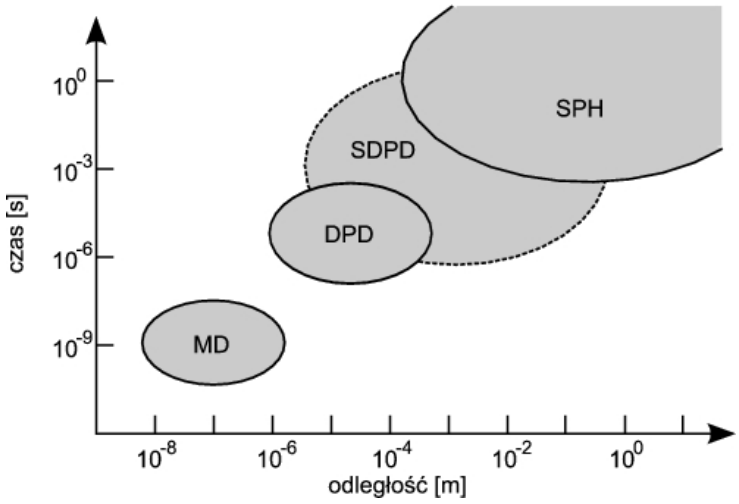
Powszechnie stosowanymi schematami są schematy wielokrokowe (ang. multistep methods). Ich główną ideą jest to, że obliczane wartości w kroku $n+1$ nie zależą jedynie od wartości w kroku n, ale również od wartości z wcześniejszych kroków. Jako przykład może posłużyć rodzina schematów Adamsa-Bashfortha. W swojej najogólniejszej postaci schemat Adamsa-Bashfortha rzędu N przyjmuje postać [24]:

$$x^{n+1} - x^n = \mathrm{d}t \sum_{j=0}^{N-1} a_j \dot{x}^{n-j}. \tag{3.52}$$

Współczynniki a_j powinny być tak określone, aby po wstawieniu rozwinięć w szereg Taylora dla x oraz $\dot{x}$ w (3.52) skróciły się wszystkie wyrazy rzędu mniejszego niż N. Przy zastosowaniu schematu Adamsa-Bashfortha należy oczywiście zmienną x zastąpić położeniem $\mathbf{r}$ oraz prędkością $\mathbf{v}$.

Obecność w metodach członu losowego powoduje, że równania ruchu układu mają charakter równań stochastycznych. Niestety, stosowane w metodach deterministycznych schematy całkowania w przypadku stochastycznych równań ruchu powodują „dryf" energii kinetycznej układu w przypadku, gdy powinna ona zachowywać wartość stałą oraz inne artefakty. Konieczna wówczas staje sie modyfikacja schematów tak, aby były one poprawne dla równań stochastycznych. Odpowiednie modyfikacje przedstawiono w pracy [132]. Polegają one na uwzględnieniu w warunku równowagi termodynamicznej wpływu fluktuacji termicznych.

Ponieważ równania ruchu są dwuczęściowe, tzn. w każdym kroku symulacji należy obliczać dwie wielkości: położenia i prędkości, możliwe jest zastosowanie podejścia mieszanego. Taki sposób został zrealizowany w pracy [42], w której do obliczania położeń zastosowano algorytm żabiego skoku, natomiast to obliczania gęstości zastosowano schemat Adamsa-Bashfortha. Podejście to jest kompromisem pomiędzy szybkim, ale mniej dokładnym schematem drugiego rzędu, a dokładniejszymi i wewnętrznie spójnym schematem rzędu wyższego. Przy zastosowaniu dodatkowych zmiennych możliwe jest przekształcenie tego podejścia w schemat rzędu trzeciego [42].



Rysunek 3.1: Metody cząstek i odpowiadające im skale przestrzenno-czasowe.

3.7 Podsumowanie

W rozdziale przedstawiono podstawowe bezsiatkowe metody cząstek wykorzystywane do modelowania płynów. Dla każdej z nich określono skale przestrzenno-czasowe, w której metoda operuje jak również zaprezentowano podstawowe równania metody. Umiejscowienie przedstawionych metod na tle skal przestrzenno-czasowych zobrazowano na rysunku (3.1). Dla każdej metody podano również przykładowe zastosowania, w których metoda okazała się skuteczna. Przedstawiono również schamaty całkowania używane wraz z omówionymi metodami. Metody przedstawione w rozdziale należą zarówno do tych dobrze udokumentowanych i szeroko stosowanych (MD, DPD, SPH), jak i takich, które zostały wyprowadzone stosunkowo niedawno i nie znalazły jeszcze szedokiego zastosowania (SDPD).

Rozdział 4

Aspekty implementacji sekwencyjnej

W poprzednim rozdziale przedstawiono teoretyczne modele oddziaływań pomiędzy cząstkami. Modele te, zaproponowane na podstawie teoretycznych rozważań, muszą zostać efektywnie zaimplementowane. Zagadnienia implementacji dotyczą zarówno doboru struktur danych, wyboru właściwych algorytmów (np.znajdywania sąsiadów cząstki czy rozwiązywania równań ruchu) jak również zapisu instrukcji programu realizujących wyznaczanie wartości zmiennych charakterystycznych modelu. Dla zapewnienia poprawności symulacji istotne są również wartości parametrów fizycznych, takich jak np. temperatura czy krok czasowy. Zagadnienia te stanowią treść niniejszego rozdziału.

4.1 Dobór wartości kroku czasowego

Krok czasowy Δt jest podstawową wielkością charakteryzującą symulację i mającą wpływ na jej wyniki. Podczas przeprowadzania symulacji pożądane jest, aby wykorzystywana wartość kroku czasowego była możliwie największa. Im większa wartość kroku czasowego, tym szybszy przebieg symulowanego zjawiska. Symulacja zjawiska fizycznego zachodzącego w czasie t_1 zostanie wykonana szybciej w przypadku mniejszej liczby kroków czasowych $n = t_1/\Delta t$. Jednak podobnie, jak w przypadku metod numerycznych stosowanych do rozwiązywania równań różniczkowych i stosowanego elementu skończonego, od wielkości kroku czasowego zależy poprawność rozwiązania. Zbyt duży krok czasowy generuje błąd w wynikach czyniąc je niepoprawnymi. Dlatego też w sytuacji, gdy zbyt mały krok czasowy niepotrzebnie zużywa dostępne zasoby obliczeniowe, a zbyt duży krok czasowy wypacza wyniki, konieczne jest zastosowanie wartości optymalnej.

Istnieje kilka kryteriów wspomagających wybór wielkości kroku czasowego. Jednym z nich, dotyczącym symulacji układu izolowanego, jest wymaganie, aby jego energia kinetyczna pozostawała stała [118]. W przypadku, gdy krok czasowy jest zbyt duży, energia kinetyczna układu nie jest stała i zawsze rośnie. Zbyt duży krok czasowy powoduje, że cząstki zbliżają się do siebie na zbyt małe odległości, przez co w układzie wzrasta energia potencjalna. Energia ta w kolejnych krokach obliczeniowych przekształca się w energię kinetyczną cząstek, przez co energia kinetyczna całego układu wzrasta. Dlatego też zazwyczaj konieczne jest uruchomienie kilku symulacji testowych w celu sprawdzenia zachowania się energii kinetycznej układu. Dopiero po analizie wyników symulacji wstępnych możliwe jest uruchomienie właściwych obliczeń.

Istnieją też przesłanki wskazujące na właściwą wartość kroku czasowego bez konieczności uruchamiania symulacji testowych. Rząd wielkości kroku czasowego można odczytać z rysunku (3.1). Przykładowo, dla symulacji metodą dynamiki molekularnej krok czasowy

jest rzędu femtosekundy ($1\text{fs} = 10^{-15}\text{s}$) i jego zastosowanie ogranicza czas trwania symulowanego zjawiska do nanosekund [8]. Podobnie, na podstawie skali, w której metoda operuje, można określić wielkość kroku czasowego dla metod DPD, SDPD oraz SPH. W przypadku metod DPD oraz SDPD jest to odpowiednio $10^{-12}\text{s} \div 10^{-10}\text{s}$ oraz $10^{-11}\text{s} \div 10^{-6}\text{s}$. W przypadku metody SPH krok czasowy jest niemożliwy do ograniczenia od góry, gdyż metoda ta może modelować układy dowolnie duże. Można natomiast ograniczyć od dołu i ograniczenie to wynosi 10^{-9}s. Powyższe wartości są jedynie wskazówkami dla wartości dokładnych, będącymi punktem wyjścia dla ich dokładniejszego określenia. Aby dokładniej określić wielkość kroku czasowego, konieczna jest znajomość parametrów symulacji takich, jak średnia odległość między cząstkami $\Delta x_{\text{śr}}$ oraz ich średnia prędkość $v_{\text{śr}}$. Powszechnie stosowanym kryterium określającym wartość kroku czasowego jest wymaganie, aby czastka poruszająca się ze średnią prędkością przebyła średnią odległość pomiędzy cząstkami w około dziesięciu krokach czasowych [18]. Stąd wzór na wartość kroku czasowego: $\Delta t = 0.1\Delta x_{\text{śr}}/v_{\text{śr}}$.

Innym warunkiem określającym krok czasowy jest warunek Couranta [115]. Jest on powszechnie stosowany w symulacjach metodą SPH. Zgodnie z nim maksymalna wartość kroku czasowego nie może być większa niż:

$$\Delta t_C = C \min\left(\frac{h_i}{c}\right), \tag{4.1}$$

gdzie c oznacza prędkość dźwięku w układzie, h_i długość rozmycia dla cząstki i a C jest stałą zwykle ustalaną na 0.4. Warunek ten, razem z innymi może być wykorzystany dla bardziej dokładnego wyznaczenia wartości kroku czasowego [97]. W tym celu najpierw oblicza się wartości:

$$\delta t_{\text{f}} = \min_i\left(\frac{h_i}{|\mathbf{f}_i|}\right) \tag{4.2}$$

oraz

$$\delta t_{\text{cv}} = \min_i \frac{h}{c_i + 0.6(\alpha c_i + \beta \max_j \mu_{ij})}. \tag{4.3}$$

Pierwsze z tych równań jest oparte na wartości siły na jednostkę masy $|\mathbf{f}_i|$. Drugie natomiast jest złożeniem warunku Couranta oraz warunku opartego na członie lepkości. Po obliczeniu obydwu powyższych wartości przyjmuje się $\Delta t = 0.25 \min(\delta t_{\text{f}}, \delta t_{\text{cv}})$.

4.2 Postać funkcji ważącej W

Opisywane w niniejszej pracy metody cząstek działają w skali mezoskopowej oraz w skalach wyższych. Stąd w modelach tych odległości pomiędzy punktami reprezentującymi cząstki są znacznie większe niż odległości międzycząsteczkowe, a same cząstki reprezentują obszar płynu o skończonej objętości. Zostało to uwzględnione w funkcjach rozmycia, które pojawiają się w odpowiednich wzorach opisujących oddziaływania pomiędzy cząstkami modelowanego płynu.

W metodzie SPH funkcja rozmycia nazywana jest inaczej funkcją jądra (ang. kernel function). Może ona być stosowana do interpolacji dowolnej wielkości skalarnej zgodnie z wzorem:

$$A(\mathbf{r}) = \int A(\mathbf{r}')W(\mathbf{r} - \mathbf{r}', h)d\mathbf{r}', \tag{4.4}$$

gdzie h jest długością rozmycia w metodzie SPH. Równanie to daje dokładną wartość na $A(\mathbf{r})$ w dowolnym punkcie $\mathbf{r}$, jeśli funkcja jądra jest dystrybucją δ Diraca. W praktyce

implementacyjnej wzór ten nie jest przydatny, gdyż wymagałby on znajomości wielkości A w dowolnym punkcie. W rzeczywistości wystepuje sytuacja, w której wielkość ta znana jest jedynie w skończonej liczbie punktów - cząstkach płynu. Stosuje się wówczas funkcje rozmycia, które dążą do δ Diraca, gdy długość rozmycia $h \to 0$. W sytuacji takiej, stosując przekształcenie $A(\mathbf{r}')\mathrm{d}\mathbf{r}' = \frac{A(\mathbf{r}')}{\rho(\mathbf{r}')}\rho(\mathbf{r}')\mathrm{d}\mathbf{r}' = \frac{A(\mathbf{r}')}{\rho(\mathbf{r}')}m(\mathrm{d}\mathbf{r}')$, całkę we wzorze 4.4 zastępuje się sumą:

$$A(\mathbf{r}) = \int m(\mathrm{d}\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)\frac{A(\mathbf{r}')}{\rho(\mathbf{r}')} = \sum_i m_i W(\mathbf{r}_i-\mathbf{r}',h)\frac{A_i}{\rho_i}. \tag{4.5}$$

Suma w powyższym wzorze teoretycznie przebiega po wszystkich cząstkach w układzie. W praktyce jednak uwzglednia się jedynie te cząstki, dla których funkcja rozmycia przyjmuje wartości niezerowe. Przegląd wykorzystywanych funkcji rozmycia w metodzie SPH można znaleźć w pracy [99].

W historycznie pierwszej pracy dotyczącej SPH [58] Gingold i Monaghan jako funkcję rozmycia wykorzystali funkcję Gaussa:

$$W(r,h) = \frac{1}{h\sqrt{\pi}}\mathrm{e}^{-x^2/h^2}, \tag{4.6}$$

natomiast Lucy [90] w opisywanych przez siebie symulacjach trójwymiarowych korzystał z funkcji

$$W(r,h) = \frac{105}{16\pi h^3}\left(1+3\frac{r}{h}\right)\left(1-\frac{r}{h}\right)^3. \tag{4.7}$$

Jednak najczęściej spotykanymi w literaturze funkcjami rozmycia są funkcje sklejane Shoenberga M_n. Są to funkcje o zwartym nośniku, posiadające ciągłe pochodne do rzędu $n-2$. Definiuje się je przy pomocy transformacji Fouriera:

$$M_n(x,h) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\left(\frac{\sin kh/2}{kh/2}\right)^n \cos(kx)\mathrm{d}k, \tag{4.8}$$

a odpowiednie wzory algebraiczne podane zostały w pracach [122] oraz [96]. Przykładowo dla $n=2$, $M_2 = 1-r/h$ dla $0 \leqslant r \leqslant h$. Jednak funkcja ta ma nieciągłą pierwszą pochodną. Funkcją sklejaną, która ma pierwszą i drugą pochodne ciągłe jest funkcja M_4 o postaci:

$$M_4(x,h) = \begin{cases} \frac{1}{6}((2-r/h)^3 - 4(1-r/h)^3) & \text{, gdy } 0 \leqslant r \leqslant h, \\ \frac{1}{6}(2-r/h)^3 & \text{, gdy } h < r \leqslant 2h, \\ 0 & \text{, gdy } r > 2h. \end{cases} \tag{4.9}$$

Jest to powszechnie wystepująca w literaturze funkcja, będąca punktem odniesienia dla innych funkcji. Funkcje jądra oparte na funkcjach M_n muszą zostać jeszcze pomnożone przez odpowiednie czynniki. Aby zapewnić odpowiedni wymiar funkcji mnoży się ją przez czynnik $1/h^D$, gdzie D jest wymiarem przestrzeni. Dodatkowo, w każdym wymiarze należy jeszcze dokonać normalizacji funkcji rozmycia. W ogólności, funkcje sklejane oparte na M_n dla $n > 4$ nie przynoszą żadnych korzyści w porównaniu do $n = 4$ i dlatego nie znalazły szerszego zastosowania [99]. Dodatkowo, metoda dobierania odpowiedniej postaci funkcji jądra do konkretnego rodzaju symulacji została przedstawiona w pracy [27]. Dobór funcji jądra został w szerszym ujęciu przedstawiony w publikacjach [138, 116, 123].

W pracy przedstawiającej metodę SDPD Español zaproponował skorzystanie z funkcji określonej wzorem (4.7), przez co obecna w równaniach metody SDPD funkcja $F(r)$ przyjmuje postać:

$$F(r) = \frac{315}{4\pi h^5}\left(1 - \frac{r}{h}\right)^2. \tag{4.10}$$

Ponieważ metoda SDPD wciąż nie znalazła szerokiego zastosowania, nie pojawiły się jeszcze żadne prace prezentujące różne funkcje rozmycia i ich wpływ na przeprowadzane symulacje.

4.3 Konfiguracja początkowa

Podczas uruchamiania symulacji istotne jest początkowe ułożenie cząstek symulowanego układu. Najprostszym podejściem do tego problemu jest przypisanie im położeń w pudle obliczeniowym w sposób losowy. Wówczas odległości pomiędzy sąsiednimi cząstkami mają różne wartości. W przypadku implementacji metody SPH, w której korzysta się z wzoru (3.11) nadaje im się na początku symulacji jednakową gęstość. W rezultacie tego działania cząstki nie zmieniają swoich początkowych położeń, chociaż intuicja podpowiada, że wszelkie nieregularności w ich rozłożeniu wynikające z przypadkowości ich początkowych położeń powinny zniknąć. Możliwe są dwa rozwiązania tego problemu:

1. gęstości początkowe cząstek nie są jednakowe, ale wyznaczone ze wzoru (3.9),
2. położenia początkowe cząstek nie są losowe, ale stanowią węzły sieci regularnej.

Pierwsze podejście jest nieprawidłowe, gdyż dopuszcza duże względne różnice gęstości początkowych, co przy modelowaniu cieczy nieściśliwej jest nieakceptowalne. W przypadku takim siły wynikające z różnic gęstości są na tyle duże, że mogą w skrajnym przypadku powodować przerwanie symulacji z powodu błędów numerycznych. Natomiast drugie podejście jest dopuszczalne i często stosowane. Cząstki są umieszczone w węzłach siatki regularnej typu P i wszystkie mają przypisaną taką samą gęstość początkową. Dopuszcza się również małe zmiany losowe w gęstościach początkowych nie przekraczające 1%, aby od początku symulacji wymusić w ten sposób ruch wzajemny cząstek. Warunki te nie są aż tak rygorystyczne w innych metodach cząstek, jak np. w metodzie DPD.

4.4 Przebieg jednego kroku symulacji

W niniejszym podrozdziale zawarto opis poszczególnych bloków obliczeniowych, jakie znajdują się w pętli głównej algorytmu, przedstawionej schematycznie na rysunku (2.5). Poszczególne bloki zostały wymienione w kolejności takiej, w jakiej znajdują się w pętli głównej. Jej strukturę przedstawiono w sposób uproszczony we fragmencie kodu źródłowego (4.1).

Fragment kodu źródłowego 4.1: Przebieg pętli głównej.

```
for ( ; stepCounter < totalSteps ; ) {

  //---------------------------------------------------------------
  // GATHERING
  for ( i1=0 ; i1 < cells.size() ; i1++ ) {
    cells[i1]->prtcls26C.clear();

    for ( i2=0 ; i2 < 27 ; i2++ ) {
      tC = cells[i1]->neighsC[i2];
      if ( ( tC == NULL ) || ( tC == cells[i1] ) ) continue;
      bTmp = false;

      for ( i3=0; i3 < 3 ; i3++ ) {
        if ( tC->indC[i3] - cells[i1]->indC[i3] == 1 ) {
          bTmp = true;
          break;
```

```
        }
        if ( tC->indC[i3] - cells[i1]->indC[i3] == -1 ) {
          break;
        }
      }
      if ( bTmp )
        cells[i1]->prtcls26C.add( tC->prtclsC.ptr(), tC->prtclsC.size() );
    }
  }

  //---------------------------------------------------------------------
  // FORCES
  for ( i1 = 0 ; i1 < cells.size() ; i1++ ) {
    for ( i2=0 ; i2 < cells[i1]->prtclsC.size() ; i2++ ) {
      for ( i3=0 ; i3 < cells[i1]->fluidsC.size() ; i3++ )
        if ( cells[i1]->prtclsC[i2] < cells[i1]->prtclsC[i3] )
          force( cells[i1]->prtclsC[i2], cells[i1]->prtclsC[i3] );
      for ( i3=0 ; i3 < cells[i1]->prtcls26C.size() ; i3++ )
        force( cells[i1]->prtclsC[i2], cells[i1]->prtcls26C[i3] );
    }
    cells[i1]->prtclsC.clear()
  }

  //---------------------------------------------------------------------
  // MOTION
  for ( i1 = 0 ; i1 < prtcls.size() ; i1++ ) {
    for (i2=0 ; i2 < 3 ; i2++) {
      prtcls[i1]->v0[i2]  = prtcls[i1]->v[i2];
      prtcls[i1]->v[i2] = prtcls[i1]->v0[i2] + prtcls[i1]->a[i2] * dt;
      prtcls[i1]->vAB[i2] = 0.5*(3*prtcls[i1]->v[i2] - prtcls[i1]->v0[i2]);

      prtcls[i1]->x00[i2] = prtcls[i1]->x0[i2];
      prtcls[i1]->x0[i2]  = prtcls[i1]->x[i2];
      prtcls[i1]->x[i2] = prtcls[i1]->x0[i2] + prtcls[i1]->v[i2] * dt;

      prtcls[i1]->a[i2] = 0.0;
    }

    index = 0;
    for ( i2=0 ; i2 < 3 ; i2++ ) {
      dTmp = prtcls[i1]->x[i2] - xmin[i2];
      ncx[i2] = dTmp / cellSize;
      index += ncx[i2];
      if ( i2 < 2 ) index *= nc[i2+1];
    }
    prtcls[i1]->iamfrom = cells[index];
    cells[index]->prtclsC.push_back( prtcls[i1] );
  }

  //---------------------------------------------------------------------
  // STATISTICS

  for ( i1 = 0 ; i1 < prtcls.size() ; i1++ ) {
    ek = 0.0;
    for (us1=0 ; us1 < DIMENSION ; us1++ )
      ek += 0.25 * (prtcls[i1]->v[us1]+prtcls[i1]->v0[us1])
            * (prtcls[i1]->v[us1]+prtcls[i1]->v0[us1]);
    ek *= 0.5 * particleTypes[ prtcls[i1]->type ].mass;
    EKin += ek;
    Entropy += prtcls[i1]->ent;
    Temperature += prtcls[i1]->T();
  }

  Temperature /= prtcls.size();
  cout << "Ekin=" << EKin << "\nEntropy=" << Entropy
    << "\nTemperature=" << Temperature << endl;
  EKin = 0.0; Entropy = 0.0; Temperature = 0.0;

  //---------------------------------------------------------------------

  cout << "\nSTEP:␣" << stepCounter << "\n";
  stepCounter++;
```

```
90 } // main loop
```

Pierwszym blokiem pętli głównej algorytmu jest gromadzenie informacji o cząstkach z cel sąsiednich. W tym celu każda cela posiada specjalną tablicę o nazwie `prtcls26C` przeznaczoną do przechowywania wskaźników do cząstek znajdujących się w 26 sąsiednich celach. Cele te są dostępne poprzez wskaźniki z tablicy `neighsC`, która dla każdej celi jest tworzona na etapie inicjalizowania symulacji. W pętli z linii 5-25 tablica `prtcls26C` jest wypełniana odpowiednimi wartościami. Jest ona wykorzystywana w kolejnym bloku programu, w którym obliczane są wartości sił oddziaływania pomiędzy cząstkami. Wówczas korzysta się z niej podczas obliczania sił dla każdej cząstki aby ograniczyć liczbę potencjalnych cząstek, z którymi rozpatrywana cząstka oddziałuje. Korzystanie z tej tablicy nie jest jednak konieczne. Możliwe jest wyznaczanie potencjalnych cząstek sąsiednich dla każdej cząstki na bieżąco, w momencie gdy przystępuje się do obliczania wartości sił oddziaływania z jej cząstkami sąsiednimi. Jednak o wiele efektywniej jest wykonać takie obliczenia tylko raz, co ma miejsce w opisywanym bloku pętli głównej.

Aby nie powtarzać dwukrotnie tych samych obliczeń można wykorzystać fakt, że w opisywanych symulacjach spełniona jest trzecia zasada dynamiki Newtona. Przykładowo, podczas obliczania siły działającej na cząstkę o numerze i oblicza się jej oddziaływanie z cząstką o numerze j. Jednak wartość siły wynikającej z tego oddziaływania działającej na cząstkę j jest taka sama jak dla siły działającej na cząstkę i. Nie jest więc konieczne ponowne obliczanie wartości tego oddziaływania podczas wyliczania całkowitej siły działającej na cząstkę j. Oddziaływanie pomiędzy cząstkami i oraz j wystarczy obliczyć tylko raz.

Istnieje wiele możliwych uporządkowań, według których można decydować, kiedy dla rozpatrywanej pary cząstek należy obliczyć oddziaływanie, a kiedy skorzystać z wartości już wcześniej obliczonej. Przykładowo, można cząstki uporządkować względem ich numerów. Wówczas, w pętli przebiegającej po wszystkich cząstkach o liczniku k oddziaływanie dla pary cząstek (k, j) oblicza się tylko w przypadku, gdy $k < j$. Jednak w przypadku symulacji korzystającej ze struktury sześciennych cel dla przypadku cząstek z różnych cel, wygodnie jest oprzeć uporządkowanie na numerach cel. Oddziaływanie pomiędzy cząstkami i oraz k oblicza się, gdy $cell(i) < cell(k)$, gdzie $cell(i)$ oznacza numer celi, w której znajduje się cząstka i. Dla aktualnie rozpatrywanej celi zostało to przedstawione na rysunku (2.4): kolorem szarym zostały oznaczone cele zawierające cząstki, dla których oblicza się oddziaływania. Wyznaczenie tych cel odbywa sie w liniach 11-23 powyższego kodu źródłowego, poprzez porównywanie współrzędnych cel sąsiednich ze współrzędnymi aktualnie rozpatrywanej celi (tablica `indC`). W przypadku, gdy cela `tC` ma numer mniejszy, to zmienna logiczna `bTmp` jest ustawiana jako prawdziwa i od jej wartości zależy, czy wskaźniki cząstek znajdujące się w celi `tC` zostaną skopiowane do tablicy `prtcls26C` aktualnie rozpatrywanej celi (linia 23).

Kolejnym blokiem programu jest obliczanie sił pomiędzy cząstkami. Fragment ten został zaimplementowany w liniach 29-38. Dla każdej cząstki (indeks `i2`) z każdej celi (indeks `i1`) przeprowadza się obliczenia w dwóch etapach. W pierwszym etapie (linie 31-33) oblicza się siły pochodzące od cząstek z tej samej celi. Aby nie obliczać tego samego oddziaływania dwukrotnie dla każdej pary cząstek, wykorzystuje się uporządkowanie cząstek względem ich numerów. W drugim etapie (linie 34-35) oblicza sie oddziaływania cząstki z cząstkami z cel sąsiednich. W tym celu korzysta sie z wcześniej wypełnionej tablicy `prtcls26C`.

W następnym bloku programu (linie 42-64), na podstawie obliczonych sił i przyspieszeń, i poprzednio wyznaczonych prędkości nadaje się cząstkom nowe położenia oraz wyznacza nowe prędkości. W tym celu stosuje się wybrany schemat całkowania (schematy całkowania zostały omówione w podrozdziale 3.6). We fragmencie 4.1 w liniach 44-50 przedstawiony został schemat Adamsa-Bashfortha [24]. Następnie, na podstawie nowych wartości poło-

żenia cząstki, wyznacza się indeks celi, w której cząstka się znajduje (linie 55-61). Na tej podstawie uaktualnia się odpowiednie zmienne zarówno dla cząstki (l. 62) jak i dla celi (l. 63).

Ostatnim blokiem programu jest wyznaczanie statystyk i zmiennych biegnących z całego układu cząstek (ll. 69-83). W przykładzie przedstawionym we fragmencie 4.1 są to: energia kinetyczna układu, jego entropia i temperatura. W celu ich obliczenia, do zmiennych globalnych dodaje się odpowiednie składniki pochodzące od każdej z cząstek. W przypadku temperatury, która jest zmienną intensywną, konieczne jest również podzielenie tak otrzymanej sumy przez liczbę cząstek w układzie (l. 80).

Wydruk 4.1 jest uproszczonym przykładem kodu źródłowego symulacji, który przedstawia jedynie ogólny zarys struktury pętli głównej. W rzeczywistości, w kodzie źródłowym uwzględniono możliwość występowania cząstek różnych typów, a tym samym implementacji różnych oddziaływań zależnych od typów cząstek z pary. Dodakowo, możliwe jest wprowadzenie cząstek ścian modelujących rzeczywiste warunki brzegowe wystepujące w modelowanym zjawisku. Przedstawienie wszystkich możliwych przypadków uwzgledniannych w kodzie źródłowym, a które nie są istotne z rozpatrywanego tu punktu widzenia, zwiększyłoby znacząco rozmiar wydruku 4.1 i dlatego zostało pominięte.

4.5 Porównanie metod znajdowania sąsiadów cząstek stosowanych w metodzie SPH

W metodzie SPH istnieje kilka sposobów definiowania relacji sąsiedztwa pomiędzy cząstkami. Podstawowym parametrem, przy pomocy którego się tego dokonuje, jest odległość wygładzania h (ang. smoothing length). Już w pierwszej pracy Monaghana i Gingolda [58] stała dla wszystkich cząstek wielkość h zmieniała się wraz z czasem symulacji, gdy średnia odległość pomiędzy cząstkami ulegała zmianie. Wszystkie sposoby definiowania sąsiedztwa w metodzie SPH starają się dostosowywać wartość promienia wygładzania h do „rozdzielczości" metody, która związana jest z gęstością. Najpopularniejszym sposobem definiowania sąsiedztwa w metodzie SPH jest takie określenie wartości h, aby każda cząstka miała taką samą, z góry ustaloną, liczbę cząstek z nią oddziałujących [70]. Uzyskuje się dzięki temu mechanizm automatycznego dostosowania promienia oddziaływania do lokalnej gęstości. Jeśli cząstek jest lokalnie dużo i dzięki temu wartość gęstości płynu również jest duża, wówczas promień oddziaływania w tym obszarze staje się mały. W przeciwnym przypadku, gdy liczba cząstek jest lokalnie mała i gęstość płynu również jest mała, promień oddziaływania staje się duży. Powyższy sposób określania sąsiedztwa został z powodzeniem zastosowany do symulacji astrofizycznych [113]. W takich symulacjach gęstość modelowanego płynu (a więc również cząstek) przyjmuje wartości z dużego przedziału wartości. W takim przypadku stosowanie relacji sąsiedztwa opartej na stałej liczbie sąsiadów i zmiennej wartości promienia sąsiedztwa jest uzasadnione.

Jak wspomniano w podrozdziale 3.3, metodę SPH można również stosować do symulacji płynów nieściśliwych. W symulacjach takich gęstość płynu nie zmienia się znacząco od wielkości ustalonej. Powstaje więc pytanie, czy konieczne jest stosowanie definicji sąsiedztwa opartej na stałej liczbie sąsiadów. Być może możliwe jest skorzystanie z definicji sąsiedztwa opartej, podobnie jak w większości innych bezsiatkowych metod cząstek, na stałym promieniu obcięcia r_{cut} (w metodzie SPH $r_{cut} = 2h$). Można się spodziewać, że w przypadku modelowania płynów nieściśliwych takie podejście powinno dać bardzo podobne rezultaty jak sąsiedztwo oparte na stałej ilości sąsiadów. Dodatkowo, taki sposób definiowania sąsiedztwa pozwala na wykorzystanie efektywnych metod znajdowania sąsiadów cząstek stosowanych dla potencjałów krótkozasięgowych.

W dalszej części przedstawione zostały obydwie definicje sąsiedztwa jak również porównanie ich implementacji, złożoności obliczeniowych oraz uzyskanych wyników dla przypadków klasycznych symulacji.

4.5.1 Definicja sąsiedztwa oparta na stałej liczbie sąsiadów

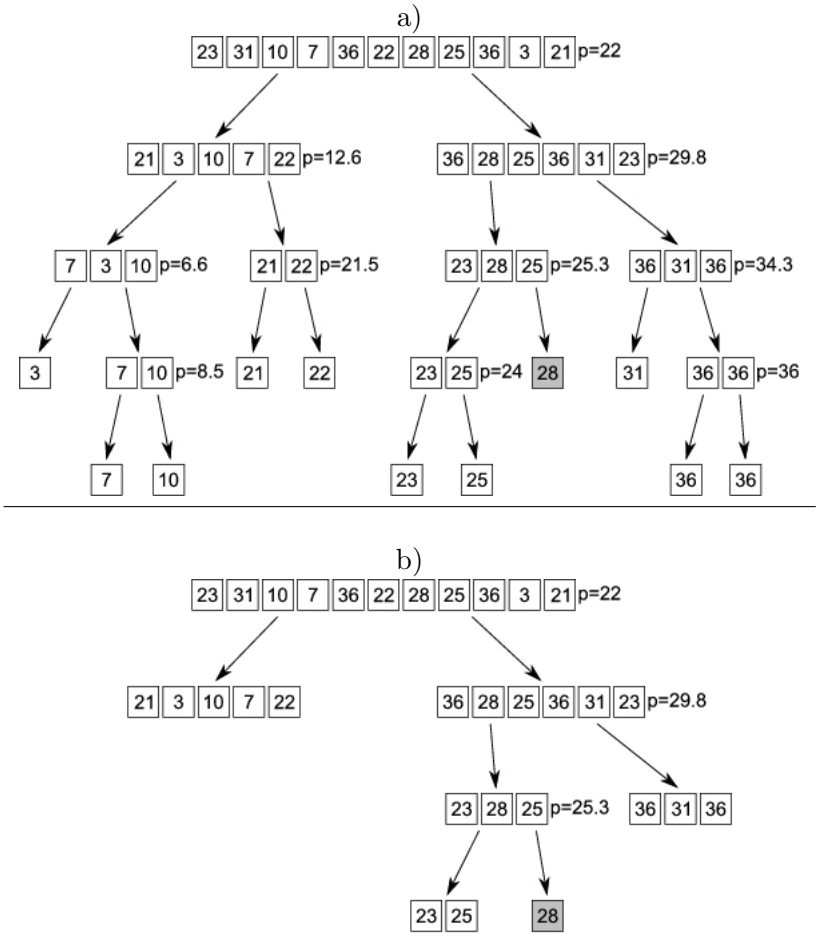
Poniżej przedstawiony został pseudokod algorytmu wyszukiwania sąsiadów przy założeniu, że relacja sąsiedztwa pomiędzy cząstkami oparta jest na stałej liczbie sąsiadów.

```
NEIGHBORS()
1  for i ← 1 to nParticles
2      do
3          potentialNeighbors ← getPotentialNeighbors(i)
4          distances ← distance(i, potentialNeighbors)
5          SortN(neighNum, potentialNeighbors, distances)
6          neighbors ← potentialNeighbors[1, neighNum]
7
```

W powyższym algorytmie zmienna *nParticles* oznacza liczbę cząstek w układzie. W pierwszym kroku procedura *getPotentialNeighbors(i)* tworzy listę potencjalnych sąsiadów dla cząstki o numerze i, która jest zapisywana w tablicy *potentialNeighbors*. Następnie procedura *distance(i,potentialNeighbors)* tworzy tablicę odległości pomiędzy czastką i a cząstkami z tablicy *potentialNeighbors*, która jest zapisywana do tablicy *distances*. Następnie, procedura *SortN(neighNum, potentialNeighbors, distances)* sortuje tablice *potentialNeighbors* oraz *distances* na podstawie wartości z tablicy *distances*. Procedura *SortN* nie sortuje tablic, które otrzymuje jako swoje argumenty, w sposób dosłowny. Zamiast tego przestawia ona ich elementy w następujący sposób: na miejscu o numerze *neighNum* znajduje się odpowiadający temu numerowi element $x[neighNum]$ względem ustalonego uporządkowania. Na pozycjach $1 \ldots neighNum - 1$ znajdują się elementy mniejsze od $x[neighNum]$. Natomiast na pozycjach $neighNum + 1 \ldots length(potentialNeighbors)$ znajdują się elementy większe od $x[neighNum]$. W obydwu częściach, $1 \ldots neighNum - 1$ oraz $neighNum + 1 \ldots length(potentialNeighbors)$, elementy nie są posortowane. Wyniki działania metody *SortN* zostały zobrazowane na rysunku (4.1). Szczegóły implementacji procedury *SortN* są przedstawione w pracy [113]. Ostatecznie, pierwsze *neighNum* elementów z tablicy *potentialNeighbors* jest przypisywane tablicy *neighbors*, która od tego momentu zawiera listę cząstek, z którymi oddziałuje cząstka *i*.

Fakt, że w metodzie tej każda cząstka ma stałą liczbę sąsiadów powoduje, że cząstki mają różne zasięgi oddziaływań. Dla każdej cząstki zasięg ten jest obliczany adaptacyjnie w każdym kroku czasowym.

Funkcja *getPotentialNeighbors* zwraca tablicę zawierającą wszystkie cząstki, które potencjalnie mogą być sąsiadami danej cząstki. Teoretycznie więc tablica ta powinna zawierać wszystkie cząstki jakie znajdują sie w układzie. W przedstawianym porównaniu jednak tak nie jest. Ponieważ pudło obliczeniowe podzielone jest na regularne cele, struktura tych cel może być wykorzystana do poszukiwania cząstek sąsiednich. Dlatego też funkcja *getPotentialNeighbors* zwraca jedynie numery tych cząstek, które znajdują się w otoczeniu celi, do której należy rozpatrywana cząstka. W przypadku, gdy w tym otoczeniu byłaby zbyt mała liczba cząstek, mogłoby to powodować zaburzenie działania metody. Jednak w przypadku cieczy nieściśliwych, dla których wykonywane jest porównanie, przypadek ten jest praktycznie niemożliwy.



Rysunek 4.1: Schemat działania procedury *SortN*: a) tradycyjny quicksort; b) procedura *SortN*.

4.5.2 Definicja sąsiedztwa oparta na stałym promieniu obcięcia

Poniżej przedstawiony został algorytm wyszukiwania sąsiadów przy założeniu, że relacja sąsiedztwa pomiędzy cząstkami oparta jest na stałym promieniu obcięcia r_{cut}, to znaczy sąsiadami rozpatrywanej cząstki są wszystkie inne, do których ogległość jest mniejsza od r_{cut}.

```
NEIGHBORS()
1  for i ← 1 to nParticles
2      do
3          potentialNeighbors ← getPotentialNeighbors(i)
4          distances ← distance(i, potentialNeighbors)
5          neighbors ← cutoff(potentialNeighbors, distances, r_cut)
6
7
```

W porównaniu do algorytmu bazującego na definicji opartej na stałej liczbie sąsiadów, brak w powyższym algorytmie funkcji *SortN*. Zamiast niej obecna jest linia, w której do tablicy *neighbors* wpisywane są wszystkie cząstki z tablicy *potentialNeighbors*, których odległość obliczona i zapisana w tablicy *distances* jest mniejsza lub równa od promienia obcięcia r_{cut}. Zadanie to jest wykonywane w procedurze *cutoff*. Dodatkowo, procedury *distance* i *cutoff* mogą zostać połączone i wykonane tylko w jednej pętli przebiegającej tablicę *potentialNeighbors*.

Fakt, że cząstki w tej metodzie mają ten sam promień obcięcia powoduje, że mogą one mieć różne liczby sąsiadów. Ponieważ gęstość cząsteczkowa może się zmieniać zarówno w czasie jak i w przestrzeni, to również liczba cząstek w kuli o promieniu r_{cut} i środku w rozpatrywanej cząstce (czyli jej sąsiadów) będzie się zmieniała.

W przypadku definicji opartej na stałym promieniu obcięcia relacja sąsiedztwa, (x jest sąsiadem y) $\Leftrightarrow$ ($y \leftharpoonup x$), pomiędzy dwoma cząstkami jest symetryczna:

$$(y \leftharpoonup x) \Longrightarrow (x \leftharpoonup y). \tag{4.11}$$

Niestety nie jest to prawdą dla definicji opartej na stałej liczbie sąsiadów. Przy zastosowaniu tej definicji nie ma ustalonego, globalnego promienia obcięcia dla wszystkich cząstek. Zamiast tego każda cząstka ma swój własny promień obcięcia ($r_{cut,i} = 2h_i$), który dodatkowo może się zmieniać wraz z postepem symulacji. Może się więc zdarzyć sytuacja, w której dla dwóch cząstek o numerach i oraz j będzie zachodzić:

$$r_i < r_{ij} < r_j, \tag{4.12}$$

co jest równoważne

$$(j \leftharpoonup i) \wedge \neg(i \leftharpoonup j). \tag{4.13}$$

Relacja sąsiedztwa w tym przypadku jest niesymetryczna. Wynikają z tego pewne trudności: ponieważ cząstka i jest sąsiadem cząstki j, wywiera ona na tę cząstke pewną siłę. Jednak cząstka j nie wywiera siły reakcji na cząstkę i, gdyż nie należy ona do jej zbioru sąsiadów. Jest to niezgodne z trzecią zasadą dynamiki Newtona i powoduje naruszenie zasady zachowania pędu w modelowanym układzie. Również w przypadku, gdy dwie cząstki są wzajemnie swoimi sąsiadami, lecz mają różne promienie obcięcia, trzecia zasada dynamiki Newtona nie jest zachowana. W przypadku takim cząstka j działa na cząstkę i siłą o wartości:

$$F_{ij} = m_i m_j W'(r_{ij}, h_j) \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \Pi_{ij} \right), \tag{4.14}$$

natomiast cząstka i działa na cząstkę j siłą o wartości:

$$F_{ji} = m_j m_i W'(r_{ij}, h_i) \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \Pi_{ji} \right). \tag{4.15}$$

Siły te są różne, gdyż $W'(r_{ij}, h_j) \neq W'(r_{ij}, h_i)$. Także w tym przypadku trzecia zasada dynamiki Newtona nie jest zachowana.
Aby zasada ta była zachowana, w przeprowadzanej symulacji stosuje się różne modyfikacje. Najczęściej spotykane polegają na symetryzacji funkcji jądra albo poprzez obliczanie średniej z wielkości h dla obydwu cząstek [13]:

$$W_{ij} = W\left(r_{ij}, \frac{h_i + h_j}{2} \right), \tag{4.16}$$

albo poprzez obliczenie średniej z wartosci funkcji jądra dla obydwu cząstek [70]:

$$W_{ij} = \frac{1}{2} \left(W(r_{ij}, h_i) + W(r_{ij}, h_j) \right). \tag{4.17}$$

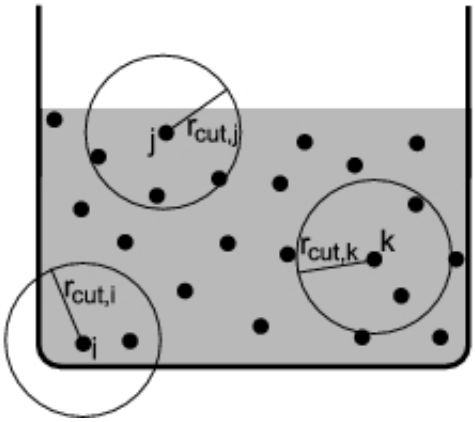
Dokładniejszy opis tego problemu znajduje się w [53]. Obydwa powyższe warianty dają w przybliżeniu takie same rezultaty [92].

4.5.3 Zachowanie cząstek przy powierzchni swobodnej i granicy płyn - naczynie

Przy zastosowaniu definicji opartej na stałej liczbie sąsiadów cząstki znajdujące się w pobliżu powierzchni badź granicy cieczy mają większy zasięg oddziaływania, niż cząstki znajdujące się daleko od powierzchni cieczy, w wewnętrznym jej obszarze. Jest to konsekwencją dążenia cząstek do tego, aby w swoim zasięgu oddziaływania mieć taką samą liczbę sasiadów. Ponieważ przy powierzchni cieczy przynajmniej część kuli (lub koła w przypadku dwuwymiarowym) o środku w rozpatrywanej cząstce i o promieniu równym jej zasięgowi oddziaływania jest pusta (nie zawiera czastek), dlatego też promień oddziaływania musi zostać zwiększony, aby w kuli tej znajdowała się ustalona liczba cząstek. Problem ten został przedstawiony na rysunku (4.2).

4.5.4 Rozmiar cel

W obydwu definicjach podczas określania, czy pomiędzy dwoma cząstkami zachodzi relacja sąsiedztwa, konieczne jest obliczenie odległości pomiędzy nimi. Obliczenia te mogą mieć znaczący wpływ na czas wykonania symulacji prowadząc do jego nieakceptowalnego wydłużenia, jeśli odległości należy policzyć dla zbyt dużej liczby par cząstek. Jak już wspomniano, w celu ograniczenia liczby par, dla których należy obliczyć odległość, wykorzystuje się strukturę regularnych cel. Rozmiar cel musi być na tyle duży, aby zbiór cząstek sąsiednich każdej cząstki zawierał się w sześciennym bloku cel o boku długości trzy razy większej niż rozmiar cel. Należy jednak tak dobrać rozmiar cel, aby był on na tyle mały, aby liczba par cząstek, dla których należy obliczyć odległości była jak najmniejsza. W przypadku definicji opartej na stałym promieniu obcięcia oczywiste jest, że rozmiar celi powinien być równy największemu promieniowi obcięcia występującemu podczas symulacji. W przypadku mniejszego rozmiaru celi istnieje możliwość, że niektóre pary cząstek, dla których $r_{ij} < r_{cut}$, nie będą rozpatrywane jako wzajemnie sąsiednie. Gdy natomiast rozmiar



Rysunek 4.2: Różne ilości płynów dla cząstek o numerach i, j, k w kulach o promieniach odpowiadajacych im promieniom obcięcia.

celi będzie większy, wówczas przy obliczaniu odległości będą brane pod uwagę pary czastek, dla których z góry wiadomo, że zachodzi $r_{ij} > r_{cut}$, co niepotrzebnie wydłuży czas obliczeń. W przypadku definicji opartej na stałej liczbie sąsiadów należy również wykorzystać strukturę regularnych cel. W tym przypadku jednak określenie rozmiaru cel nie jest tak oczywiste. Pierwszym przybliżeniem wartości rozmiaru celi może być wartość promienia obcięcia obliczona ze wzoru (4.18). Ponieważ jednak sąsiedzi dla konkretnej cząstki mogą być rozmieszczeni wokół niej w sposób nieregularny, mogą zachodzić trzy sytuacje:

1. w otaczających 27 celach znajduje się mniej niż N cząstek,
2. w otaczających 27 celach znajduje się co najmniej N czastek, jednak w przypadku zwiększenia rozmiaru cel, lista sąsiadów uległa by zmianie. Może to mieć miejsce, gdy rozmieszczenie cząstek w otaczających 27 celach jest niesymetryczne,
3. w otaczających 27 celach znajduje się co najmniej N cząstek, i wyznaczona dla nich lista sąsiadów nie ulegnie zmianie w przypadku zwiększenia rozmiaru cel.

Ostatni przypadek, najbardziej pożądany, niestety nie zawsze jest prawdziwy dla rozmiaru cel równego r_N wynikającego z równania (4.18). Dlatego też, podczas porównywania obydwu sposobów wyznaczania cząstek sąsiednich, rozmiar cel jest równy $1.26r_N$. Dzięki temu objętość celi jest dwukrotnie większa, niż w przypadku r_N. Oczywiście, wartość ta jest pewnym kompromisem. Warto przypomnieć, że objętość kuli zawierającej N sąsiadów powinna być nawet cztery bądź osiem razy większa dla cząstek położonych przy krawędziach lub w narożnikach pudła obliczeniowego.

4.5.5 Sposób porównania definicji sąsiedztwa

Dla porównania obydwu definicji sąsiedztwa uruchomiono dla każdej z nich serię symulacji tych samych zjawisk. Przed porównaniem definicji należało jednak dobrać odpowiednie wartości parametrów z jakimi obie symulacje były uruchamiane. W szczególności należało dopasować liczbę sąsiadów N w metodzie ze stałą liczbą sąsiadów oraz promień obcięcia r_{cut} w metodzie ze stałym promieniem obcięcia. Może się bowiem okazać, że dla ustalonego

parametru h (definicja oparta na stałym promieniu obcięcia) średnia liczba sąsiadów dla cząstek różni się znacznie od wartosci ustalonej N dla drugiego z porównywanych sposobów. Na problem ten można również spojrzeć z drugiej strony i zauważyć, że dla ustalonego parametru N w metodzie ze stałą liczbą sąsiadów wartości parametru h mogą znacznie odbiegać od wartosci ustalonej h w drugiej z porównywanych metod. W takim przypadku porównanie czasowe obydwu definicji byłoby bezcelowe, gdyż ma ono sens jedynie wtedy, gdy wartości danych wejściowych (w tym przypadku liczba sąsiadów) są takie same w obydwu przypadkach.

Dlatego też, oprócz ustalenia wartości parametrów symulacji takich, jak współczynnik lepkości, wielkość kroku czasowego, itp. konieczne było dobranie wartości N oraz r_{cut} w uruchamianych symulacjach. Jeśli objętość płynu przypisana cząstce SPH, ocznaczana jako V_{SPH}, jest znana, relacja pomiędzy powyższymi parametrami ma postać:

$$\frac{4}{3}\pi r_{cut}^3 = NV_{SPH}. \tag{4.18}$$

Równanie to może być zweryfikowane przy pomocy symulacji. W tym celu uruchomiono kilka symulacji dla metody ze stałą liczbą sąsiadów N i dzięki uzyskanym wynikom dla każdej z nich wyznaczono wartość średnią promienia obcięcia. Podobnie jak poprzenio, uruchomiono kilka symulacji dla metody ze stałym promieniem obcięcia i dla każdej z nich wyznaczono średnią liczbę sąsiadów. Wyniki przedstawiono na rysunku (4.3). Na rysunku tym zamieszczono trzy wykresy:

1. $r_{cut}(N)$ - wykres zależności promienia obcięcia r_{cut} od liczby sąsiadów N otrzymany z serii symulacji uruchomionych dla metody ze stałą liczbą sąsiadów,

2. $N(r_{cut})$ - wykres zależności liczby sąsiadów N od promienia obcięcia r_{cut} otrzymany z serii symulacji uruchomionych dla metody ze stałym promieniem obcięcia,

3. th - wykres zależności określonej wzorem (4.18).

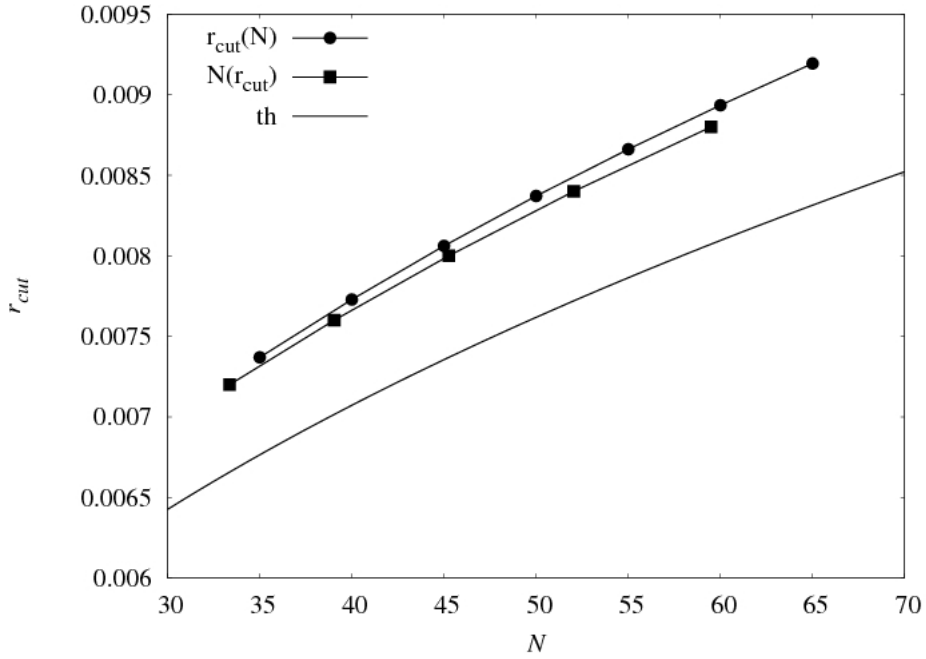
Dwa wykresy na rysunku (4.3) powstałe z danych empirycznych praktycznie się pokrywają, podczas gdy wykres zależności (4.18) różni się od nich znacząco. Najprawdopodobniej jest to związane z tym, że zależność teoretyczna nie obejmuje wszystkich czynników, które są obecne w realizacji numerycznej. Główną tego przyczyną powinna być zależność pomiędzy N a r_{cut} w pobliżu granic i powierzchni cieczy, kiedy to liczba cząstek w kuli o promieniu r_{cut} jest mniejsza, niż mogłoby to wynikać z równania (4.18).

Podczas porównywania obydwu metod wykorzystano zależności otrzymane drogą empiryczną. Mają one tę przewagę nad zależnością opisaną wzorem (4.18), że uwzględniają wszystkie czynniki i efekty, które pojawiają się przy realizacji numerycznej symulacji, a które nie są uzwględnione w zależności teoretycznej. Zależności te jednak nie mogłyby być wykorzystywane przy porównywaniu dowolnych symulacji. Powodem jest to, że nie będą one odpowiadać sytuacji, w której kształt naczynia i zarazem wypełniającego go płynu zmieni się na tyle, że stosunek objętości płynu do jego powierzchni zmieni się znacząco. W takim przypadku zależności te muszą być wyznaczone ponownie.

Do dwóch pokrywających się wykresów z rysunku (4.3) dopasowano krzywą opisaną równaniem:

$$N = 1.33 \cdot 10^8 \cdot r_{cut}^3 - 1.42 \cdot 10^6 \cdot r_{cut}^2 + 1.3 \cdot 10^4 \cdot r_{cut} - 37. \tag{4.19}$$

Powyższa relacja została wykorzystana podczas porównywania obydwu metod.



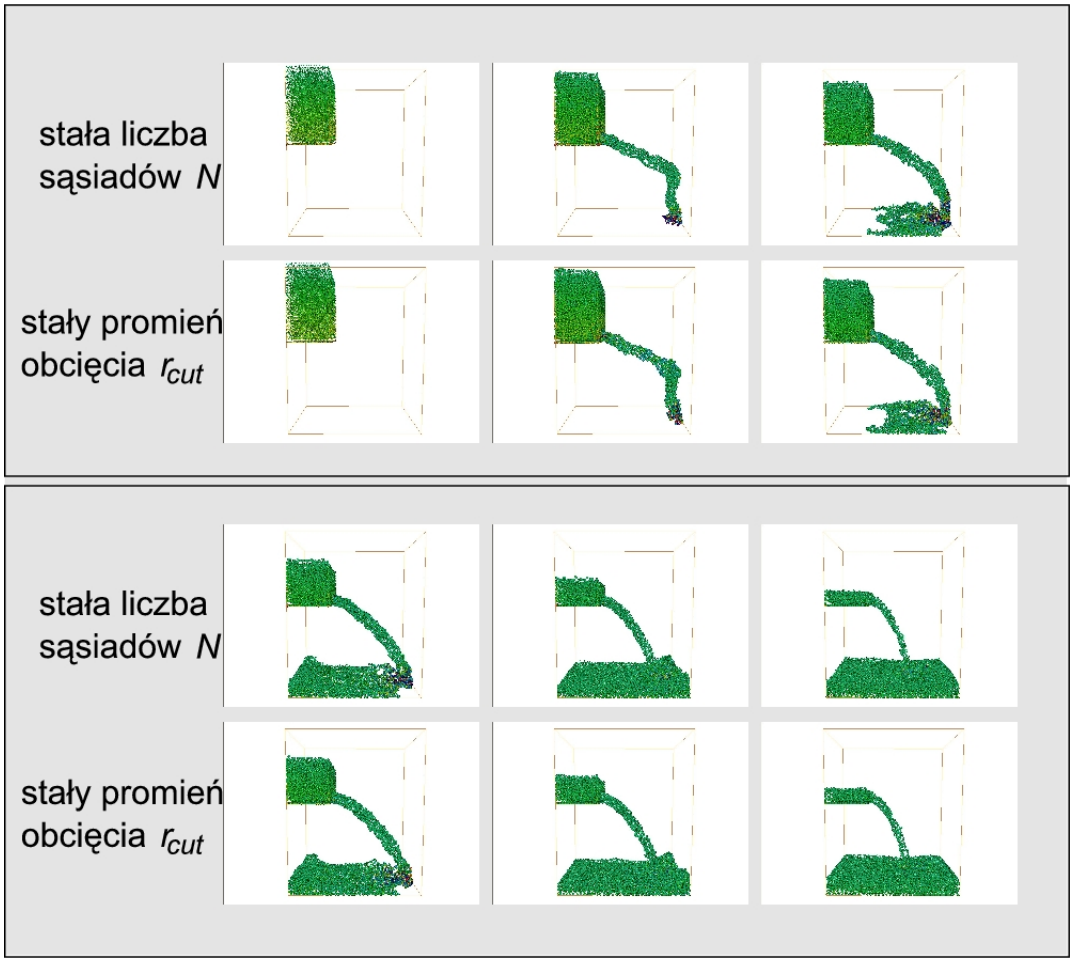
Rysunek 4.3: Wzajemna zależność pomiędzy wartością promienia obcięcia r_{cut} (w jednostkach programowych) a liczbą sąsiadów N.

4.5.6 Porównanie jakościowe definicji sąsiedztwa

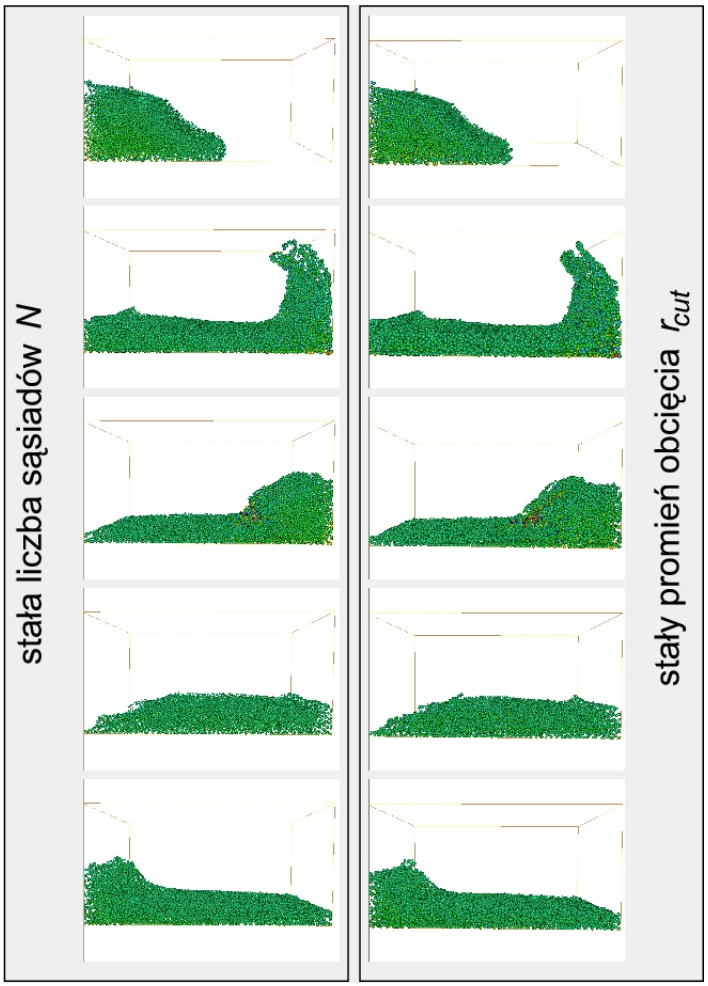
Na rysunkach (4.4) oraz (4.5) przedstawione są odpowiadające sobie kroki czasowe symulacji otrzymanych dla dwóch różnych definicji sąsiedztwa. Na rysunku (4.5) przedstawione zostało zjawisko przerwania tamy, natomiast na rysunku (4.4) przedstawiono modelowanie wypływu cieczy z naczynia. Zjawisko przerwania tamy powszechnie występuje w literaturze i jest używane jako przypadek testowy sprawdzający poprawność implementacji i modyfikacji metody [34, 98]. Korzystając z obydwu zestawu rysunków można jakościowo porównać wyniki otrzymane w wyniku zastosowania rozpatrywanych definicji sąsiedztwa. Niewielkie różnice widoczne na rysunkach są pomijalne dla modelowanych zjawisk. Na rysunkach tych gęstości cząstek oznaczono kolorem. Każdy kolor, od ciemno-niebieskiego do czerwonego ma przypisaną wartość gęstości. Kolor czerwony oznacza gęstość wysoką. Gęstości w przedstawionych symulacjach przyjmowały wartości od 995 kg/m^3 do 1005 kg/m^3. Pomimo, że modelowany płyn w założeniu jest nieściśliwy, gęstości na rysunku różnią się w widoczny sposób od ustalonej, zadanej gęstości równej 1000 kg/m^3 - widoczne sa zarówno ciemnoniebieskie, jak i czerwone cząstki. Jest to konsekwencją samej natury metody obliczeniowej wykorzystanej do symulacji oraz użytego wraz z nią równania stanu. Siła hydrodynamiczna zależy od stosunku gęstości cząstki do gęstości ustalonej. Gdyby obie gęstości były sobie równe, wówczas siła hydrodynamiczna nie występowałaby.

Zgodnie z przyjętym równaniem stanu (3.12) ciśnienie hydrodynamiczne zależy od przyjętej prędkości dźwięku w modelowanym płynie. W symulacjach przedstawionych na rysunkach (4.4) oraz (4.5) celowo przyjęto wartość prędkości dźwieku mniejszą niż w rzeczywistości (15m/s zamiast 1500m/s). Rzeczywista wartość prędkości dźwięku wymuszałaby stosowanie kroku czasowego zbyt małego, aby uzyskać pożądane zaawansowanie symulacji w akceptowalnym czasie obliczeń [34].

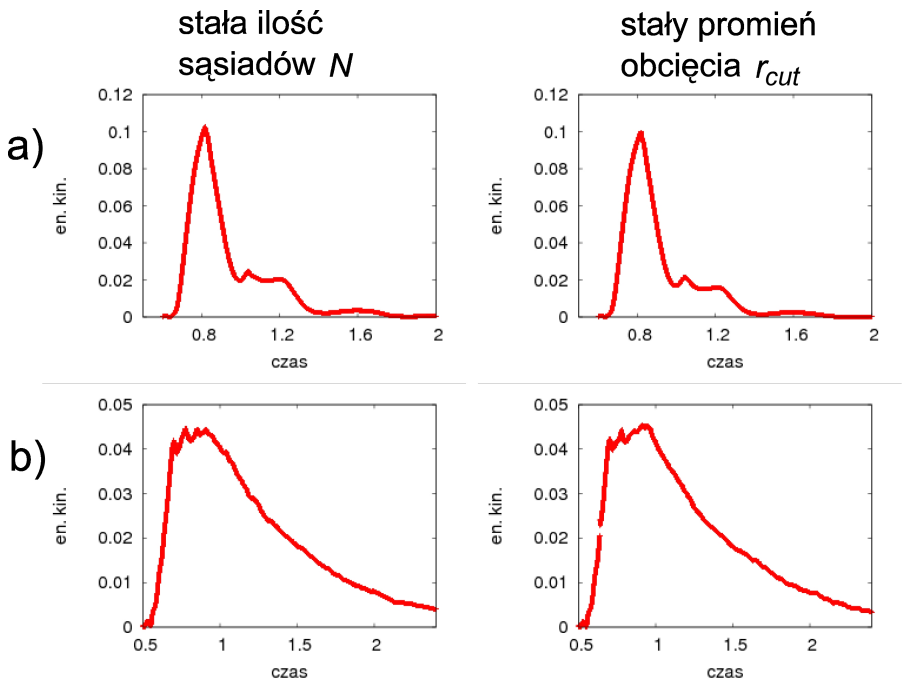
Rysunek (4.6) zawiera natomiast wykresy energii kinetycznych symulowanych układów. Z ich porównania również widać brak istotnych różnic porównywanych metod.



Rysunek 4.4: Jakościowe porównanie wyników symulacji dla dwóch różnych metod znajdowania sąsiadów. Symulacja wypływania cieczy z naczynia.



Rysunek 4.5: Jakościowe porównanie wyników symulacji dla dwóch różnych metod znajdowania sąsiadów. Symulacja przerwania tamy.



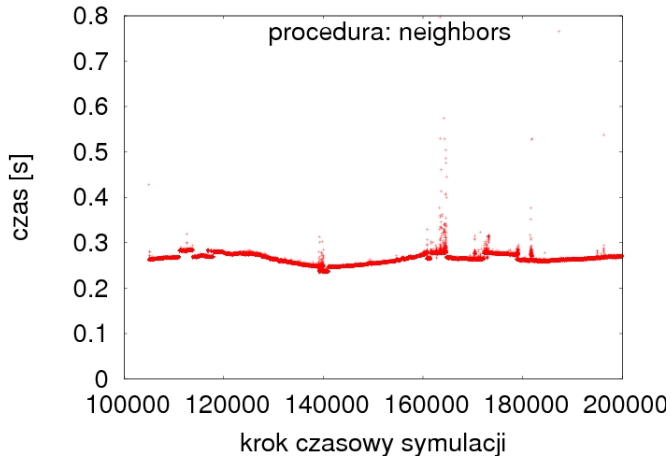
Rysunek 4.6: Porównanie wykresów energii kinetycznych modelowanych układów: a) przerwanie tamy, b) wypływ cieczy z naczynia.

4.5.7 Wydajność obliczeniowa porównywanych definicji sąsiedztwa

Oprócz porównania jakościowego symulacji celowe jest porównanie obydwu definicji pod względem czasu obliczeń. W tym celu w odpowiednich miejscach kodu źródłowego programu symulacji umieszczono wywołania funkcji `gettimeofday()`, dzięki którym możliwy był pomiar rzeczywistego czasu wykonania głównych bloków w każdej iteracji symulacji. Jednym z takich bloków jest procedura wyszukiwania sąsiadów, której czas wykonania mierzono podczas trwania symulacji, a następnie przedstawiono na rysunku (4.7). Pomiary dokonano podczas symulacji zjawiska przerwania tamy. Z rysunku tego widać, że czas wykonania procedury znajdowania sąsiadów jest prawie stały przez cały czas trwania symulacji i nie zależy od aktualnej dynamiki modelowanego zjawiska. Na tej podstawie założono, że czas wykonania dowolnego bloku głównej pętli symulacji może być zmierzony w dowolnym kroku czasowym.

W celu porównania czasu wykonania uruchomiono dwie symulacje z takimi samymi parametrami, jednak dla różnych definicji sąsiedztwa. Liczba sąsiadów oraz promień obcięcia zostały dobrane na podstawie zależności (4.19). Główną pętlę algorytmu symulacji podzielono na pięć bloków, które przedstawiono poniżej:

1. `moving` - wyznaczanie nowych położeń i prędkości cząstek,
2. `collecting` - procedura pomocnicza przyporządkowująca cząstki do cel, w których aktualnie się znajdują,
3. `neighbors` - dla każdej cząstki wyznaczana jest lista jej sąsiadów,
4. `force-density` - procedura obliczająca oddziaływania pomiędzy cząstkami oraz ich gęstości,
5. `stats` - procedura pomocnicza, obliczająca średnie statystyczne i wypisująca je na standardowe wyjście .



Rysunek 4.7: Czas wykonania procedury wyszukującej sąsiadów na przestrzeni całego okresu trwania symulacji.

Dla każdego z powyższych bloków zmierzono czas jego wykonania w obydwu uruchomionych symulacjach. Wyniki zamieszczono na rysunku (4.8).

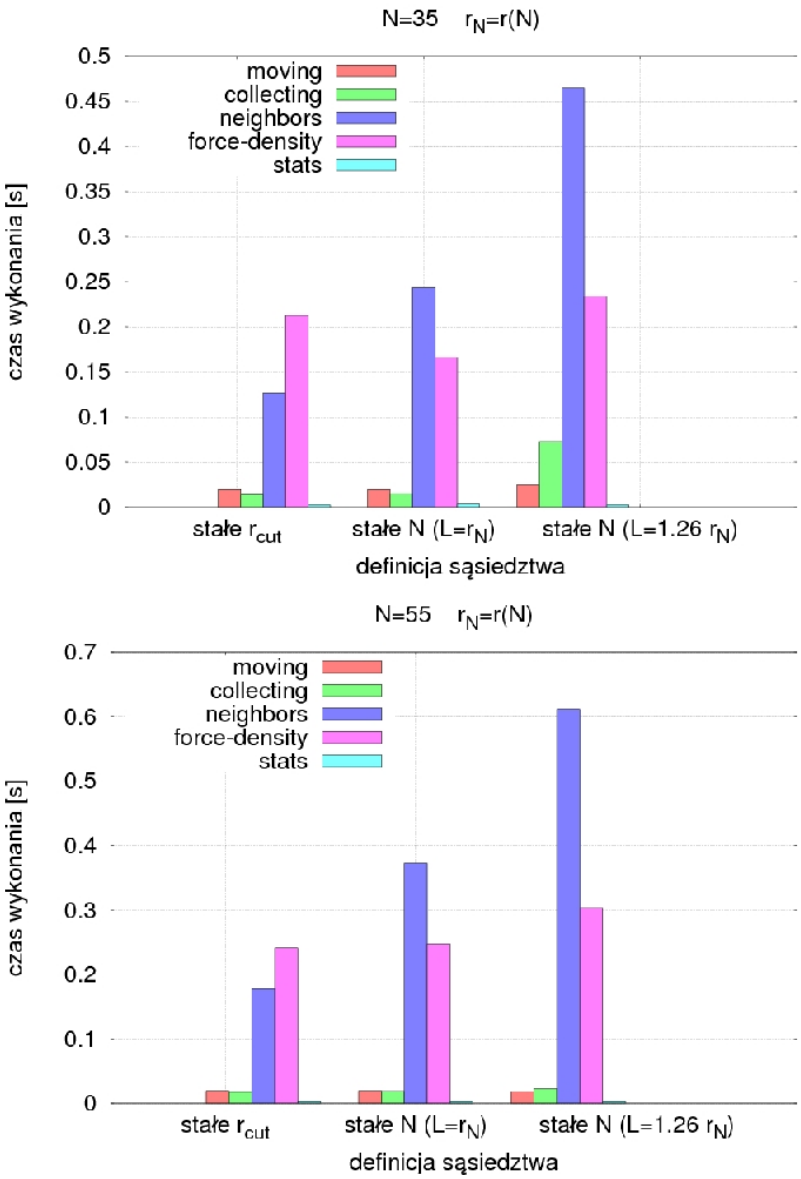
Na każdym z wykresów z rysunku (4.8) przedstawione są trzy grupy wyników: pierwsza dotyczy definicji sąsiedztwa opartej na stałym promieniu sąsiedztwa, druga definicji opartej na stałej liczbie sąsiadów, ale dla przypadku, gdy rozmiar cel był równy promieniowi obcięcia r_{cut}. Natomiast trzecia dotyczy definicji opartej na stałej liczbie sąsiadów, gdy rozmiar celi jest równy $1.26 r_{cut}$. Bezpośrednio można porównać wyniki z grupy pierwszej i trzeciej. Wyniki z grupy drugiej mają charakter poglądowy. Z rysunku (4.8) można wywnioskować, że:

- jedynie dla metody znajdowania sąsiadów, czas wykonania różni się znacząco dla dwóch różnych sposobów definiowania sąsiedztwa,
- sama procedura znajdowania sąsiadów jest znacznie szybsza dla definicji sąsiedztwa opartej na stałym promieniu obcięcia niż dla definicji opartej na stałej liczbie sąsiadów. Przy definicji sąsiedztwa dla stałego promienia obcięcia konieczne jest jedynie obliczenie odległości i porównanie ich z ustaloną wartością, podczas gdy dla definicji opartej na stałej liczbie sąsiadów, konieczne jest posortowanie całej tablicy sąsiadów.

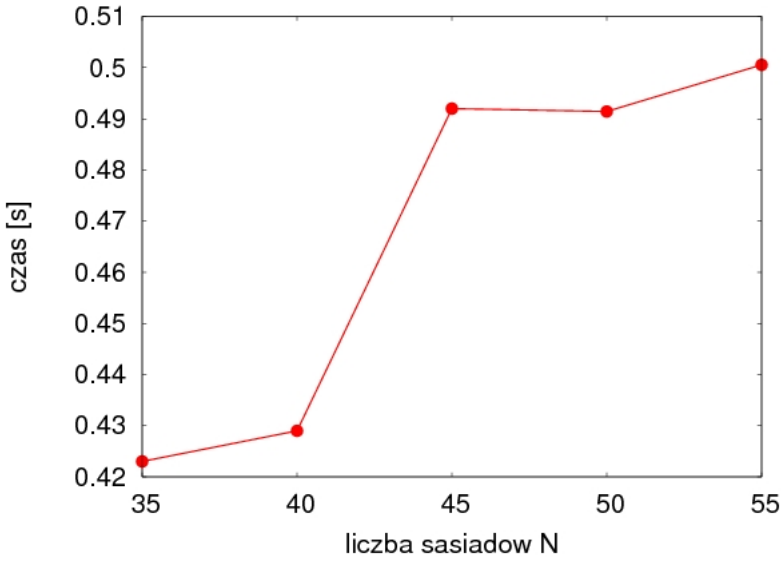
Po zsumowaniu czasów wykonania wszystkich bloków, na jakie została podzielona główna pętla algorytmu, można zauważyć, że algorytm oparty na definicji sąsiedztwa dla stałego promienia obcięcia pozwala skrócić czas wykonania symulacji o czynnik $1.5 - 2.5$ w porównaniu z algorytmem bazującym na definicji opartej na stałej liczbie sąsiadów. Dokładna zależność tej różnicy od liczby sąsiadów w metodzie ze stałą liczbą sąsiadów została wyznaczona i przedstawiona na rysunku (4.9).

4.5.8 Wnioski

Z przedstawionego porównania wynika, że jeśli metoda SPH jest wykorzystywana do modelowania płynów nieściśliwych, wówczas o wiele efektywniej jest stosować definicję sąsiedztwa opartą na stałym promieniu obcięcia, niż na stałej liczbie sąsiadów. Wyniki jakościowe otrzymane przy użyciu obydwu definicji są bardzo do siebie zbliżone, a praktycznie nierozróżnialne, podczas gdy koszty obliczeniowe obydwu definicji jednoznacznie wskazują na tę opartą na stałym promieniu obcięcia jako korzystniejszą. Rozumie się przez to przede



Rysunek 4.8: Porównanie czasów wykonania poszczególnych bloków głównej pętli algorytmu dla $N = 35$ oraz $N = 55$.



Rysunek 4.9: Różnica czasów wykonania jednej pętli algorytmu symulacji dla dwóch definicji sąsiedztwa.

wszystkim krótszy czas obliczeń. Należy jednak wspomnieć również o prostocie implementacji, która cechuje definicję opartą na stałym promieniu obcięcia. Definicja ta jest pozbawiona wielu aspektów komplikujących implementację, z których przykładowo można wymienić niezachowywanie trzeciej zasady dynamiki Newtona przy różnych promieniach obcięcia oddziałujących cząstek, konieczność sortowania listy sąsiadów dla każdej cząstki w każdym kroku obliczeniowym, i wiele innych. Definicja sąsiedztwa oparta na stałej liczbie sąsiadów wciąż jednak jest lepsza w zastosowaniach dotyczących płynów ściśliwych, to jest takich, w których gęstość płynu, a więc i cząstek może przyjmować wartości z dużego przedziału. Wówczas zmienny zasięg oddziaływań jest rzeczą pożądaną, i może być łatwo osiągniety poprzez zastosowanie definicji ze stałą liczbą sąsiadów.

4.6 Podsumowanie

W rozdziale przedstawione zostały podstawy implementacyjne algorytmu sekwencyjnego symulacji. W większej części rozdziału jako przykład wykorzystano implementację metody SPH. Zostały przedstawione sposoby doboru wielkości kroku czasowego oraz zaprezentowano możliwe wybory postaci funkcji ważącej oraz wynikające z tego konsekwencje. Przeglądu tego dokonano nie tylko dla metody SPH, ale dla wszystkich innych metod omawianych w pracy. Kolejno, został opisany problem budowy konfiguracji początkowej symulacji. Następnie szczegółowo przedstawiono przebieg jednego kroku pętli głównej symulacji, wraz z wykorzystywanymi zmiennymi, strukturami oraz podziałem na poszczególne bloki obliczeniowe. W dalszej kolejności zostały przedstawione wyniki porównania dwóch definicji sąsiedztwa dla metody SPH. Wyniki te wskazują, która z przedstawionych definicji jest korzystniejsza w zastosowaniach modelujących zahowanie nieściśliwych i ściśliwych płynów makroskopowych.

Autor do swych głównych osiągnięć zalicza:

1. zaproponowanie w metodzie SPH definicji sąsiedztwa opartej na stałym promieniu obcięcia,

2. opracowanie przesłanek dla efektywnej implementacji algorytmów oddziałujących cząstek dla modeli DPD oraz SPH oraz ich realizację,

3. dokonanie jakościowego porównania wyników symulacji metody SPH wypływu cieczy lepkiej z naczynia oraz przerwania tamy dla zaproponowanej przez autora definicji sąsiedztwa oraz wykorzystywanej w literaturze,

4. porównanie efektywności algorytmów symulacji bazujących na metodzie SPH wykorzystujących omawiane definicje sąsiedztwa.

Rozdział 5

Aspekty implementacji równoległej

Przeprowadzanie symulacji na architekturach wieloprocesorowych umożliwia z jednej strony obserwację układów zbudowanych z coraz większej liczby cząstek, z drugiej, wykonywanie większej liczby kroków symulacji, czyli obserwację zjawisk o długim czasie przebiegu. Implikuje to również możliwość stosowania coraz bardziej złożonych modeli oddziaływań bez konieczności stosowania istotnych uproszczeń implementacyjnych. W rozdziale przedstawiono szczegóły implementacji równoległej oraz analizę wydajności programu symulacji. Pokazano także możliwe podejścia do implementacji równoległej algorytmów metod cząstek. Zaimplementowane algorytmy zostały wykorzystane do symulowania wybranych zjawisk, co zostało przedstawione w kolejnym rozdziale.

Symulacje numeryczne metodami cząstek złożonych systemów wymagają wykorzystania systemów komputerowych o dużej mocy obliczeniowej. Duże wymagania pamięciowe tego typu problemów wymuszają dostępność względnie dużych zasobów. Dlatego też powszechne w tych zagadnieniach jest wykorzystanie architektur wieloprocesorowych z pamięcią rozproszoną lub współdzieloną. Pierwsze z wymienionych są rozwiązaniami względnie tanimi, jednak paradygmat oparty na przesyłaniu komunikatów, który jest wymuszany przez te architektury, wymaga dużego doświadczenia programistycznego i dużych nakładów czasowych na implementacje. Przeważnie architektury te są realizowane na klastrach obliczeniowych, które charakteryzuje wiele instancji systemu operacyjnego i stosunkowo duże opóźnienia komunikacyjne. Architektury z pamięcią współdzieloną są relatywnie droższe, jednak środowiska programistyczne służące do równoległej implementacji problemów są względnie łatwe i pozwalają na szybką implementację. Architektury te, określane jako SMP (ang. Symmetric Multiprocessing) charakteryzują się jedną instancją systemu operacyjnego i jedną przestrzenią pamięci liniowo adresowalną. Metody cząstek zostały zaimplementowane przy pomocy środowisk: OpenMP oraz MPI, które, odpowiednio, realizują obydwa paradygmaty programowania równoległego.

5.1 Wskaźniki obliczeń równoległych

W celu określenia wydajności implementacji równoległej algorytmu stosuje się kilka wskaźników obliczeń równoległych. Stanowią one miary pozwalające określić stopień wykorzystania zasobów obliczeniowych oraz ocenę, dzięki której możliwe jest porównanie dwóch różnych implementacji równoległych tego samego problemu. Najczęściej używanymi wskaźnikami są [62]:

- S_N - przyspieszenie (ang. speedup). Jest to wielkość określająca o ile szybszy jest algorytm uruchomiony równolegle od algorytmu uruchomionego sekwencyjnie. Wartość przyspieszenia oblicza się z wzoru:

$$S_N = \frac{t_1}{t_N}, \tag{5.1}$$

gdzie N oznacza liczbę procesorów, t_1 czas wykonania algorytmu sekwencyjnego, a t_N czas wykonania algorytmu równoległego na N procesorach. Czas t_1 można uzyskać na dwa sposoby: albo poprzez wykonanie algorytmu sekwencyjnego, albo poprzez uruchomienie algorytmu równoległego na jednym procesorze. W zależności od tego przyspieszenie nazywa się odpowiednio przyspieszeniem bezwzględnym albo przyspieszeniem względnym.

- E_N - efektywność. Wielkość tę można interpretować jako stopień wykorzystania dostępnych zasobów obliczeniowych. Wartość efektywności otrzymuje się z wzoru:

$$E_N = \frac{S_N}{N} = \frac{t_1}{N t_N}. \tag{5.2}$$

W zależności od tego, czy we wzorze (5.2) wykorzystano przyspieszenie względne czy bezwzględne, definiuje on efektywność odpowiednio względną lub bezwzględną. Pozwala ona ocenić w jakim stopniu wykorzystane są dostępne jednostki obliczeniowe w porównaniu do czasu, który został wykorzystany na komunikację i synchronizację pomiędzy nimi. Efektywność E_N przyjmuje wartości z przedziału $(0, 1)$, o ile nie mamy do czynienia z przypadkiem przyspieszenia superliniowego.

W każdym programie można wyróżnić dwie części: część sekwencyjną, niemożliwą do zrealizowania w sposób równoległy oraz część, która może zostać wykonana równolegle na wielu jednostkach obliczeniowych. Można przyjąć, że jeśli w przypadku uruchomienia programu na jednym procesorze czas wykonania programu wynosi t_1, to wówczas czas wykonania części sekwencyjnej jest równy $t_{1s} = st_1$, a części równoległej $t_{1p} = pt_1$, gdzie $t_{1s} + t_{1p} = t_1 \Leftrightarrow s + p = 1$. W przypadku algorytmu sekwencyjnego, którego żadna część nie może być zrównoleglona, zachodzi $p = 0$, natomiast w przypadku algorytmu idealnie równoległego $p = 1$. Prawo Amdahla [4] mówi, że jeśli dowolny program zostanie uruchomiony na N procesorach, wówczas maksymalne przyspieszenie będzie równe:

$$S_N = \frac{1}{s + p/N}, \tag{5.3}$$

co jest równoważne poniższemu równaniu na czas wykonania symulacji:

$$t_N = t_s + \frac{t_p}{N}. \tag{5.4}$$

Prawo to określa ograniczenie na przyspieszenie jak również na czas wykonania dla algorytmu równoległego. Według tego prawa czas wykonania algorytmu równoległego t_N nigdy nie będzie krótszy od t_s, niezależnie od liczby wykorzystanych jednostek obliczeniowych. Czas t_s jest natomiast granicą dla czasu wykonania algorytmu, jeśli liczba jednostek $N \to \infty$. W praktyce przyspieszenie wykonania równoległego jest ograniczone również poprzez narzut komunikacyjny oraz obecność pamięci podręcznych dla każdego procesora, co dodatkowo zmniejsza wartość przyspieszenia. Prawo to przez wiele lat było uważane za ograniczenie efektywności implementacji równoległych oraz wpływało na kierowanie badań nad zwiększeniem efektywności na inne dziedziny.
Inną powszechnie stosowaną zależnością w obliczeniach równoległych jest prawo Gustafsona [69]:

$$S_N = N - s'_N(N - 1), \tag{5.5}$$

gdzie $s'_N = t_{Ns}/t_N$ jest stosunkiem czasu wykonania części sekwencyjnej do czasu wykonania całego programu na N procesorach. Zgodnie z tą definicją $s = s'_1$. Obydwa prawa określają zależność przyspieszenia od stosunku czasu wykonania części sekwencyjnej t_{Ns} do czasu wykonania całego programu t_N. O ile jednak prawo Amdahla (5.3) jest zależnością typu x^{-1}, to prawo Gustafsona jest zależnością liniową. Ta pozorna różnica wynika z różnych definicji wielkości s oraz s'_N dla $N > 1$ i znika, gdy skorzysta się z tożsamości:

$$s'_N = N\left(N - 1 + s^{-1}\right)^{-1}. \tag{5.6}$$

Tożsamość tę łatwo wykazać korzystając z definicji wielkości s, s'_N oraz pamiętając, że $t_{1s} = t_{Ns}$ oraz $t_{1p} = Nt_{Np}$. Po jej zastosowaniu okazuje się, że obydwa prawa są sobie równoważne. Wynika stąd, że prawo Gustafsona jest prawem Amdahla wyrażonym w przedefiniowanych zmiennych. Widać więc, że kluczowymi dla otrzymania każdego z tych praw są definicja oraz wzajemny stosunek wielkości s oraz p.

Osobną kwestią, którą poruszył Gustafson w swojej pracy [69] jest zmiana podziału czasu wykonania programu na s oraz p w zależności od rozmiaru problemu n. Zauważył on bowiem, że wielkość s (jak również p) jest zależna od rozmiaru problemu n: $s \equiv s(n)$ i najczęściej wraz ze wzrostem rozmiaru problemu n coraz większa jego część przypada na część równoległą p. Relacja ta jest zależna od konkretnego problemu i nie można podać dla niej żadnego ogólnego wzoru, a ujawnia się ona zwykle podczas zwiększania liczby dostępnych jednostek obliczeniowych. W przypadku takim, zamiast uruchamiać zadanie o tym samym rozmiarze problemu na większej liczbie procesorów, zwiększa się go proporcjonalnie do wzrostu liczby procesorów, aby w tym samym czasie przetworzyć większą ilość danych. Sytacja taka występuje powszechnie w zastosowaniach praktycznych, niezwiązanych z przypadkami akademickimi. Jednak w jej przypadku niemożliwe jest skorzystanie z przyspieszenia zdefiniowanego wzorem (5.1), gdyż dotyczy ona różnych rozmiarów tego samego problemu, podczas gdy definicja ta dotyczy stałego rozmiaru problemu. Dlatego też wygodnie jest zdefiniować jeszcze jeden wskaźnik obliczeń równoległych:

- przyspieszenie przeskalowane, które dane jest wzorem:

$$S_{\text{scaled}} = \frac{Nt_{1,1}}{t_{N,n}}, \tag{5.7}$$

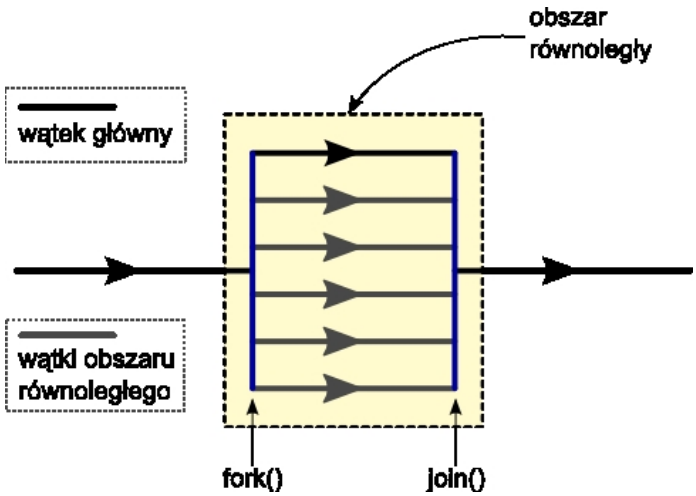
  gdzie N oznacza liczbę procesorów, a n rozmiar problemu. Zwykle obie te wielkości są do siebie proporcjonalne. Czas $t_{1,1}$ odnosi się do czasu wykonania problemu o umownym rozmiarze 1 na jednym procesorze, natomiast czas $t_{N,n}$ do czasu wykonania tego samego problemu o rozmiarze n na N procesorach.

Wartość przyspieszenia przeskalowanego S_{scaled} jest najczęściej o wiele większa niż przyspieszenia "zwykłego". Stąd też powszechnie panująca opinia, że wyniki zaprezentowane w pracy Gustafsona złamały prawo Amdahla [126]. W rzeczywistości jednak ujmują one kwestię obliczeń równoległych z innej strony i poprzez definicję nowych wielkości oraz uwzględnienie nowych aspektów wskazują na korzyści płynące z implementacji równoległej, którą w dotychczasowym ujęciu ograniczało prawo Amdahla.

Wskaźnikiem wykonania równoległego stosowanym do oceny algorytmów równoległych prezentowanych w niniejszej pracy jest efektywność względna.

5.2 Implementacja dla architektur z pamięcią współdzieloną

Jeden z paradygmatów programowania dla komputerów z pamięcią współdzieloną polega na utworzeniu i wykorzystaniu dodatkowych wątków w ramach procesu głównego oraz po-



Rysunek 5.1: Algorytm OpenMP: utworzenie wątków w procesie.

dziale zaplanowanych obliczeń na części wykonywane przez poszczególne wątki. Korzystają one ze wspólnej przestrzeni adresowej oraz zmiennych dostępnych jednakowo dla każdego wątku. Natomiast każdy wątek jest wykonywany na innej jednostce obliczeniowej. Środowisko uruchomieniowe zarządza wykonaniem wątków oraz przydziela je do dostępnych jednostek obliczeniowych (procesorów).

5.2.1 Środowisko OpenMP

W chwili obecnej jednym z najpopularniejszych narzędzi służących do tworzenia implementacji równoległych na maszynach z pamięcią współdzieloną jest środowisko OpenMP. Stanowi ono standard określający zestaw narzędzi dostępnych z poziomu programisty, pozwalających na równoległą implementację rozwiązywanego problemu. Umożliwia ono uruchomienie i rozwiązanie problemu poprzez utworzenie wątków w ramach procesu, z których każdy bierze udział w rozwiązywaniu zadania, rys. (5.1). Środowisko OpenMP ukrywa przed programistą większość szczegółów związanych z tworzeniem, synchronizacją oraz usuwaniem wątków. Nie musi on nawet dokładnie określać liczby wątków przy pomocy których ma zostać rozwiązane dane zagadnienie. Zadaniem programisty natomiast jest określenie, które części programu powinny zostać wykonane w sposób równoległy, wynikająca z tego reorganizacja kodu źródłowego, określenie dostępności zmiennych w poszczególnych wątkach i inne kwestie związane z synchronizacją danych pomiędzy wątkami. Kod źródłowy implementacji OpenMP jest przenośny i niezależny od konkretnej architektury.
Standard OpenMP został opracowany przez konsorcjum ARB (The OpemMP Architecture Review Board) powołane specjalnie w tym celu przez głównych producentów sprzętu i oprogramowania. Z głównych członków ARB należy wymienić takie firmy jak AMD, Cray, Intel, IBM, SGI, Sun czy Microsoft. Kompletną ich listę można znaleźć na stronie głównej konsorcjum [109]. Wsparcie od tak wielu jednostek i organizacji zapewniło szybką akceptację środowiska jako standardu. Pierwsza specyfikacja OpenMP ujrzała światło dzienne w 1997 roku. Od tamtej pory środowisko jest ciągle rozwijane i wzbogacane o nowe funkcjonalności. Ostatnia wersja standardu, oznaczona numerem 3.0 została opublikowana w maju 2008 roku, a wprowadzone w niej nowe możliwości zostały opisane w jej specyfikacji [108].

Standard OpenMP obejmuje zestaw dyrektyw, funkcji bibliotecznych oraz zmiennych środowiskowych. Dyrektywy służą jako instrukcje dla kompilatora wskazujące, które bloki programu i pętle powinny być wykonane w sposób równoległy poprzez rozdzielenie wykonania procesu na wątki. Za pomocą dyrektyw programista określa także, w jaki sposób

uruchamiane wątki mają współdzielić zmienne, w jakiej kolejności mają wykonywać pętle oraz wiele innych szczegółowych kwestii związanych z implementacją równoległą. Większość pętli i bloków występujących w kodzie źródłowym programu może zostać prawie natychmiast zrównoleglona poprzez wstawienie przed nimi pojedynczej dyrektywy. Przykładowo w języku C++ pętla `for` może być zrównoleglona w następujący sposób:

Fragment kodu źródłowego 5.1: Metoda zrównoleglenia pętli `for` przy pomocy dyrektywy OpenMP.

```
#pragma omp parallel for schedule(dynamic,1)
for ( i1 = 0 ; i1 < cells.size() ; i1++ )
{
  ...
}
```

Jest to jedynie przykład, jeden z wielu możliwych. Opcja `shedule(dynamic,1)` określa dokładny sposób zrównoleglenia pętli. Oznacza ona, że iteracje pętli zostaną podzielone na grupy zawierające po jednej iteracji każda, i będą przydzielane do poszczególnych wątków dynamicznie, to znaczy na bieżąco, w miarę jak będą one kończyły poprzednio przydzielone iteracje.

Funkcje biblioteczne dostępne w standardzie OpenMP służą w większości do odczytywania informacji o bieżącym stanie wykonania równoległego, takich jak liczba uruchomionych wątków, numer bieżącego wątku i kilka innych parametrów dotyczących wykonania równoległego. Możliwe jest także ustawienie z ich wykorzystaniem liczby watków w kolejnym bloku równoległym czy też ustawienie blokady na wybranym obiekcie. Wątek, ustawiając taką blokadę, zapewnia sobie wyłączność na dostęp do takiego obiektu.

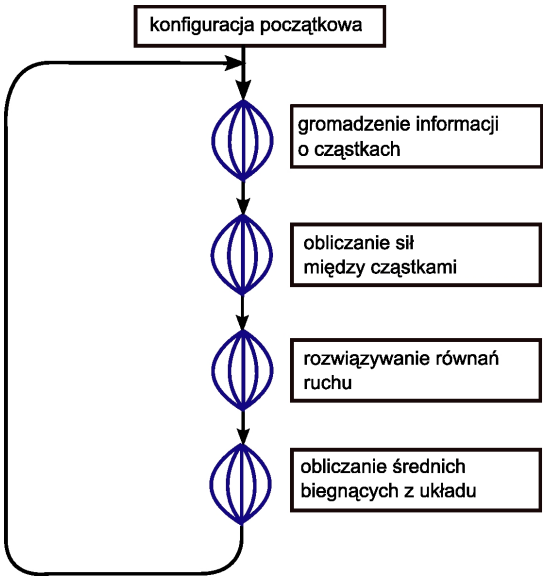
Standard OpenMP jest przeznaczony dla języków C/C++ oraz Fortran, a szczegóły składniowe dyrektyw i funkcji są dostępne w specyfikacjach standardu. Konkretne implementacje standardu są tworzone bądź przez producentów sprzętu, którzy dostarczają na produkowane przez siebie architektury kompilatory obsługujące standard OpenMP, bądź też przez oddzielne organizacje zajmujące się tworzeniem kompilatorów. W pracy zostały wykorzystane implementacje OpenMP dołączone do kompilatorów dostarczanych przez firmy Intel [79] oraz PGI [114].

Bardzo istotną cechą środowiska OpenMP jest możliwość stopniowego zrównoleglania kodu źródłowego programu. Może on być zrównoleglany krok po kroku, gdzie za każdym razem kolejny blok programu jest przeznaczany do wykonania równoległego. W każdym takim etapie kod źródłowy programu może być kompilowany i wykonywany, a programista może go testować i debugować. Nie jest to możliwe w niektórych innych podejściach do programowania równoległego, w których projekt wykonania równoległego musi zostać utworzony przed przystąpieniem do implementacji, i który nie może już zostać zmieniony w późniejszym etapie.

Środowisko OpenMP znalazło szerokie zastosowanie zarówno w implementacjach naukowych jak i praktycznych. Wiele z nich jest przedstawianych na corocznej konferencji IWOMP [44]. Szerokie omówienie programowania z wykorzystaniem OpenMP można znaleźć w pracy [30].

5.2.2 Realizacja przy pomocy środowiska OpenMP

Jak wspomniano, realizacja algorytmu symulacji metodami cząstek polega na wygenerowaniu konfiguracji początkowej układu (ustalenie współrzędnych położeń i prędkości cząstek) oraz wykonaniu określonej liczby iteracji głównej pętli symulacji. Pętla ta została podzielona na kilka bloków, z których część mogła być wykonana w sposób równoległy, rys. (5.2), a pozostała część sekwencyjnie. Przed bloki, które mogły być wykonane równolegle wsta-



Rysunek 5.2: Schemat algorytm metody cząstek zrównoleglonego przy pomocy środowiska OpenMP.

wiono odpowiednie dyrektywy OpenMP. Bloki te (wraz z liniami, w których się znajdują na listingu 5.2) to:

1. gromadzenie informacji o cząstkach z cel sąsiednich (linie: 6-19),
2. obliczanie sił pomiędzy oddziałującymi cząstkami zgodnie z założonym modelem oddziaływania (linie: 21-41),
3. rozwiązanie układu równań ruchu, czyli wyznaczenie nowych położeń i prędkości cząstek, przypisywanie cząstek do cel (linie: 43-66),
4. wyznaczeniu wartości bieżących parametrów układu (linie: 68-87).

Poniżej zamieszczono kod źródłowy pętli głównej, w której występują wymienione wyżej bloki wraz z dyrektywami OpenMP, dzięki którym bloki te zostały zrównoleglone.

Fragment kodu źródłowego 5.2: Zrównoleglenie pętli głównej poprzez wstawienie dyrektyw OpenMP do kodu źródłowego.

```
#pragma omp parallel private(i1,i2,i3,i4,tmpCell,threadNum,index) \
        shared(EKin,Entropy,Temperature)
for ( ; stepCounter < totalSteps ; ) {
  threadNum = omp_get_thread_num();

  //----------------------------------------------------------------
  // GATHERING
  #pragma omp for schedule(dynamic,1)
  for ( i1=0 ; i1 < cells.size() ; i1++ ) {
    cells[i1]->prtcls26C.clear();

    for ( i2=0 ; i2 < 27 ; i2++ ) {
      tmpCell = cells[i1]->neighsC[i2];
      if ( ( tmpCell == NULL ) || ( tmpCell == cells[i1] ) ) continue;
      /* dodatkowe obliczenia, aby pary czastek sie nie powtarzaly */
      cells[i1]->prtcls26C.add(tmpCell->prtclsC.ptr(),
                               tmpCell->prtclsC.size());
    }
  }

```

```
  //------------------------------------------------------------------
  // FORCES
  #pragma omp for schedule(dynamic,1)
  for ( i1 = 0 ; i1 < cells.size() ; i1++ ) {
    for ( i2=0 ; i2 < cells[i1]->prtclsC.size() ; i2++ ) {
      for ( i3=0 ; i3 < cells[i1]->prtclsC.size() ; i3++ )
        if ( cells[i1]->prtclsC[i2] < cells[i1]->prtclsC[i3] )
          force( ompPrtcls[threadNum],
            cells[i1]->prtclsC[i2], cells[i1]->prtclsC[i3] );
      for ( i3=0 ; i3 < cells[i1]->prtcls26C.size() ; i3++ )
        force( ompPrtcls[threadNum],
            cells[i1]->prtclsC[i2], cells[i1]->prtcls26C[i3] );
    }
  }

  for (i1=0 ; i1 < prtcls.size() ; i1++ ) {
    omp_set_lock( &(prtcls[i1]->pLock) );
    for ( i2=0 ; i2 < 3 ; i2++ )
      prtcls[i1]->_acc[i2] += ompPrtcls[threadNum][i1].accs[i2];
    omp_unset_lock( &(prtcls[i1]->pLock) );
  }

  //------------------------------------------------------------------
  // MOTION
  #pragma omp for schedule(dynamic,1) nowait
  for ( i1=0 ; i1 < cells.size() ; i1++ ) {
    cells[i1]->prtclsC.clear(); cells[i1]->prtcls26C.clear();
  }
  #pragma omp for schedule(dynamic,1)
  for ( i1 = 0 ; i1 < prtcls.size() ; i1++ ) {
    /* wyznaczanie nowych wartości polozenia, przyspieszenia itp dla
          czastki o numerze i1 */

    index <- numer celi w ktorej znajduje sie czastka i1 0;
    prtcls[i1]->iamfrom( cells[index] );
    ompCells[threadNum][index].push_back( prtcls[i1] );
  }   // tutaj jest domyślna bariera

  for ( i2=0 ; i2 < cells.size() ; i2++ ) {
    omp_set_lock( &(cells[i2]->cLock) );
    cells[i2]->prtclsC.add( ompCells[threadNum][i2].ptr(),
                  ompCells[threadNum][i2].size() );
    omp_unset_lock( &(cells[i2]->cLock) );

    ompCells[threadNum][i2].clear();
  }

  //------------------------------------------------------------------
  // STATISTICS
  #pragma omp for schedule(dynamic,1) reduction(+:EKin,Entropy,Temperature)
  for ( i1 = 0 ; i1 < prtcls.size() ; i1++ ) {
    /*
    energia kinetyczna, entropia i temperatura kazdej czastki
    sa dodawane do zmiennych globalnych dla wszystkich czastek: EKin,
    Entropy, Temperature
    */
  }

  #pragma omp single
  {
    Temperature /= prtcls.size();
    cout << "Ekin=" << EKin << "\nEntropy=" << Entropy
      << "\nTemperature=" << Temperature << endl;
    EKin = 0.0; Entropy = 0.0; Temperature = 0.0;
  }

  //------------------------------------------------------------------
  #pragma omp single
  {
    cout << "\nSTEP:␣" << stepCounter << "\n";
    stepCounter++;
  }
```

```
} // main loop
```

Pierwsza instrukcja środowiska OpenMP, która znajduje się w linii 1, to dyrektywa `#pragma omp parallel`. Jest to podstawowa dyrektywa środowiska. W momencie jej napotkania w trakcie wykonywania programu tworzona jest grupa wątków, które wykonują następujący po dyrektywie blok programu w sposób równoległy. Wątki są niszczone gdy sterowanie napotka koniec tego bloku programu. Do dyrektywy `parallel` dołączone są klauzule współdzielenia zmiennych. Zmienne występujące w klauzuli `shared` są współdzielone przez wszystkie wątki, natomiast dla zmiennych występujących w klauzuli `private` tworzone są ich kopie dla każdego wątku. Zmiany wartości takich zmiennych dokonywane w jednym wątku nie będą wpływały na wartości tych zmiennych w pozostałych wątkach. W następnej kolejności, w linii 4 występuje wywołanie funkcji bibliotecznej w celu określenia numeru bieżącego wątku i zapisania go w zmiennej `threadNum`. Dodatkowe zmienne związane z implementacją OpenMP, które musiały zostać wprowadzone, zostały przedstawione w tabeli (5.1).

nazwa zmiennej	**typ zmiennej**	**opis zmiennej**
`threadNum`	`int`	numer bieżącego wątku
`ompPrtcls[threadNum]`	`**`	tymczasowa tablica przechowująca obliczane siły dla cząstek
`ompCells[threadNum]`	`**`	tymczasowa tablica przechowująca indeksy cząstek dla cel
`Cell.cLock`	`omp_lock`	zmienna wykorzystywana przy ustawianiu blokady na celę
`Particle.pLock`	`omp_lock`	zmienna wykorzystywana przy ustawianiu blokady na cząstkę

Tabela 5.1: Zmienne wprowadzone w implementacji równoległej OpenMP.

Pierwszym blokiem programu rezlizowanym w sposób równoległy jest gromadzenie informacji o cząstkach z cel sąsiednich. W porównaniu z wersją sekwencyjną jedyną zmianą w wersji równoległej jest zrównoleglenie pętli z linii 9-19 w sposób podobny jak dla pętli z bloków obliczających siły i nowe położenia.

Kolejnym blokiem programu, który został zrealizowany w sposób równoległy jest część obliczająca siły pomiędzy cząstkami. W części tej podstawową zmianą w porównaniu do implementacji sekwencyjnej (fragment kodu źródłowego 4.1) jest dodanie kolejnego argumentu do funkcji `force()` obliczającej siły pomiędzy dwoma cząstkami, którym jest wskaźnik do tablicy `ompPrtcls[threadNum]`. Tablica ta ma rozmiar równy liczbie cząstek w układzie, `prtcls.size()`, a jej elementami są trójelementowe tablice typu `double`, które przechowują wartości przyspieszeń dla każdej cząstki (w trzech wymiarach). W funkcji `force(tab,p1,p2)` obliczane są siły występujące pomiędzy cząstkami o numerach `p1` oraz `p2`. W przypadku implementacji sekwencyjnej oddziaływania te mogły być od razu dodawane do innych oddziaływań działających na te cząstki. W przypadku implementacji równoległej na komputerze z pamięcią współdzieloną nie jest to już możliwe. Mogłoby się bowiem okazać, że dwa wątki próbują zmodyfikować wartość siły działającej na wybraną cząstkę równocześnie. Wówczas wartość siły po takim zapisie jest niemożliwa do zdeterminowania i niezwiązana z rzeczywistymi obliczeniami, a przez to wyniki symulacji mogą być niepoprawne. Aby uniknąć takiej sytuacji wprowadzono tablicę `ompPrtcls[threadNum]` dla każdego wątku i skorzystano

z faktu, że cząstka `p1`, w przeciwieństwie do cząstki `p2`, zawsze znajduje się w tej celi, która została przydzielona do bieżącego wątku przy wykonywaniu pętli z linii 24-34. Każdy wątek w funkcji `force()` zapisuje obliczoną wartość siły dla cząstki `p1` w globalnej tablicy cząstek `prtcls`, natomiast dla cząstki `p2` w dostępnej tylko dla siebie tablicy `ompPrtcls[threadNum]`. W ten sposób nie jest możliwe wystąpienie kolizji podczas zapisu. Ostatecznie, po obliczeniu wszystkich oddziaływań pomiędzy cząstkami, każdy wątek wykonuje pętlę z linii 36-41, w której zgromadzone wartości w tablicy `ompPrtcls[threadNum]` wpisuje do tablicy globalnej. Każdy wątek zapewnia sobie wyłączność na zapis wartości dla wybranej cząstki do tablicy globalnej `prtcls` poprzez wywołanie funkcji bibliotecznych OpenMP zakładających blokadę na zmienne charakteryzujące tę cząstkę: `omp_set_lock()` oraz `omp_unset_lock()` w liniach 37 oraz 40. W ten sposób siły pomiędzy cząstkami obliczone przez wszystkie wątki zostaną zapisane do odpowiednich komórek tablicy globalnej i będzie możliwe przejście z obliczeniami do następnego etapu.

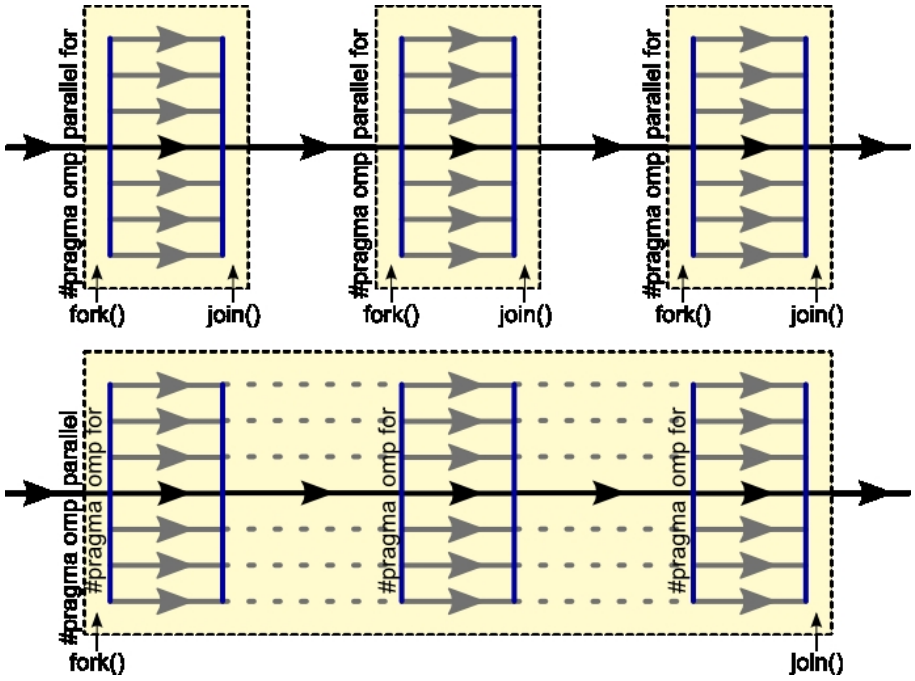
Kolejnym fragmentem głównej pętli algorytmu jest obliczanie nowych wartości położeń i prędkości cząstek na podstawie wcześniej obliczonych sił. W porównaniu z wersją sekwencyjną, w implementacji równoległej zrównoleglono pętle z linii 46-47 oraz 50-57. Podobnie, jak przy obliczaniu sił, konieczne okazało się wprowadzenie dodatkowej tablicy na dane tymczasowe. Każdy wątek posiada tablicę `ompCells[threadNum]` o rozmiarze równym liczbie cel, której elementami są tablice przechowujące wskaźniki do czastek znajdujących się w danej celi. W wersji sekwencyjnej możliwe było dodawanie wskaźników cząstek do tablic odpowiedniej celi w strukturze globalnej `cells`. W wersji równoległej z pamięcią współdzieloną nie jest to jednak możliwe, gdyż w przypadku takim mogłoby się zdarzyć, że dwa wątki próbują jednocześnie dokonać zapisu do tablicy przypisanej tej samej celi. Wówczas wartości zmiennych będą niedeterministyczne, co prowadzi do zaburzenia wyników symulacji. Dlatego też wprowadzono dodatkową, tymczasową dla każdego wątku tablicę `ompCells[threadNum]`, z której dane są następnie uwzględniane w tablicy globalnej, w pętli z linii 59-66. W pętli tej, podobnie jak przy obliczaniu sił wykorzystuje się mechanizm blokad udostępniany przez środowisko OpenMP (linie 60 oraz 63).

Ostatnim blokiem pętli głównej jest obliczanie średnich bieżących i statystyk z układu. Główną różnicą w porównaniu z wersją sekwencyjną jest zrównoleglenie pętli z linii 71-77 oraz wykonanie w niej redukcji operatorem `'+'` na zmiennych skalarnych wymienionych w klauzuli `reduction` . Oznacza to, że zmienne te po zakończeniu pętli z linii 71-77, wykonywanej równolegle, będą miały wartości równe sumie ich prywatnych kopii ze wszystkich wątków, które wykonywały tę pętlę. Występująca w linii 79 dyrektywa `single` sprawia, że następujący po niej blok programu z linii 80-85 zostanie wykonany tylko przez jeden wątek. To samo dotyczy inkrementacji wskaźnika zliczającego iteracje pętli głównej z linii 91.

Nasuwa się pytanie, dlaczego nie zrównoleglono każdej z pętli `for` występującej w pętli głównej w sposób pokazany na listingu 5.1, czyli poprzez jednoczesne użycie dyrektyw `parallel` oraz `for` za każdym razem. Powodem tego jest to, że z każdym wystąpieniem dyrektywy `parallel` związane jest utworzenie i zniszczenie grupy wątków, co zajmuje dodatkowy czas procesora oraz zwiększa udział części s w całkowitym czasie wykonania programu [108]. O wiele efektywniej jest wywołać dyrektywę `parallel` tylko raz, a następnie każdą pętlę zrównoleglać osobno. W części programu wykonywanej sekwencyjnie utworzone wątki przechodzą w stan spoczynku i oczekują na kolejne wywołanie dyrektywy `for`. Obydwa sposoby zostały poglądowo przedstawione na rysunku (5.3).

5.2.3 Efektywność implementacji

W celu określenia efektywności implementacji równoległej w środowisku OpenMP wykorzystano symulację przy pomocy metody SPH zjawiska przerwania tamy opisaną w [137]. Uruchomiono cztery wersje tej symulacji. Wersje te różniły się liczbą cząstek znajdujących



Rysunek 5.3: Wpływ wykorzystanie dyrektyw `parallel` oraz `for` razem lub osobno w implementacji równoległej OpenMP na tworzenie oraz usuwanie wątków.

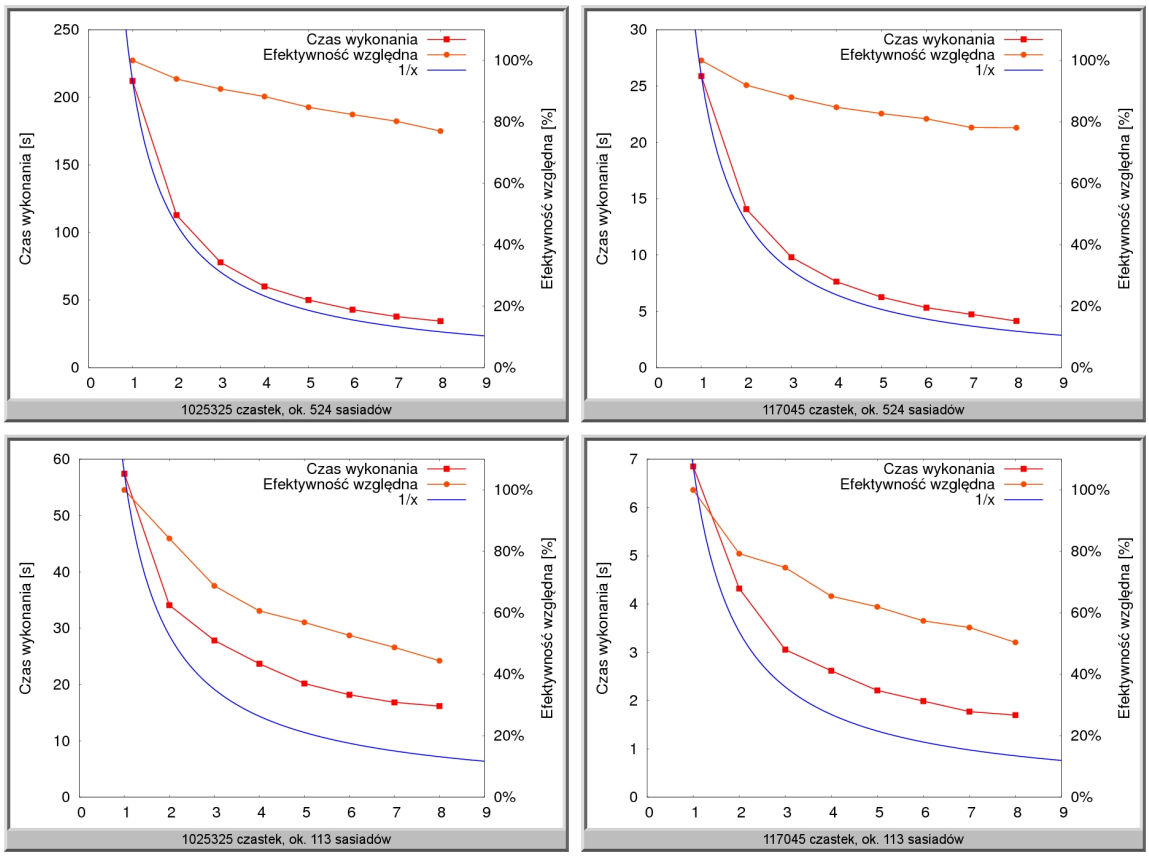
się w układzie (odpowiednio 117045 oraz 1025325) oraz średnią liczbą sąsiadów dla cząstek (odpowiednio ok. 113 oraz ok. 524). Każda z wymienionych wersji została uruchomiona na różnej liczbie procesorów. Dzięki temu możliwe było wyznaczenie efektywności względnej implementacji równoległej tych symulacji. Wyniki pomiarów średniego czasu wykonania jednego kroku symulacji przedstawiono na rysunku (5.4).

Na rysunku tym widoczne są cztery wykresy. Każdy przedstawia czas wykonania dla symulacji o innych parametrach. Na każdym z nich przedstawiony został: rzeczywisty średni czas wykonania jednej pętli symulacji, przypadek idealny ($s = 0$, $p = 1$) oznaczony jako $1/x$ oraz efektywność względna symulacji. Z tych czterech wykresów można wywnioskować, że efektywność względna zależy od średniej liczby sąsiadów w symulacji. Na efektywność względną nie ma natomiast wpływu wielkość układu. Wpływa ona natomiast na czas wykonania pojedynczej pętli symulacji (a więc również całej symulacji).

Wyznaczanie wartości sił oddziaływania między cząstkami jest kosztowne obliczeniowo. Jednak obliczenia te są wykonywane w bloku programu, który został zrealizowany równolegle (część p). Kiedy średnia liczba sąsiadów jest duża, to stosunek czasu obliczeniowego zużytego na obliczenia realizowane równolegle do czasu obliczeniowego zużytego na obliczenia zrealizowane sekwencyjnie t_p/t_s jest większy, niż w przypadku, gdy średnia liczba sąsiadów cząstek jest mniejsza. Widać więc, że wyniki zaprezentowane na rysunku (5.4) są jakościowo zgodne z oczekiwaniami.

5.3 Wersja dla komputerów z pamięcią rozproszoną

Wieloprocesorowe systemy komputerowe oparte na architekturze z pamięcią rozproszoną ograniczają możliwy sposób zrównoleglania programu do paradygmatu opartego na przesyłaniu komunikatów. W przypadku takim implementowany algorytm musi zostać poddany dekompozycji na mniejsze części, z których każda wykonywana jest na oddzielnej jednostce obliczeniowej. Procesy uruchomione na poszczególnych jednostkach komunikują się między



Rysunek 5.4: Wyniki pomiarów średnich czasów wykonania jednego kroku symulacji dla różnej liczby procesorów oraz różnych parametrów symulacji. Implementacja z wykorzystaniem środowiska OpenMP.

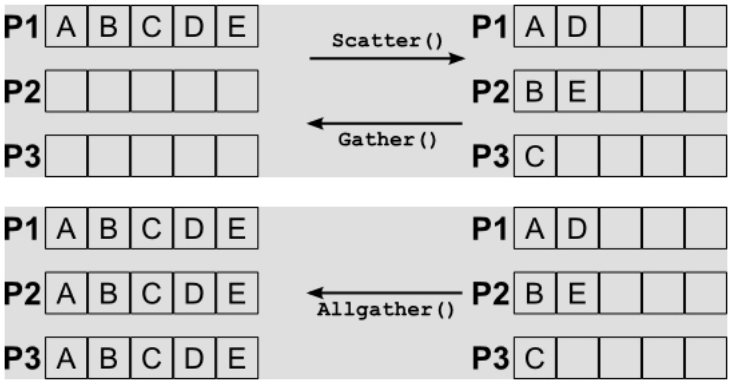
sobą poprzez przesyłanie komunikatów, które mogą zawierać zarówno sygnały synchronizujące jak i dane. Programista ma dostęp do funkcji przesyłających komunikaty i to do niego należy zaprojektowanie komunikacji, synchronizacja oraz organizacja przesyłanych danych.

Jednym z najbardziej popularnych środowisk wspomagających implementację równoległą opartą o przesyłanie komunikatów jest środowisko MPI (ang. Message Passing Interface) [94]. Jest to niezależny od wykorzystywanego języka programowania czy architektury protokół komunikacji pomiędzy jednostkami obliczeniowymi. Dopuszcza on komunikację pomiędzy dwoma dowolnymi procesami, jak również komunikację zbiorową pomiędzy nimi w ramach zdefiniowanego uprzednio zbioru. MPI zapewnia dużą wydajność, skalowalność i przenośność aplikacji napisanych z jego wykorzystaniem i jest powszechnie wykorzystywanym środowiskiem w obliczeniach dużej mocy [5]. MPI realizuje model przetwarzania równoległego SPMD (ang. Single Program Multiple Data), który jest częścią modelu MIMD (ang. Multiple Instruction Multiple Data) ujętego w taksonomii Flynna [52]. W modelu tym ten sam kod maszynowy wykonywany jest na wielu jednostkach obliczeniowych w ramach oddzielnych procesów, i każdy z nich przetwarza inne dane. Ważną cechą środowiska MPI jest otwartość jego specyfikacji, czyli zarówno składni poszczególnych funkcji udostępnianych przez MPI, jak również ich semantykę. Dzięki temu dostępnych jest wiele implementacji tego standardu.

Specyfikacja MPI określa interfejsy funkcji dla jezyków Fortran, C i od wersji 2.0 dla C++. Dostępnych jest również wiele innych interfejsów, nie ujętych w ramach specyfikacji, do języków takich jak Java, Python, OCaml i wielu innych. Dzięki tym cechom MPI jest bardzo szeroko wykorzystywanym środowiskiem implementacji równoległej. W chwili obecnej wyszukiwarki artykułów naukowych znajdują ponad 200 tysięcy tytułów artykułów zawierających frazę "Message Passing Interface". Uznaje się, że MPI jest aktualnie faktycznym standardem dla implementacji obliczeń równoległych w środowisku rozproszonym.

5.3.1 Środowisko MPI

Od końca lat 80-tych wiele grup badawczych prowadziło badania i próby stworzenia wydajnego systemu komunikacji dla systemów z pamięcią rozproszoną. W efekcie tych prac powstały takie projekty, jak nCUBE [102], LAM [22], PICL [55], PVM [54] czy biblioteka P4 [25]. Okazało się jednak że twórcy tych systemów opracowywali i rozwiązywali te same zagadnienia, przez co duplikowali swoje wysiłki. W wyniku tego wniosku, na konferencji Supercomputing 92, większość z nich postanowiła opracować wspólny standard komunikacji w środowisku rozproszonym. Wysiłki te zaowocowały powstaniem standardu MPI, którego specyfikację w wersji 1.0 ukończono w roku 1994, a jej pierwsze implementacje ukazały się już w 1996 roku. Od tego czasu standard ten stał się powszechnie wykorzystywanym w obliczeniach naukowych i zastosowaniach komercyjnych wymagających obliczeń na superkomputerach. Jednym ze znanych projektów wykorzystujących MPI jest Earth Simulator [131]. W roku 1998 opublikowano wersję 2.0 specyfikacji, która wzbogacała środowisko o nowe funkcjonalności. W wersji tej wprowadzono komunikację jednostronną, dynamiczne zarządzanie procesami, rozszerzono komunikację grupową, dodano interfejs dla języka C++ oraz dodano równoległą specyfikację podsystemu I/O. Liczba funkcji dostępnych w MPI zwiększyła się z około 90 w wersji 1.2 do blisko 500 w wersji 2.0. Jednak z powodu skomplikowania i gwałtownego wzrostu rozmiaru specyfikacji w wersji 2.0 pierwsza jej implementacja ukazała się dopiero w roku 2002. Kolejnych implementacji 2.0 jest bardzo niewiele. Również migracja użytkowników MPI do wersji 2.0 następuje bardzo powoli. Powodem tego są m.in. obawy i niechęć do wprowadzania poprawek do już działających programów, które mogłoby zaowocować pojawieniem się nowych błędów oraz problemy z uruchamianiem programów dynamicznie zarządzających procesami w powszechnie wykorzystywanych systemach kolejkowych.



Rysunek 5.5: Przykład działania niektórych funkcji MPI służących do komunikacji grupowej.

Dostępne w środowisku MPI funkcje można podzielić pod względem ich funkcjonalności na 3 grupy: przesyłanie komunikatów, zbieranie informacji o środowisku oraz kontrola systemu wykonania równoległego. Z grupy funkcji kontrolujących należy wymienić funkcję `MPI::Init()`, która inicjalizuje komunikację między procesami oraz skojarzoną z nią funkcję `MPI::Finalize()` kończącą tę komunikację. Wywołanie tych funkcji jest konieczne przed rozpoczęciem jakiejkolwiek wymiany komunikatów. Z funkcji dostarczających informacje o środowisku można wymienić `Comm.Get_size()` zwracającą liczbę uruchomionych procesów oraz funkcję `Comm.Get_rank()` podającą identyfikator bieżącego procesu.

Podstawowym pojęciem w środowisku MPI jest komunikator. Jest to obiekt, który łączy między sobą grupę procesów umożliwiając komunikację pomiędzy nimi. Predefiniowany, zawsze obecny komunikator o nazwie `COMM_WORLD` grupuje wszystkie uruchomione procesy. Programista w ramach jednego programu może zdefiniować wiele komunikatorów realizując w ten sposób różne typy i różne topologie połączeń. W poniższym opisie ograniczono się do języka C++.

Specyfikacja MPI zawiera wiele funkcji pozwalających programiście na skonstruowanie komunikacji pomiędzy procesami. Komunikację można podzielić na komunikację punkt-punkt, w której wymiana komunikatów odbywa się pomiędzy dwoma, i tylko dwoma procesami, oraz na komunikację grupową, która dotyczy wielu procesów, na ogół w ramach wcześniej zdefiniowanej grupy lub komunikatora.

Komunikacja punkt-punkt jest realizowana poprzez funkcje `MPI::Send()` oraz `MPI::Receive()` lub poprzez ich liczne odmiany. Inne wersje tych funkcji służą do ustanawiania różnych typów komunikacji, takich jak komunikacja synchroniczna, asynchroniczna, blokująca, nieblokująca czy buforowana. Szczegóły poszczególnych funkcji zostały omówione w specyfikacji środowiska. Do programisty należy odpowiednie zaprojektowanie komunikacji i wybór odpowiednich funkcji do jej realizacji.

Podstawową funkcją używaną w komunikacji grupowej jest funkcja `MPI::Bcast()`, która wysyła komunikat do wszystkich innych procesów w ramach komunikatora. Innymi funkcjami realizującymi komunikację grupową są m.in. `MPI::Reduce()`, `MPI::Allreduce`, `MPI::Scatter()`, `MPI::Gather()` czy `MPI::Allgather`. Schematycznie działanie tych funkcji zostało przedstawione na rysunku (5.5). W przypadku, gdy programista chce dokonać synchronizacji procesów, wówczas wywołuje funkcję `MPI::Barrier()`, która ustawia barierę w danym miejscu programu. Bariera oznacza, że sterowanie przejdzie do następnych instrukcji tylko w przypadku, gdy wszystkie procesy z grupy wywołają tę funkcję.

Kolejną cechą środowiska MPI jest ukrywanie przed programistą ścieżki przepływu ko-

munikatów. Cecha ta szczególnie ujawnia się przy komunikacji grupowej, kiedy te same komunikaty mają dotrzeć do wielu procesów, a sama komunikacja grupowa oparta jest na przesyłaniu komunikatów typu punkt-punkt. Programista wywołując funkcję `MPI::Send()` bądź `MPI::Bcast()` ma pewność, że komunikat dotrze w miejsce docelowe, natomiast to biblioteka MPI dba o maksymalną wydajność komunikacji poprzez korzystanie odpowiednich algorytmów.

Środowisko MPI pozwala również programiście na definiowanie typów złożonych na użytek konkretnej implementacji. Zapewnia ono, że tak zdefiniowany typ będzie niezależny od platformy (kolejności bajtów), na której będzie działał program. Definiowanie własnych typów znacznie ułatwia projektowanie bardziej złożonej komunikacji pomiędzy procesami.

MPI udostępnia jeszcze szereg innych funkcji umożliwiających takie cechy jak pakowanie danych czy konstruowanie wirtualnych topologii. Ponieważ jednak cechy te nie zostały wykorzystane w pracy, nie są tutaj omawiane. Ich szczegółowy opis można znaleźć w specyfikacji środowiska.

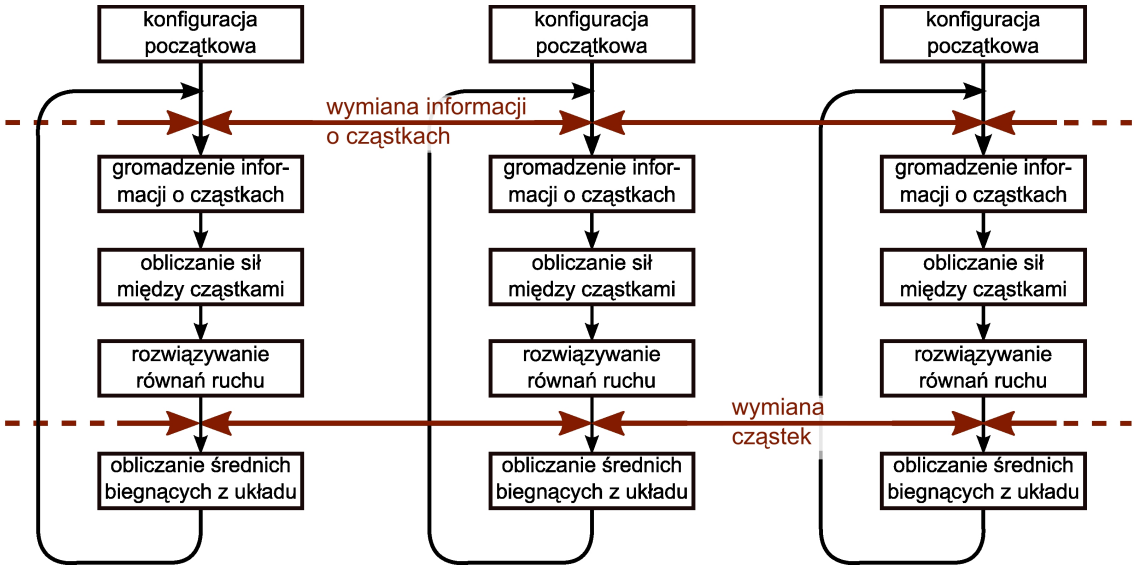
Powszechnie używanymi i bezpłatnymi implementacjami MPI są LAM/MPI [23] oraz MPICH [66]. Jedną z rozwojowych implementacji jest OpenMPI [61], jednak w powszechnej opinii nie osiągnęła ona jeszcze swojego stanu dojrzałego. Oprócz tego, wielu producentów wieloprocesorowych systemów komputerowych udostępnia własne imeplementacji MPI wraz z udostępnianymi kompilatorami przeznaczone na dostarczane przez siebie architektury [79].

5.3.2 Implementacja z wykorzystaniem środowiska MPI

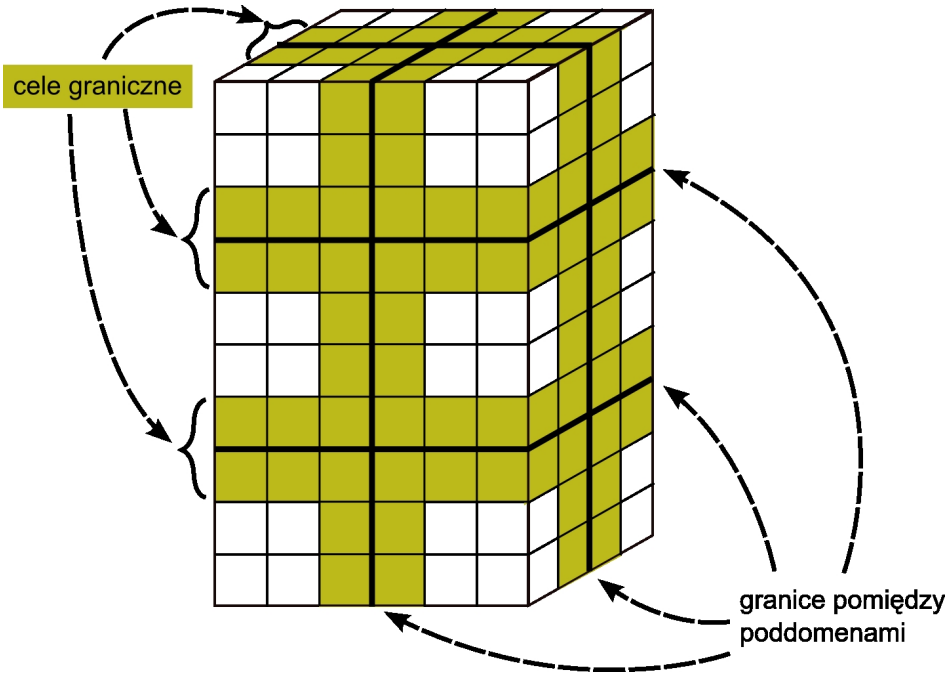
Dla implementacji równoległej algorytmu na architekturze z pamięcią rozproszoną konieczna jest jego dekompozycja na części przeznaczone dla każdej jednostki obliczeniowej. W przypadku symulacji płynów metodą cząstek podział ten realizuje się poprzez dekompozycję domenową, czyli podzielenie pudła obliczeniowego na rozłączne poddomeny. Każdy procesor wykonuje obliczenia tylko dla danych z poddomeny jaka została mu przypisana. Ponieważ cząstki poruszają się po całym pudle obliczeniowym oraz oddziałują między sobą również poprzez granice poddomen, konieczne jest przesyłanie informacji pomiędzy poszczególnymi procesorami. Dokonuje się tego za pomocą komunikatów. Schemat algorytmu symulacji z wykorzystaniem przesyłania komunikatów został przedstawiony na rysunku 5.6, natomiast podział pudła obliczeniowego na poddomeny został przedstawiony na rysunku 5.7. Cele, z których informacja musi być wymieniona pomiędzy poddomenami (procesami) zostały oznaczone kolorem. Poniżej, na listingu 5.3, znajduje się uproszczony kod źródłowy pętli głównej zrównoleglonej przy pomocy funkcji środowiska MPI wraz z krótkim komentarzem.

Fragment kodu źródłowego 5.3: Zrównoleglenie pętli głównej z wykorzystaniem mechanizmu wymiany komunikatów.

```
for ( ; stepCounter < totalSteps ; ) {

  //---------------------------------------------------------------
  // GATHERING
  exportClones();

  for ( i1=0 ; i1 < lCells.size() ; i1++ ) {
    lCells[i1]->prtcls26C.clear();

    for ( i2=0 ; i2 < 27 ; i2++ ) {
      tmpCell = lCells[i1]->neighsC[i2];
      if ( ( tmpCell == NULL ) || ( tmpCell == lCells[i1] ) ) continue;
      /* dodatkowe obliczenia, aby pary czastek sie nie powtarzaly */
      lCells[i1]->prtcls26C.add(tmpCell->prtclsC.ptr(),
                                tmpCell->prtclsC.size() );
    }
  }

```



Rysunek 5.6: Schemat wersji równoległej algorytmu symulacji dla komputerów z pamięcią rozproszoną.



Rysunek 5.7: Podział pudła obliczeniowego na poddomeny.

```
  //-------------------------------------------------------------------
  // FORCES
  for ( i1 = 0 ; i1 < lCells.size() ; i1++ ) {
    for ( i2=0 ; i2 < lCells[i1]->prtclsC.size() ; i2++ ) {
      for ( i3=0 ; i3 < lCells[i1]->prtclsC.size() ; i3++ )
        if ( lCells[i1]->prtclsC[i2] < lCells[i1]->prtclsC[i3] )
          force( lCells[i1]->prtclsC[i2], lCells[i1]->prtclsC[i3] );
      for ( i3=0 ; i3 < lCells[i1]->prtcls26C.size() ; i3++ )
        force( lCells[i1]->prtclsC[i2], lCells[i1]->prtcls26C[i3] );
    }
  }

    removeClones();

  //-------------------------------------------------------------------
  // MOTION

  for ( i1=0 ; i1 < cells.size() ; i1++ ) {
    cells[i1]->prtclsC.clear(); cells[i1]->prtcls26C.clear();
  }
  for ( i1 = 0 ; i1 < prtcls.size() ; i1++ ) {
    /* wyznaczanie nowych wartości polozenia, przyspieszenia itp dla
          czastki o numerze i1 */
    index <- numer celi w ktorej znajduje sie czastka i1 0;
    prtcls[i1]->iamfrom( cells[index] );
    cells[index]->prtclsC.push_back( prtcls[i1] );
  }

  exportParticles();

  //-------------------------------------------------------------------
  // STATISTICS

  for ( i1 = 0 ; i1 < prtcls.size() ; i1++ ) {
    /*
    energia kinetyczna, entropia i temperatura kazdej czastki
    sa dodawane do zmiennych globalnych dla wszystkich czastek: EKin,
    Entropy, Temperature
    */
  }

  MPI::COMM_WORLD.Allreduce( &LeKin, &EKin, 1, MPI::DOUBLE, MPI::SUM);
  MPI::COMM_WORLD.Allreduce( &Lentropy, &Entropy, 1, MPI::DOUBLE, MPI::SUM);
  MPI::COMM_WORLD.Allreduce( &Ltemperature, &Temperature, 1, MPI::DOUBLE,
                                  MPI::SUM);

  if ( process.id == MPI_MASTER ) {
    Temperature /= prtcls.size();
    cout << "Ekin=" << EKin << "\nEntropy=" << Entropy
      << "\nTemperature=" << Temperature << endl;
    LeKin = 0.0; Lentropy = 0.0; Ltemperature = 0.0;
    EKin = 0.0; Entropy = 0.0; Temperature = 0.0;
  }

  //-------------------------------------------------------------------
  if ( process.id == MPI_MASTER )
    cout << "\nSTEP: " << stepCounter << "\n";
  stepCounter++;
} // main loop
```

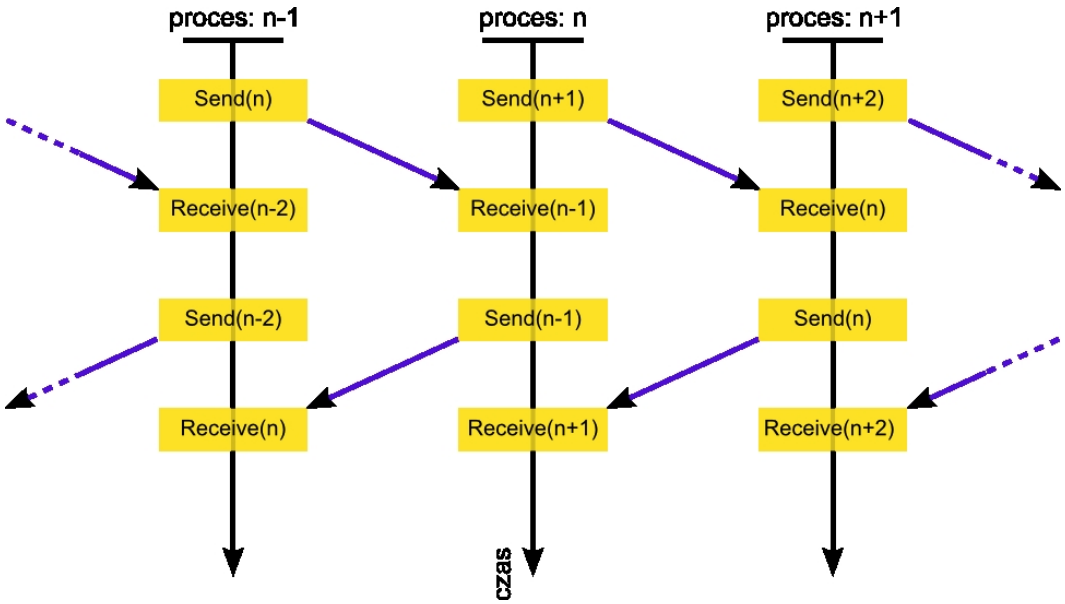
Konieczność komunikacji pomiędzy procesami wynika z charakteru symulacji metodą cząstek. Dwie istotne z tego punktu widzenia cechy takich symulacji to oddziaływanie cząstek na odległość oraz poruszanie się cząstek w pudle obliczeniowym. Przy podziale pudła obliczeniowego na poddomeny naturalnym jest, że poruszające sie cząstki będą przekraczały granice pomiędzy poszczególnymi poddomenami. Przykładowo, w i-tym kroku symulacji cząstka o numerze k przekracza granicę pomiędzy domenami z domeny A do domeny B. Dotychczas cząstka ta znajdowała się w domenie A i obliczenia dla niej przeprowadzał procesor p_A, a dane dotyczące tej cząstki znajdowały się w dostępnej dla niego pamięci ope-

nazwa zmiennej	typ zmiennej	opis zmiennej
`lCells`	`Cell*`	tablica zwierająca tylko cele należące do bieżącego procesu. Tablica `cells` zawiera cele składające się na całe pudło obliczeniowe.
`LeKin`	`double`	lokalna energia kinetyczna. Suma energii kinetycznych cząstek należących w danym kroku czasowym do bieżącego procesu.
`Lentropy`	`double`	lokalna entropia.
`Ltemperature`	`double`	lokalna temperatura.
`prtcls`	`Particle *`	tablica cząstek należących jedynie do aktualnego procesu.

Tabela 5.2: Zmienne wprowadzone w implementacji równoległej wykorzystującej środowisko MPI.

racyjnej. Wraz z przekroczeniem granicy wszystkie informacje o cząstce k muszą zostać usunięte z pamięci procesora p_A, przesłane do procesora p_B i wpisane w odpowiednie struktury danych znajdujące się w pamięci tego procesora. Operacje te są realizowane w procedurze `exportParticles()`. Instrukcje dotyczące usuwania i dodawania informacji o cząstkach w odpowiednich strukturach danych nie są istotne i nie zostały tutaj omówione. Schemat komunikacji przesyłającej dane cząstki k z poddomeny A do poddomeny B przedstawiono na rysunku (5.8). Ponieważ poszczególne procesy są logicznie ułożone w kartezjańską siatkę, każdy z nich posiada 26 sąsiadów, tj. procesów sąsiednich, z którymi wymienia komunikaty. Dla każdego procesu komunikacja ta jest przeprowadzana w każdym z 13 kierunków, za każdym razem z dwoma sąsiednimi procesami. Dzięki skorzystaniu z funkcji komunikacji asynchronicznej (nieblokujących) i wywoływaniu ich w kolejności przedstawionej na rysunku (5.8) uzyskuje się pewność, że nie dojdzie do zablokowania (zakleszczenia) wątków i wszystkie komunikaty dotrą do swojego miejsca przeznaczenia.

Kolejną cechą symulacji metodą cząstek, istotną z punktu widzenia implementacji równoległej dla architektury z pamięcią rozproszoną, jest oddziaływanie cząstek na odległość. Chcąc bowiem obliczyć wartość siły działającej na cząstkę k znajdującą się na tyle blisko granicy pomiędzy poddomenami A i B, że jej zasięg oddziaływania wykracza poza tę granicę, trzeba obliczyć oddziaływania cząstki k z cząstkami znajdującymi się w sąsiedniej poddomenie. Niestety, informacje o tych cząstkach nie są dostępne dla aktualnego procesora. Dlatego też muszą zostać one przesłane z sąsiedniej poddomeny. Aby być pewnym, że wszystkie potrzebne informacje o cząstkach zostaną przesłane, wystarczy przesłać informacje o wszystkich cząstkach z cel przylegających do granicy pomiędzy poddomenami. Wynika to z rozmiaru cel, który jest niemniejszy od wartości największego promienia obcięcia w modelowanym układzie. Procedura odpowiadająca za przesyłanie tych informacji to znajdująca się w linii 5 funkcja `exportClones()`. Jej struktura jest bardzo podobna do struktury funkcji `exportParticles()` i dlatego nie będzie tutaj prezentowana. Jedyną różnicą jest ograniczenie przesyłanych informacji do wyłącznie takich, które są potrzebne podczas obliczania sił oraz to, że informacje te nie są usuwane z oryginalnej poddomeny. Po wykorzystaniu tych informacji przy obliczaniu sił informacje te są usuwane przy pomocy funkcji `removeClones()`.



Rysunek 5.8: Schemat komunikacji w procedurze `exportParticles()`.

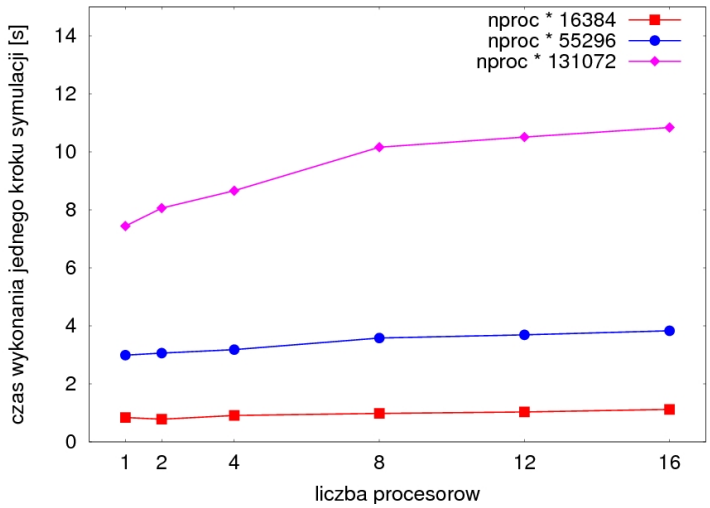
Dodatkowo, w celu obliczenia wielkości statystycznych modelowanego układu konieczne było wprowadzenie komunikacji grupowej. Początkowo, dla każdego podukładu obliczane są wielkości charakterystyczne tylko dla cząstek tego podukładu. Następnie, wielkości te były sumowane dla wszystkich procesów przy pomocy funkcji `Allreduce()` oraz operatora `SUM`. W ten sposób możliwe było obliczenie energii kinetycznej, entropii i temperatury dla całego modelowanego układu.

5.3.3 Efektywność implementacji

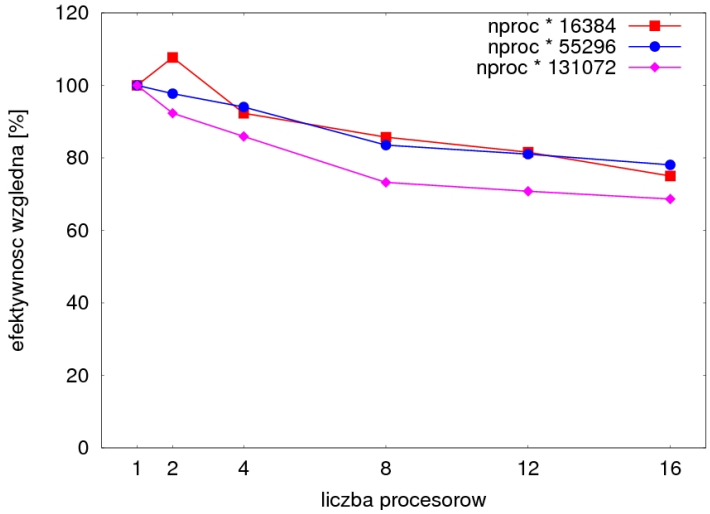
Biorąc powyższe pod uwagę dokonano implementacji równoległej algorytmu symulacji metodą SPH z wykorzystaniem środowiska MPI. Uzyskany w ten sposób algorytm może posłużyć do symulacji układu cząstek dowolnym modelem oddziaływania krótkozasięgowego. Wystarczy w tym celu jedynie zmodyfikować sposób obliczania sił oddziaływania.

W celu wyznaczenia efektywności implementacji równoległej wykonane zostały trzy symulacje przepływu płynu przez podłużne naczynie. Symulacje te różniły się pomiędzy sobą średnią liczbą cząstek na jeden procesor (odpowiednio: 16384, 55296 oraz 131072). Każda wersja symulacji została wykonana kilka razy, za każdym razem dla innej liczby procesorów. Dzięki temu możliwe było oszacowanie wydajności implementacji równoległej poprzez wyznaczenie jej efektywności. Otrzymane wyniki zostały przedstawione na rysunkach (5.9) oraz (5.10).

Na rysunku (5.9) przedstawiony został średni czas wykonania jednego kroku symulacji. Z wykresu tego można wywnioskować, że czas ten jest w przybliżeniu stały wraz ze wzrostem liczby procesorów wykorzystanych w symulacji, dla ustalonej liczby cząstek przypadającej na jeden procesor. Dodatkowy, niewielki wzrost można przypisać narzutowi komunikacyjnemu pomiędzy poszczególnymi jednostkami obliczeniowymi. Natomiast na rysunku (5.10) przedstawiona została efektywność względna tych symulacji. Z rysunku tego można odczytać, że efektywność względna nie zależy od rozmiaru problemu i dla wszystkich trzech przypadków zachowuje się podobnie wraz ze wzrostem liczby procesorów.



Rysunek 5.9: Czas wykonania programu symulacji w wersji dla komputerów z pamięcią rozproszoną, wykorzystującej środowisko MPI, w funkcji liczby procesorów.



Rysunek 5.10: Efektywność względna implementacji równoległej wykorzystującej środowisko MPI.

5.4 Optymalizacja algorytmu wykorzystującego środowisko MPI

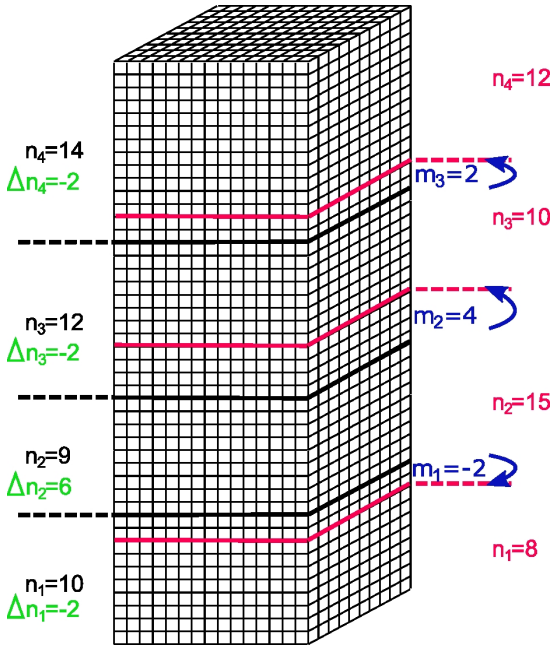
Przedstawiony w podrozdziale 5.3 algorytm równoległy przeznaczony na architektury z pamięcią rozproszoną opiera się na dekompozycji domenowej. Oznacza to, że obszar pudła obliczeniowego jest podzielony na rozłączne domeny, z których każda jest przyporządkowana do innej jednostki obliczeniowej. Poszczególne jednostki wymieniają pomiędzy sobą dane za pomocą przesyłanych komunikatów. Komunikaty przesyłane są pomiędzy procesorami tylko wówczas, gdy jest to niezbędne, a wielkość przesyłanych danych została ograniczona do minimum. Od wielkości danych i częstości ich przesyłania zależy czas, jaki program symulacji przeznacza na komunikację pomiędzy procesorami. Wielkość przesyłanych danych zależy od sposobu podziału pudła obliczeniowego na domeny. Podział ten jest optymalny wówczas, gdy granice pomiędzy domenami mają postać płaszczyzn równoległych do boków pudła obliczeniowego. Każdy inny podział zwiększa ilość przesyłanych danych. Istnieje również inna możliwość optymalizacji wykonania całego programu równoległego. Może to zostać zrealizowane poprzez odpowiednie rozmieszczenie płaszczyzn podziału uwzględniające możliwości obliczeniowe i wielkość danych poszczególnych procesorów. Podział ten może zostać odpowiednio dobrany na początku symulacji w przypadku, gdy dostępne są informacje dotyczące mocy obliczeniowej poszczególnych jednostek oraz gęstości cząstek w poddomenach przyporządkowanych tym jednostkom. W przypadku jednak, gdy dostępna moc obliczeniowa na poszczególnych procesorach może się zmieniać wraz z czasem trwania symulacji bądź też cząstki, przemieszczając się pomiędzy jednostkami zmieniają swoją gęstość, w celu optymalizacji czasu wykonania algorytmu równoległego konieczne jest wyposażenie implementacji w mechanizmy równoważenia obciążenia. Procedura ta polega na dynamicznym dostosowywaniu podziału pudła obliczeniowego na domeny tak, aby czasy wykonania kroków obliczeniowych na poszczególnych jednostkach różniły się od siebie jak najmniej. W przypadku takim sytuacje, w których jedna jednostka po wykonaniu swojej porcji obliczeń czeka na wykonanie obliczeń przez inne jednostki są ograniczone do mimimum.

W opisywanym przypadku dynamiczne równoważenie obciążenia polega na zmianie położenia granic pomiędzy poddomenami. W ten sposób zmienia się ilość cel przyporządkowanych do poszczególnych procesorów, a co za tym idzie zmienia się ilość cząstek, dla których wybrany procesor ma przeprowadzać obliczenia. Dzięki temu możliwa jest zmiana czasu obliczeń dla procesorów, a przez to optymalizacja wykonania programu przez wszystkie procesory.

Dla każdego z trzech wymiarów granice pomiędzy poszczególnymi poddomenami mają postać równoległych płaszczyzn. Skutkiem tego założenia jest fakt, że rozpatrywane domeny mają kształt prostopadłościanu oraz to, że zmieniając położenie jednej płaszczyzny zmienia się granice wszystkich domen, które ona rozdziela. W przypadku, gdy pudło obliczeniowe jest podzielone na poddomeny tylko wzdłuż jednego wymiaru, oznacza to zmianę granic tylko dwóch domen. Jeśli podział pudła na poddomeny przebiega wzdłuż dwóch lub trzech wymiarów, liczba tych poddomen jest odpowiednio większa.

Poniżej zamieszczono opis procedury równoważenia obciążenia dla przypadku, gdy podział na poddomeny został wykonany tylko wzdłuż jednego wymiaru. Ideę przedstawiono na rysunku (5.11). Rozszerzenie algorytmu dla przypadku jednowymiarowego na większą liczbę wymiarów nie jest skomplikowane i zostanie opisane w dalszej kolejności na podstawie poniższego, prostszego przypadku.

Aby odpowiednio dobrać podział pudła obliczeniowego konieczna jest znajomość aktualnego obciążenia każdego z wykorzystywanych w obliczeniach procesorów. W tym celu, w czasie trwania obliczeń dla każdego procesora i obliczany oraz sumowany w kolejnych krokach obliczeniowych jest czas wykonywania obliczeneń t_i. Na tej podstawie, co z góry



Rysunek 5.11: Przykład działania procedury równoważenia obciążenia. Zmiana podziału na domeny w jednym kierunku.

założoną liczbę kroków `lbSteps`, nastepuje sprawdzenie, czy spełniony jest warunek konieczny do zmiany podziału pudła obliczeniowego. W tym celu, obliczany jest średni czas wykonania dla wszystkich procesorów, $\bar{t} = (t_1 + t_2 + \ldots + t_N)/N$. Następnie oblicza się, o ile warstw cel należy zmienić obszar przyporządkowany rozpatrywanemu procesorowi:

$$\Delta n_i = \left\lfloor n_i \frac{\bar{t} - t_i}{(\bar{t} + t_i)/2} + \frac{1}{2} \right\rfloor, \tag{5.8}$$

gdzie n_i oznacza ilość warstw cel na procesorze o numerze i. W przypadku, gdy $\Delta n_i < 0$, wówczas procesor i jest zbyt obciążony obliczeniami w porównaniu do innych procesorów i konieczne jest w takim przypadku przekazanie „nadwyżki" warstw do innych procesorów. W przeciwnym przypadku zasoby jednostki obliczeniowej nie są wykorzystywane w pełni i aby to zmienić należy „przyjąć" od innych jednostek dane do obliczeń. Wartości wyznaczone dzięki powyższemu wzorowi mówią, ile warstw cel należy wymienić z innymi procesorami. Nawet przez cały czas trwania symulacji liczby te mogą być równe zeru. Jeśli jednak choć jedna z nich jest różna od zera, wówczas należy zmienić aktualny podział pudła obliczeniowego na domeny.

Przed przystąpieniem do zmiany granic domen konieczne jest jeszcze zmodyfikowanie liczb obliczonych powyższym wzorem tak, aby spełniały pewne warunki. Jeśli $\Delta n_i > 0$, wówczas procesor i jest procesorem odbierającym, w przeciwnym przypadku procesorem wysyłającym. Ponieważ ilość warstw cel jest stała dla każdego z trzech kierunków, to liczba cel, z których dane będą wysłane i liczba cel, z których dane należy odebrać muszą być sobie równe:

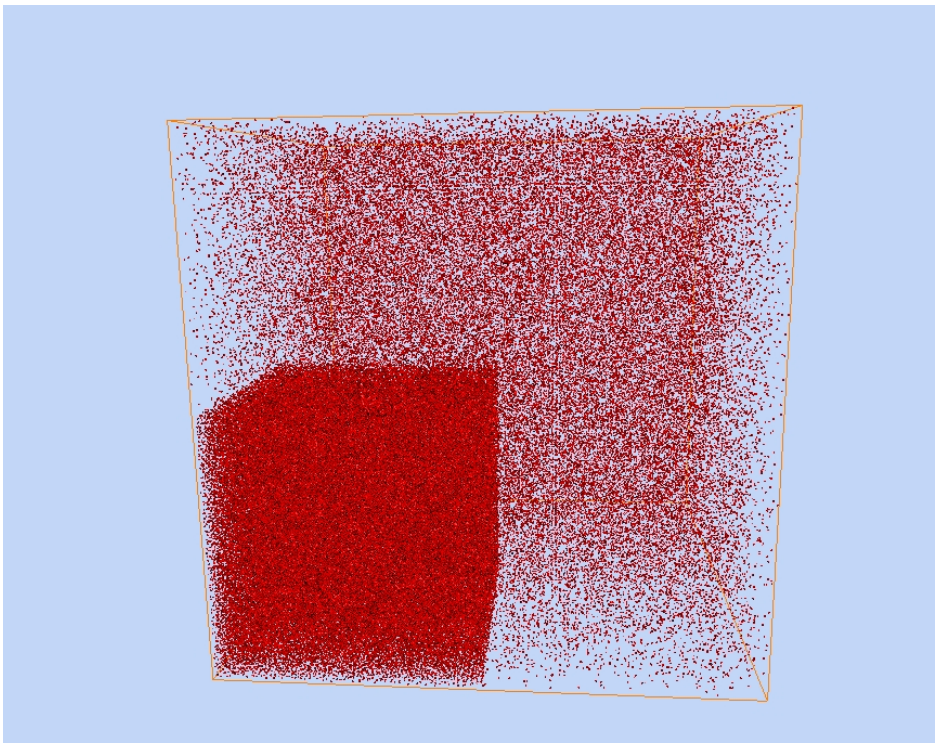
$$\left| \sum_{i \in Sending} \Delta n_i \right| = \left| \sum_{i \in Receiving} \Delta n_i \right|. \tag{5.9}$$

Warunek ten zazwyczaj nie jest spełniony dla liczb wyznaczonych zgodnie ze wzorem (5.8) i dlatego konieczna jest ich modyfikacja. W tym celu do wszystkich obliczonych liczb Δn_i

dodaje się taką samą liczbę całkowitą tak, aby ich suma $\sum_i \Delta n_i$ była równa zero. W tym przypadku, dobiera się liczbę w taki sposób, aby moduł sumy był najmniejszy, a następnie dodatkowo modyfikuje się część z liczb Δn_i o ± 1 tak, aby powyższy warunek był spełniony. Jeśli możliwych jest kilka rozwiązań powyższej procedury, wybiera się takie, dla którego suma $\sum |\Delta n_i|$ jest najmniejsza. Dzięki temu również liczba warstw cel do przesłania jest najmniejsza. Implikuje to minimalny czas potrzebny na rekonfigurację systemu. Przykładowo dla otrzymanych ze wzoru (5.8) liczb: 0, 3 w wyniku tego przekształcenia otrzymuje się liczby: -1, 1. Po przeprowadzeniu powyższego przekształcenia należy jeszcze sprawdzić, czy dla wszystkich procesorów spełniony jest warunek $\Delta n_i \geqslant n_i$. W przeciwnym wypadku, podczas przeprowadzania procedury równoważenia obciążenia dwie przeciwległe granice domeny zamienią się kolejnością i liczba wartstw cel dla procesora i przyjmie niedopuszczalną wartość mniejszą od zera. Aby tego uniknąć, w przypadku spełnienia powyższego warunku dla procesora o numerze k, liczbę Δn_k inkrementuje się odpowiednią liczbę razy, dekrementując jednocześnie za każdym razem największą z liczb dla pozostałych procesorów: $\Delta n_j = \max\{\Delta n_i, i = 0 \ldots n-1\}$. Otrzymane w wyniku powyższych przekształceń liczby Δn_i mówią dokładnie, o ile należy zmienić liczbę wartstw cel dla poszczególnych procesorów.

Po obliczeniu wielkości Δn_i, kolejną czynnością w ramach procedury równoważenia obciążenia jest wyznaczenie liczb warstw cel, o które należy przesunąć każdą granicę pomiędzy domenami. Oznaczmy granicę pomiędzy domenami o numerach i oraz $i+1$ numerem i, a liczbę warstw cel, o którą należy ją przesunąć, przez m_i. Dodatkowo, niech granica pomiędzy domenami o numerach 0 oraz $n-1$ ma numer $n-1$. Nie da się wyznaczyć liczby m_i na podstawie samych tylko liczb Δn_i oraz Δn_{i+1}, ani tym bardziej liczb m_{i-1} oraz m_i na podstawie wartości liczby Δn_i. Wynika to z zależności $\Delta n_i = m_i - m_{i-1}$. W celu obliczenia liczb m_i konieczne jest uwzględnienie wszystkich liczb Δn_i oraz dodatkowych założeń, takich jak warunki brzegowe. W przypadku nieperiodycznych warunków brzegowych rozwiązanie tego problemu jest jednoznaczne i względnie łatwo jest je wyznaczyć. Warunkiem równoważnym nieperiodycznych warunków brzegowych jest $m_{n-1} = 0$. Korzystając z tego warunku można wyznaczyć wszystkie pozostałe liczby m_i w kolejności $i = 0 \ldots n-2$ na podstawie równania $m_i = \Delta n_i + m_{i-1}$ W przypadku periodycznych warunków brzegowych rozwiązanie dla warunków nieperiodycznych jest jedynie jednym z wielu możliwych i służy jako punkt wyjścia dla wyznaczenia innych rozwiązań. Inne rozwiązania można uzyskać dodając do wszystkich liczb m_i tę samą liczbę całkowitą f. Bez warunku określającego nieperiodyczne warunki brzegowe uzyskuje się dodatkowy stopień swobody i przez to można określić inny warunek, jaki powinien być spełniony przy podziale pudła na poddomeny. Przykładowo, może to być żądanie, aby całkowita liczba przesyłanych warstw cel była minimalna lub żeby środki ciężkości poszczególnych domen pozostawały nieruchome (dzięki temu domeny nie będą się przesuwały wraz z przesuwającymi się cząstkami). W pierwszym przypadku należy dobrać taką liczbę f, aby wartość sumy była minimalna $\sum |m_i|$), natomiast w drugim przypadku liczba f powinna minimalizować sumę $|\sum m_i|$. W obydwu przypadkach minimalizacja jest łatwa do przeprowadzenia, gdyż obie fukncje mają tylko dwa obszary swojej monotoniczności w funkcji zmiennej f. Wynika stąd, że ich minimum jest minimum globalnym. W opisywanej implementacji przetestowano obydwa dodatkowe warunki. Ostatecznie, w celu zminimalizowania czasu traconego na komunikację, przyjęto to, dla którego ilość wartsw cel do przesłania jest najmniejsza.

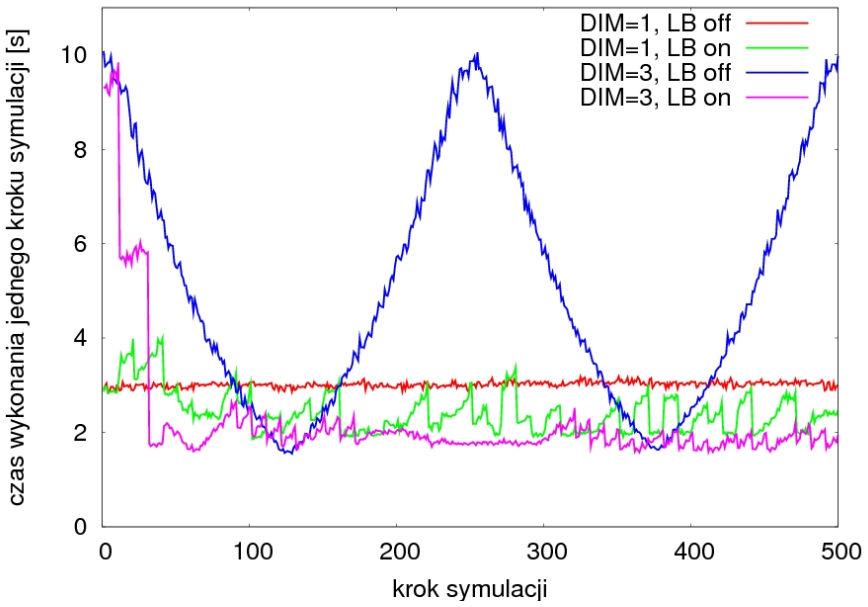
Przedstawione powyżej wyznaczenie liczb m_i jest ostatnią czynnością w ramach procedury równoważenia obciążenia. Po obliczeniu nowych granic pomiędzy domenami należy jeszcze przypisać cząstki z cel, które zostały wymienione pomiędzy domenami do odpowiednich procesorów. To zadanie jest realizowane automatycznie przez opisywaną wcześniej w podrozdziale 5.3 procedurę `exportParticles()`.



Rysunek 5.12: Konfiguracja początkowa układu dla testów funkcji równoważenia obciążenia.

Omówiona powyżej procedura dotyczy równoważenia obciążenia dla przypadku, gdy podział pudła obliczeniowego na domeny jest wykonany wzdłuż tylko jednego wymiaru. Rozszerzenie procedury na przypadek, gdy podziału dokonano dla większej liczby wymiarów jest nieskomplikowane. Jedyną zmianą jaką należy wprowadzić w implementacji jest to, że w opisywanej powyżej procedurze pojedynczą domenę należy zastąpić warstwą domen, posiadających taki sam indeks wzdłuż rozpatrywanego wymiaru. Istotną zmianą implementacyjną w tym przypadku jest to, że przed przystąpieniem do właściwych, opisanych powyżej obliczeń, należy przeprowadzić dla rozpatrywanej warstwy procesorów redukcję, aby móc obliczyć całkowity czas, jaki rozpatrywana warstwa procesorów przeznaczyła na obliczenia.

Dla zaprezentowania działania przedstawionej powyżej procedury równoważenia obciążenia przeprowadzono testową symulację. W symulacji tej pudło obliczeniowe zostało wypełnione cząstkami płynu w ten sposób, że w 1/8 jego części gęstość cząstek była pięciokrotnie wyższa niż w pozostałym obszarze pudła. Konfigurację początkową tej symulacji przedstawiono na rysunku (5.12). W symulacji wykorzystano periodyczne warunki brzegowe. Wszystkim cząstkom w symulacji nadano jednakową i stałą prędkość o kierunku wyznaczonym przez wektor o współrzędnych (1,1,1). Symulację przeprowadzono dla dwóch sposobów dekompozycji domenowej: wzdłuż jednego wymiaru (DIM=1) oraz dla trzech wymiarów (DIM=3). Dla każdego z tych przypadków przeprowadzono dwie symulacje: z włączoną (LB on) oraz wyłączoną (LB off) procedurą równoważenia obciążenia. Wyniki czasowe symulacji zostały przedstawione na rysunku (5.13). Przedstawiono na nim czas wykonania jednego kroku pętli głównej programu symulacji w sekundach wraz z postępem symulacji. Z wykresu tego widać, że procedura równoważenia obciążenia znacząco przyspiesza obliczenia. Dla przypadku trójwymiarowego wartość przyspieszenia jest równa ok. 5 dla najbardziej niekorzystnego przypadku. Dodatkowo, z rysunku tego można wywnioskować, o ile korzystniejszy jest podział pudła obliczeniowego na domeny w trzech wymiarach



Rysunek 5.13: Wyniki czasowe działania procedury równoważenia obciążenia.

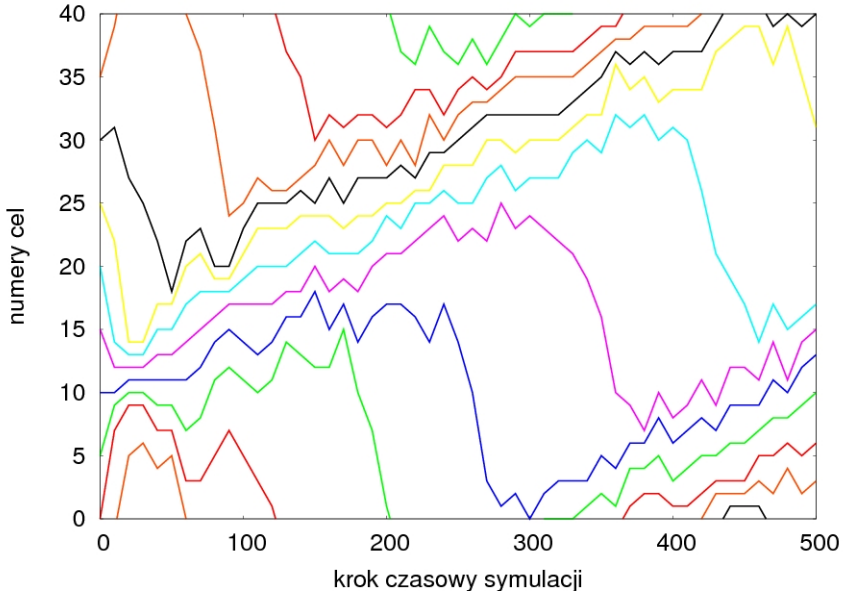
zamiast w jednym wymiarze, jeśli tylko wprowadzenie takiego podziału jest możliwe. W przypadku takim liczba cząstek do przesyłania jest o wiele mniejsza, niż w przypadku dla tylko jednego wymiaru. Z tego powodu narzut komunikacyjny w programie symulacji jest o wiele mniejszy.

Na rysunku (5.14) pokazano natomiast jak zmieniały się położenia granic pomiędzy poddomenami podczas wykonania opisywanej symulacji testowej dla przypadku jednowymiarowego. Z rysunku tego widać, że procedura równoważenia obciążenia działa w sposób zgodny z oczekiwaniami. Dopasowuje ona podział pudła obliczeniowego na poddomeny do zachodzących zmian gęstości cząstek. W obszarach, gdzie gęstość cząstek jest większa, procedura zwiększa „rozdzielczość" podziału pudła obliczeniowego na poddomeny.

5.5 Człon losowy w wersji równoległej MPI

W algorytmach nowoczesnych modeli cząstek dla komputerów z pamięcią rozproszoną należy zwrócić szczególną uwagę na implementację losowego charakter oddziaływań pomiędzy cząstkami. W szczególności dotyczy to opisywanych w niniejszej pracy modeli DPD oraz SDPD, w których oddziaływania pomiędzy cząstkami zależą od zmiennej losowej zgodnie z wzorami (3.5) oraz (3.35,3.36). Wzory te, wyprowadzone przy pomocy metod fizyki statystycznej i przy zachowaniu założeń odpowiednich dla każdego z modeli stanowią integralną część każdego z nich. Od ich prawidłowej implementacji zależy poprawność wyników otrzymywanych przy pomocy symulacji z wykorzystaniem wspomnianych modeli. Implementacja metody cząstek przeznaczona na architektury z pamięcią rozproszoną, w której nie występują oddziaływania zależne od zmiennej losowej, została przedstawiona w podrozdziale 5.3. Została tam opisana implementacja metody SPH, w której oddziaływania wystepujące pomiędzy cząstkami mają charakter hydrodynamiczny i nie występuje w nich człon zależny od zmiennej losowej. W przypadku modeli, w których występują oddziaływania stochastyczne, konieczna jest modyfikacja przedstawionej implementacji.

Konieczność wprowadzenia modyfikacji wynika z dekompozycji domenowej. W jej wyniku pudło obliczeniowe wraz ze znajdującymi się w nim cząstkami zostaje podzielone na



Rysunek 5.14: Zmiany granic pomiędzy poddomenami dla symulacji testowej. Równoważenie obciążenia wzdłuż jednego wymiaru.

rozłączne poddomeny, z których każda jest przydzielana oddzielnej jednostce obliczeniowej. Dopóki rozpatrywane są oddziaływania pomiędzy cząstkami znajdującymi się w tej samej domenie, dopóty możliwe jest stosowanie algorytmu obliczeń analogicznego jak dla modelu bez członu stochastycznego. Problem pojawia się natomiast wówczas, gdy konieczne jest obliczenie oddziaływania pomiędzy parą cząstek, z których każda znajduje się w oddzielnej poddomenie, czyli jest przetwarzana przez oddzielną jednostkę obliczeniową. W przypadku takim do sąsiednich domen przesyłane są kopie cząstek znajdujących się w sąsiedztwie granicy poddomen. Dzięki kopiom cząstek możliwe jest obliczenie sił działających na cząstki pochodzących od cząstek znajdujących się w sąsiednich poddomenach. W metodzie SPH, w której nie występują siły stochastyczne, siły obliczane dla tej samej pary cząstek na dwóch różnych jednostkach obliczeniowych są takie same. Niestety, nie jest to prawdą w przypadku metod DPD oraz SDPD, w których występują siły stochastyczne. W sytuacji takiej, w poszczególnych jednostkach obliczeniowych generowane są różne liczby pseudolosowe i przez to siły obliczane dla tej samej pary cząstek są różne. Przykładowo, dla cząstek a oraz b znajdujących sie w sąsiednich poddomenach, odpowiednio A oraz B, siła działająca na cząstkę a pochodząca od cząstki b, obliczana jako oddziaływanie z kopią b' będzie inna, niż siła działająca na cząstkę b pochodząca od cząstki a, obliczana jako pochodząca od kopii a'. Przyczyną tego będzie to, że w pierwszym przypadku do obliczenia oddziaływania wykorzystany zostanie generator liczb pseudolosowych na jednostce obliczeniowej M_A, a w drugim generator z jednostki M_B. Sytuacja taka jest niedopuszczalna, gdyż łamie ona III zasadę dynamiki Newtona. Efekty takiej sytuacji, w przypadku dużej liczby cząstek mogą być trudne do zauważenia, gdyż przypadkowy charakter sił powoduje, że nie jest widoczny żaden kolektywny ruch cząstek ani zmiany w zachowaniu się układu jako całości. Jednak niezachowanie wzorów wymaganych przez stosowany model może zaburzyć wyniki pod wzgledem termodynamicznym, tzn. mogą być one sprzeczne z założeniami fizycznymi leżącymi u podstaw stosowanego modelu. Może sie to objawiać, przykładowo, powolnym wzrostem wartości wielkości termodynamicznych układu. Wzory w odpowiednich metodach zostały wyprowadzone na podstawie ścisłej analizy matematycznej i podczas implementacji należy dołożyć wszelkich starań, aby te same rygotystyczne zasady dotyczyły implementa-

cji.

Poniżej zostały zaprezentowane trzy podejscia rozwiązujące ten problem. Następnie zostały one poddane analizie oraz przetestowane w poszczególnych symulacjach.

5.5.1 Propozycja 1

Głównym celem proponowanych rozwiązań powinno być zapewnienie, aby w celu obliczenia oddziaływania pary cząstek o tych samych numerach zostały wykorzystane te same liczby pseudolosowe na dwóch różnych jednostkach obliczeniowych. Jednym ze sposobów spełnienia tego warunku może być wykorzystanie dla każdej pary cząstek liczb losowych pochodzących z oddzielnego, dla każdej pary zdeterminowanego generatora. W ogólności, generator liczb pseudolosowych można traktować jako funkcję $\Phi(k, i)$, gdzie k jest ziarnem losowości, a i numerem kolejnej liczby z ciagu liczb pseudolosowych. Pierwsza propozycja proponuje, aby dla każdej pary cząstek określić algorytm określający k. Przykładowo, dla cząstek o numerach i oraz j, może to być funkcja:

$$k = \mathrm{bin}(r_i^x) \oplus \mathrm{bin}(r_i^y) \oplus \mathrm{bin}(r_i^z) \oplus \mathrm{bin}(r_j^x) \oplus \mathrm{bin}(r_j^y) \oplus \mathrm{bin}(r_j^z) \oplus \mathrm{bin}(t), \tag{5.10}$$

gdzie $\mathrm{bin}(\alpha)$ oznacza binarną reprezentację liczby zmiennoprzecinkowej α, a operator $\oplus$ oznacza operator `XOR`. Metoda ta generuje ziarno k na podstawie aktualnych położeń cząstek oraz bieżącego kroku czasowego.

Wadą tego rozwiązania jest to, że dla niewątpliwej wygody rezygnuje sie tutaj z matematycznej poprawności. Nie ma bowiem żadnej gwarancji ani metody dowodzącej, że tak generowane liczby losowe, jakkolwiek takie same dla dwóch jednostek obliczeniowych, będą spełniały testy, jakie powinny spełniać liczby pseudolosowe. Jest to spowodowane tym, że ciągi te zależą od reprezentacji zmiennoprzecinkowej oraz od użytych w symulacji jednostek. Przykładowo, może się okazać, że ziarno k będzie takie samo dla liczb t oraz $t + \Delta t$ i wówczas mogą się ujawinić korelacje pomiędzy poszczególnymi liczbami.

5.5.2 Propozycja 2

Polega ona na modyfikacji danych wymienianych w komunikatach pomiędzy dwoma procesami - poddomenami. Dotychczas, przed przystątpieniem do obliczania sił pomiędzy procesami przekazywane były potrzebne do obliczeń dane, tzw. kopie cząstek. Modyfikacja polega na dołączeniu do kopii każdej cząstki przesyłanej z procesu o większym identyfikatorze (proces A) do procesu o mniejszym identyfikatorze (proces B) odpowiedniej liczby liczb pseudolosowych oraz jednoczesne ich zapamiętanie w pamięci procesu A. Jeśli dodatkowo w implementacji wprowadzi się mechanizm zachowujący uporządkowanie cząstek w ramach struktury reprezentujących celę, możliwe wówczas będzie przyporządkowanie oddziałującej parze w procesie A: cząstka i - kopia cząstki j tych samych liczb pseudolosowych, co oddziałującej parze w procesie B: kopia cząstki i - cząstka j. W ten sposób wartości oddziaływania pomiędzy cząstkami i oraz j obliczone w procesach A oraz B będą takie same.

Propozycja ta wymaga znajomości liczby sąsiadów cząstki z procesu A, z jakimi będzie oddziaływała cząstka j z procesu B. Jest to konieczne, aby wygenerować odpowiednią ilość liczb pseudolosowych dla cząstki j i przesłać je razem z kopią cząstki j do procesu A. Niestety, dokładne obliczenie tej wartości jest niemożliwe przed przystąpieniem do obliczania oddziaływań. Można jednak założyć, co jest prawdą w większości występujących przypadków, że liczba ta będzie tego samego rzędu, co liczba oddziałujących sąsiadów w poprzednim kroku czasowym. W opisywanej implementacji założono, że liczba ta nie będzie większa niż dwukrotnie.

Propozycja ta korzysta z generatora liczb losowych, który nie jest zależny od położeń cząstek ani od aktualnego kroku czasowego. Dzięki temu zaimplementowane oddziaływania są zgodne z oddziaływaniami zadanymi wzorami właściwymi dla każdej metody i dzięki temu można oczekiwać poprawności otrzymywanych wyników.

5.5.3 Propozycja 3

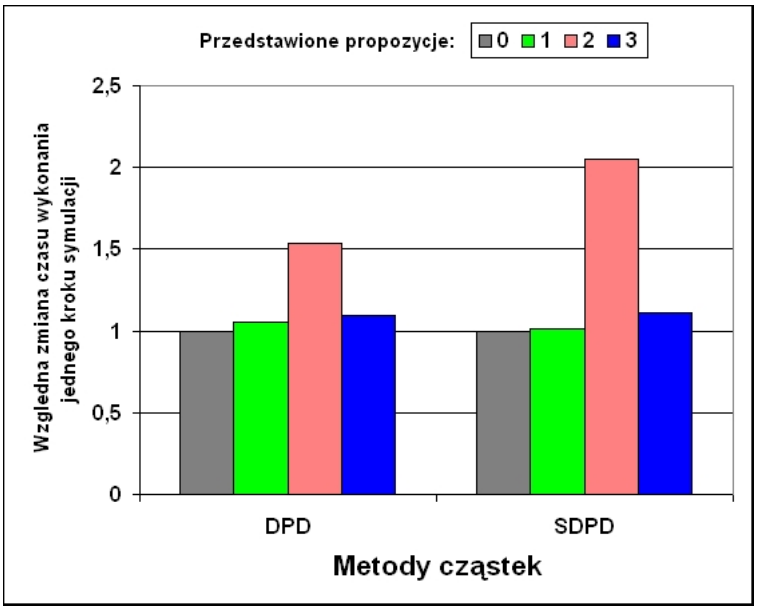
Trzecia propozycja polega na modyfikacji schematu komunikacji pomiędzy procesami. W jej ramach obliczenia oddziaływań pomiędzy parami cząstek znajdujących sie w dwóch różnych poddomenach są przeprowadzane tylko w jednym procesie (przykładowo w procesie o większym identyfikatorze). Następnie są one przesyłane do procesu o mniejszym identyfikatorze i dodawane do wartości dla odpowiednich cząstek. Dzięki temu wartości oddziaływań dla pary cząstek mają taką samą wartość na obydwóch jednostkach obliczeniowych. Rozwiązanie to wymaga zmian w dotychczasowym schemacie wymiany danych polagających na dodatkowej komunikacji mającej miejsce już po obliczeniu sił dla par cząstek. Komunikacja ta obejmuje przesyłanie kopi cząstek z powrotem do procesów o niższych identyfikatorach, a dokładniej obliczonych dla nich oddziaływań i dodanie ich do wartości obliczonych w ramach tych procesów.

Propozycja ta wprowadza asymetrię obliczeń. W dotychczasowych propozycjach oddziaływania pomiędzy parą cząstek były obliczane w ramach obydwóch procesów, do których należały cząstki. Ta propozycja przenosi obliczenia tylko do jednego z nich, kosztem jednak większego nakładu komunikacyjnego.

5.5.4 Porównanie propozycji i ich dyskusja

Wszystkie trzy opisane powyżej sposoby zostały zaimplementowane i przetestowane. Wyniki przedstawiono na rysunku (5.15), na którym przedstawiono względny wzrost czasu wykonania jednego kroku symulacji dla każdej z trzech propozycji (oznaczone odpowiednio: 1, 2 oraz 3) oraz dodatkowo wyniki czasowe dla przypadku, gdy stochastyczny charakter oddziaływań nie był uwzględniany (oznaczone przez 0). Rysunek ten przedstawia czasy wykonania poszczególnych części pętli głównej dla metod DPD oraz SDPD. W metodzie SDPD dla każdej pary oddziałujących cząstek w każdym kroku czasowym potrzebne jest wygenerowanie 28 liczb pseudolosowych o rozkładzie normalnym. Przekłada sie to na czas wykorzystywany na przesyłanie tych danych pomiędzy jednostkami obliczeniowymi w ramach propozycji drugiej. Dla metody SDPD o wiele szybsza, a zarazem efektywniejsza jest propozycja trzecia, pomimo iż wprowadza ona dodatkową komunikację do schematu wymiany danych. W porównaniu z propozycjami drugą i trzecią, propozycja pierwsza działa najszybciej, jednak z braku matematycznego dowodu jej poprawności powinna ona być stosowana jedynie w celach porównawczych.

Wady propozycji drugiej są mniejsze w przypadku metody DPD. W modelu tym dla każdej pary cząstek oddziałujących konieczne jest wygenerowanie tylko jednej liczby pseudolosowej. Stąd też przesyłanie liczb pseudolosowych pomiędzy sąsiednimi jednostkami obliczeniowymi nie jest tak kosztowne, jak w przypadku metody SDPD. Wciąż jednak, podobnie jak dla metody SDPD, w metodzie DPD propozycja 3 jest szybsza od propozycji 2 i powinna być stosowana w wymagających tego przypadkach.



Rysunek 5.15: Porównanie czasowe trzech zaprezentowanych propozycji dotyczących członu losowego w symulacjach z wykorzystaniem środowiska MPI. Opis w treści podrozdziału 5.5.4.

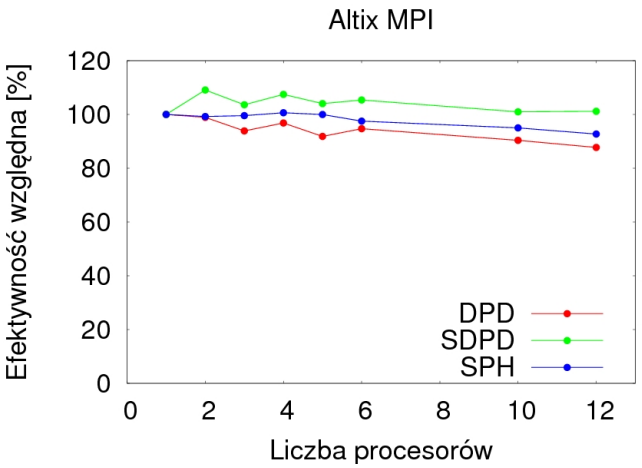
5.6 Porównanie efektywności implementacji wykorzystujących środowiska OpenMP oraz MPI

Porównanie pomiędzy dwoma modelami implementacji równoległej zostało wykonane na kilku komputerach. Były to:

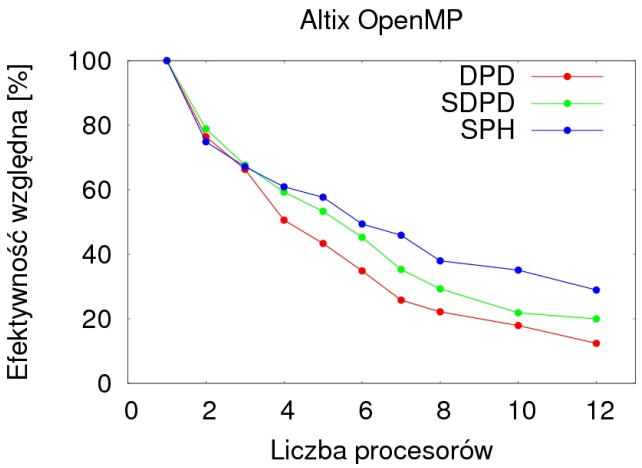
- SGI Altix 3700 (Altix),
- IBM Power4 (Power4),
- HP Proliant DL585 (Proliant),
- komputer wykorzystujący procesory AMD Opteron 270 Dual Core (Opteron).

W nawiasach podano nazwy, pod jakimi oznaczono wyniki uzyskane dla tych komputerów w dalszej części podrozdziału. Szczegółowy opis architektury każdej z nich został przedstawiony w dodatku A. Porównania dokonano nie tylko pod kątem różnych architektur, ale również dla trzech opisywanych w niniejszej pracy metod cząstek, tj. DPD, SPH oraz SDPD.

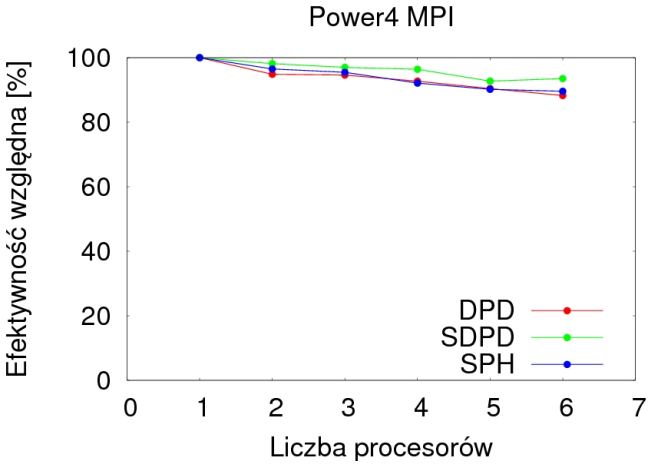
Na rysunku (5.16) przedstawiono efektywność względną algorytmów równoległych dla opisywanych metod zmierzoną na każdej z rozważanych architektur wyznaczoną zarówno dla implementacji z wykorzystaniem środowiska OpenMP, jak i MPI. Wykresy te wyznaczono dla każdej metody przy pomocy symulacji zachowania się jednorodnego płynu przeprowadzonej dla pudła obliczeniowego z nałożonymi periodycznymi warunkami brzegowymi. Wykresy te pokazują, że efektywność implementacji równoległej jest praktycznie taka sama dla trzech opisywanych metod. Wyniki otrzymane zarówno przy pomocy środowiska OpenMP jak i MPI nie zależą od zaimplementowanej metody. Dla metod SPH, SDPD oraz DPD są one praktycznie takie same. Wyniki te są zgodne z oczekiwaniami. W każdej metodzie część równoległa p programu, na którą składa się przede wszystkim procedura obliczania sił pomiędzy cząstkami, jest w przybliżeniu taka sama. Dlatego też wartości otrzymane dla efektywności względnej dla trzech metod są takie same.



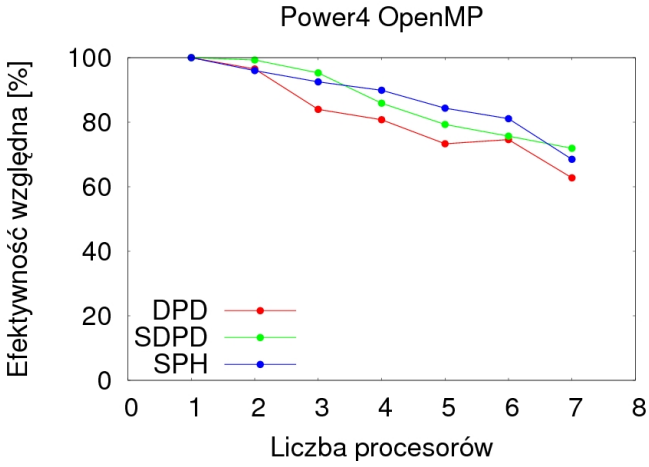
(a) Przypadek dla środowiska MPI, komputer Altix.



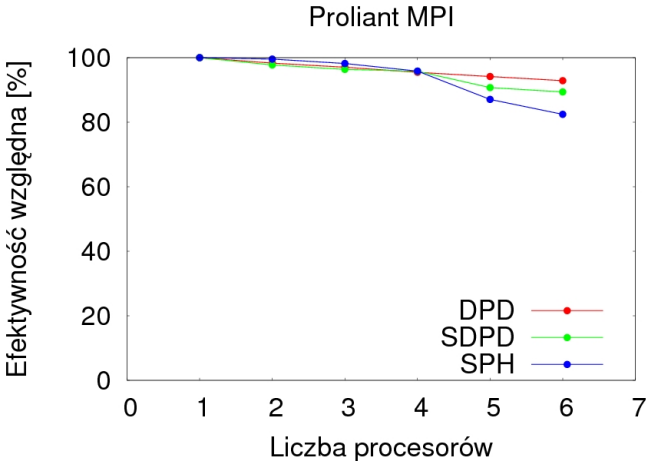
(b) Przypadek dla środowiska OpenMP, komputer Altix.



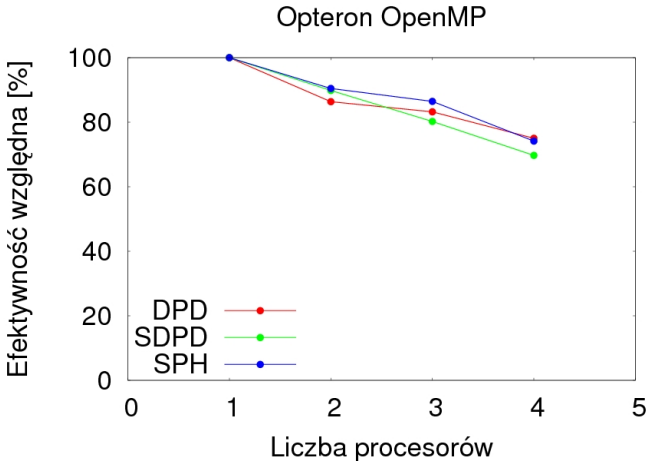
(c) Przypadek dla środowiska MPI, komputer Power4.



(d) Przypadek dla środowiska OpenMP, komputer Power4.



(e) Przypadek dla środowiska MPI, komputer Proliant.



(f) Przypadek dla środowiska OpenMP, komputer Opteron.

Rysunek 5.16: Efektywność względna implementacji równoległej algorytmów metody cząstek w funkcji liczby procesorów dla opisywanych metod uzyskane dla różnych architektur i modeli programowania.

Rysunek (5.17) przedstawia efektywność względną w funkcji liczby procesorów dla każdej metody, a otrzymane dla różnych architektur i obydwu wykorzystywanych środowisk.

Wyniki te pokazują, że dla rozpatrywanych architektur najlepsze wyniki otrzymano przy wykorzystaniu środowiska MPI. Efektywność implementacji równoległej w tym przypadku maleje bardzo wolno ze wzrostem liczby procesorów, osiągając dla 12 procesorów wynik w przybliżeniu równy 0.9. Wyniki otrzymane z wykorzystaniem środowiska OpenMP są znacząco gorsze od wyników otrzymanych przy pomocy środowiska MPI. Dla czterech procesorów efektywność względna waha się w granicach 0.5-0.7. Dodatkowo, można zauważyć pewne zróżnicowanie wyników ze wzgledu na wykorzystywaną architekturę. Wyniki otrzymywane z wykorzystaniem środowiska OpenMP na komputerze SGI Altix 3700 są znacząco gorsze od tych otrzymywanych na innych komputerach. Jest to najprawdopodobniej spowodowane specyfiką tej architektury. W rzeczywistości jest to architektura rozproszona, w której pamięć wspólna jest w rzeczywistości emulowana. Ponieważ pamięć współdzielona na tej architekturze jest jedynie emulowana, a w rzeczywistości model pamięci jest rozproszony, emulacja pamięci dzielonej wymaga implementacji paradygmatu przesyłania komunikatów w tym celu. Dodatkowy poziom wywołań i implementacji skutkuje pogorszeniem osiągów co jest widoczne na rysunku (5.17). Efekt ten widać szczególnie dobrze na rysunku (5.18), na którym umieszczeono wyniki otrzymane z wykorzystaniem środowiska OpenMP dla wszystkich opisywanych metod i ze wszystkich wykorzystywanych komputerów. Wyniki otrzymane na kompuerze Altix są widocznie oddzielone od wyników otrzymanych na innych komputerach.

Z przedstawionych wyników można wywnioskować, że podejście, w którym programista ma bardziej szczegółowy i dokładny wpływ na sposób zrównoleglenia jest wciąż bardziej efektywny, niż podejście w którym zrównoleglanie dokonuje się prawie automatycznie. Rezultaty te potwierdzają dostępne w literaturze [21, 36], a otrzymane w wyniku przeprowadzenia podobnych testów, lecz dla innych modeli oddziaływania.

5.7 Podsumowanie

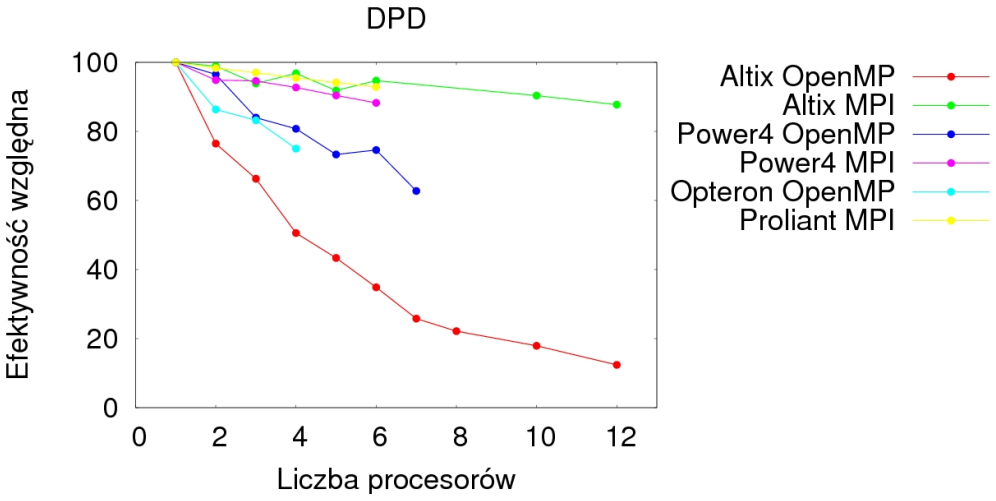
W rozdziale przedstawione zostały aspekty implementacji algorytmu równoległego. Zaprezentowano dwie jego wersje, przeznaczone na architektury z pamięcią współdzieloną i rozproszoną. Na wstępie rozdziału omówiono wskaźniki wykonania równoległego algorytmu służące do oceny jakości jego implementacji. Następnie szczegółowo przedstawiono obydwie implementacje: dla architektury z pamięcią wspólną z wykorzystaniem standardu OpenMP, oraz dla architektury z pamięcią rozproszoną z wykorzystaniem środowiska MPI. Kolejno, omówiono wyniki porównania obydwu implementacji. Wyniki te jednoznacznie wskazują na większe możliwości implementacji dla komputerów z pamięcią rozproszoną. Kolejne podrozdziały przedstawiają optymalizację implementacji. W obu przypadkach stosunkowo niskim nakładem pracy można uzyskać znaczący wzrost efektywności ich wykonania.

Autor do swoich głównych osiągnięć zalicza:

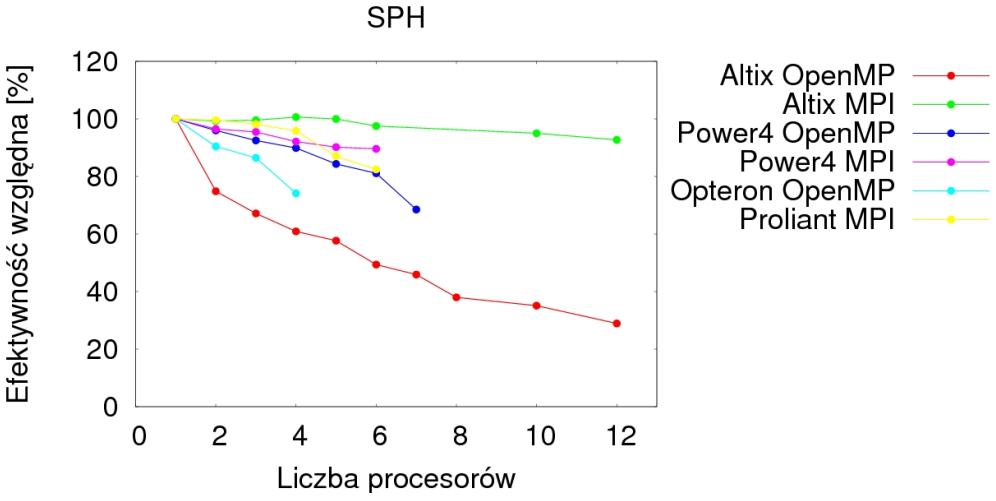
1. propozycję równoległego algorytmu symulacji metodą oddziałujących cząstek dla komputerów z pamięcią współdzieloną wykorzystującą standard OpenMP,

2. propozycję równoległego algorytmu symulacji metodą oddziałujących cząstek dla komputerów z pamięcią rozproszoną. Do jego implementacji można wykorzystać dowolne środowisko przesyłania komunikatów, przykładowo MPI,

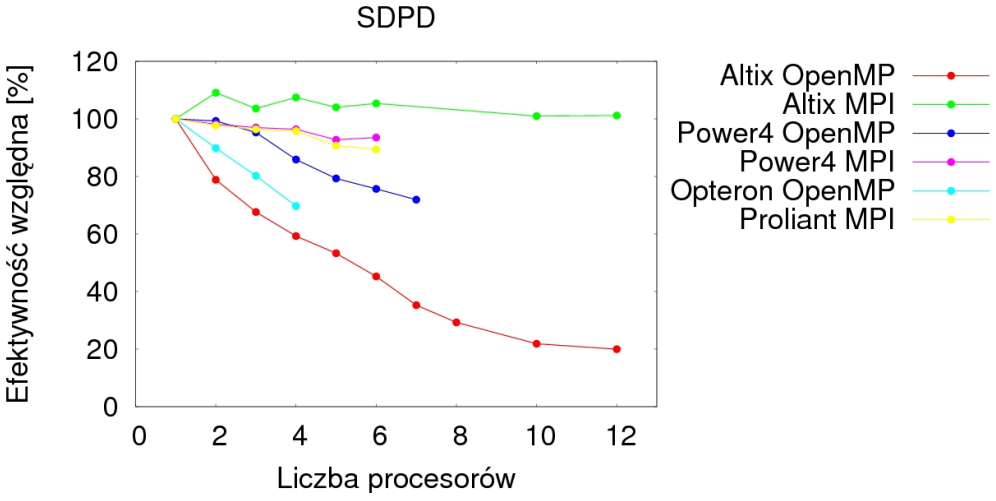
3. zaproponowanie trzech sposobów uwzględnienia losowego charakteru sił w metodach DPD oraz SDPD, ich porównanie oraz wskazanie propozycji najbardziej efektywnej,



(a) Wyniki dla metody DPD.

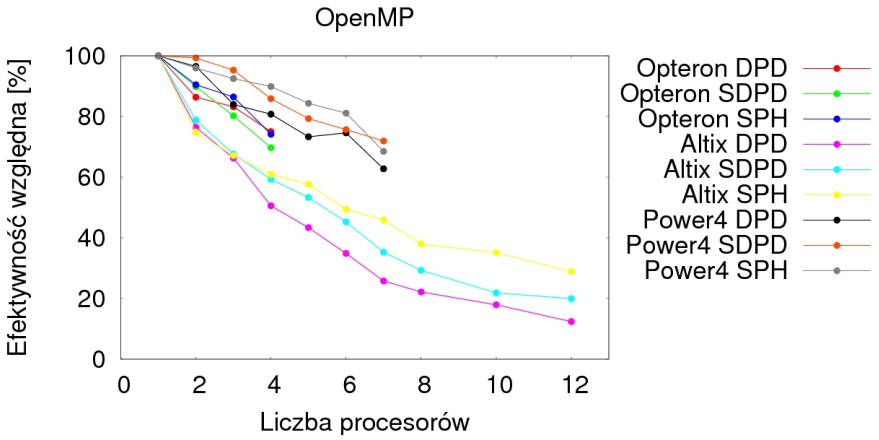


(b) Wyniki dla metody SPH.



(c) Wyniki dla metody SDPD.

Rysunek 5.17: Efektywność względna implementacji równoległej algorytmów metody cząstek w funkcji liczby procesorów dla rozpatrywanych metod.



Rysunek 5.18: Wyniki otrzymane z wykorzystaniem środowiska OpenMP.

4. zaproponowanie oraz implementacja kompletnego algorytmu równoważenia obciążenia dla implementacji pracującej w oparciu o przesyłanie komunikatów,

5. porównanie efektywności równoległych algorytmów symulacji wykorzystujących środowiska OpenMP oraz MPI.

Należy podkreślić, że zaproponowane algorytmy mogą być użyte do symulacji metodami cząstek dla dowolnego oddziaływania krótkozasięgowego. Mają zatem charakter ogólny.

Rozdział 6

Przykładowe wyniki

Zaprezentowane w poprzednich rozdziałach algorytmy symulacji zostały wykorzystane do symulacji przebiegu kilku wybranych zjawisk fizycznych. W rozdziale przedstawione są wyniki tych symulacji. Dotyczą one zjawisk, które dość powszechnie są opisywane w literaturze jako przypadki testowe, bądź też znajdują się w obszarze będącym w szczególnym zainteresowaniu z punktu widzenia nowoczesnej inżynierii.

6.1 Modyfikacja metody SPH do modelowania napięcia powierzchniowego

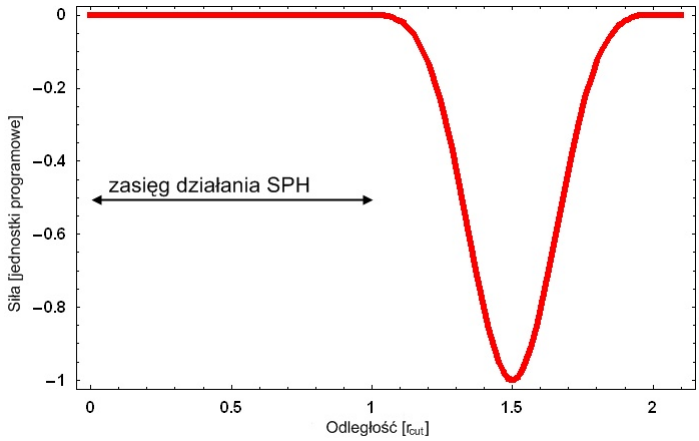
Realizacja zjawisk, w których występuje napięcie powierzchniowe płynu wymaga dodania do sił występujących w modelu dwóch dodatkowych członów. Pierwszy z nich odpowiada za wzajemne przyciąganie cząstek płynu, drugi natomiast modeluje oddziaływanie między cząstkami płynu a cząstkami ścian naczynia.

6.1.1 Oddziaływania płyn-płyn

Napięcie powierzchniowe jest efektem wzajemnego przyciagania się molekuł cieczy. W metodach cząstek operujących na skalach przestrzenno-czasowych większych niż dynamika molekularna nie jest możliwe dokładne modelowanie oddziaływań pomiędzy molekułami płynu. Skala stosowania metod jest znacznie większa, niż skala międzycząsteczkowa. Idea zjawiska jednak pozostaje taka sama, zaś jego modelowanie polega na wprowadzeniu dodatkowych oddziaływań przyciągających pomiędzy cząstkami. W metodzie SPH okazało się to być dość trudne w przypadku, gdy zasięg dodatkowej siły był taki sam jak zasięg oddziaływań SPH. W przypadku tym dodatkowa siła przyciągająca tak modyfikuje oddziaływania SPH, że prowadzi do powstania artefaktów numerycznych (cząstki SPH dążyły do łączenia się w pary). Dlatego, zgodnie z sugestią zawartą w [105], zwiększono zasięg tej siły dwukrotnie w porównaniu z oddziaływaniami SPH. W przypadku takim nie zaobserwowano żadnych niepożądanych efektów, a otrzymane rezultaty są zgodne z oczekiwaniami. Wprowadzona siła jest postaci:

$$F_{ij} = -AW\left(\frac{3}{2}r_{cut} - r_{ij}, \frac{1}{4}r_{cut}\right), \tag{6.1}$$

gdzie A to pewna stała dodatnia, W to funkcja jądra. Wykres zależności wartości siły od odległości między cząstkami został przedstawiony na rysunku (6.1).



Rysunek 6.1: Wykres dodatkowej siły działającej pomiędzy cząstkami, która modeluje napięcie powierzchniowe.

6.1.2 Oddziaływania płyn-ścianki

Aby właściwie zamodelować zjawiska zachodzące w cieczach wykazujących charakter hydrofilowy lub hydrofobowy konieczne jest wprowadzenie dodatkowych oddziaływań przyciagających pomiędzy cząstkami cieczy a cząstkami ścian naczynia. W przedstawionych dotychczas wynikach oddziaływania ścianki-płyn były modelowane jedynie jako odpychające. Wprowadzono więc dodatkowy człon przyciągający. Jego postać jest dana wzorem (6.1), czyli dokładnie takim samym, jak w przypadku oddziaływań płyn-płyn, jednak z inną wartością stałej A.

6.1.3 Wyniki

Przedstawione powyżej modyfikacje metody SPH zastosowano do symulacji dwóch zjawisk.

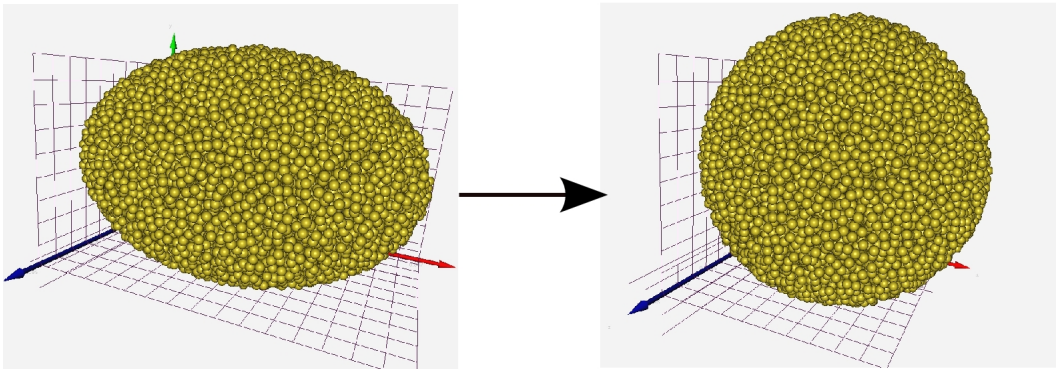
Relaksacja kropli płynu w stanie nieważkości

Pierwszym zjawiskiem, w którym zastosowano zmodyfikowaną metodę SPH jest symulacja oscylacji kulistej kropli płynu w próżni w warunkach nieważkości. Zbudowanie konfiguracji początkowej polegało na utworzeniu idealnie kulistej kropli płynu i przekształcenie jej do elipsoidy zgodnie z wzorem [105]:

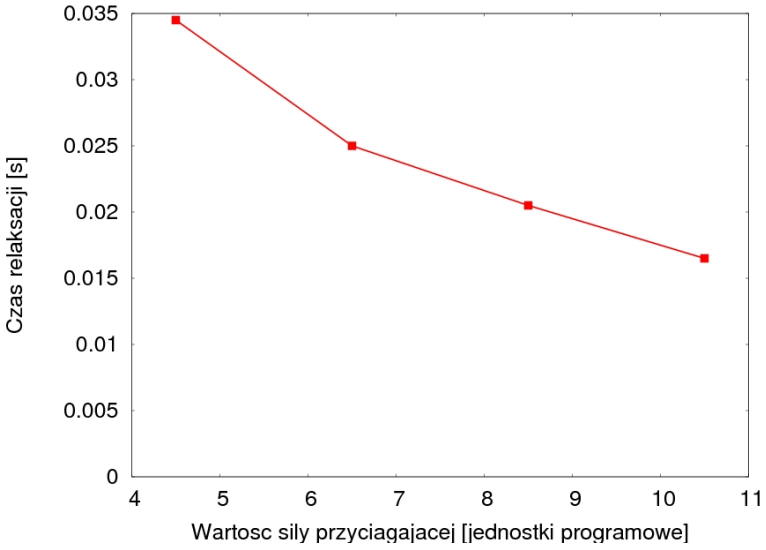
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \sqrt{\frac{2}{\sin\phi}} r \begin{pmatrix} \sin\frac{\phi}{2}\sin u \\ \cos\frac{\phi}{2}\cos u \operatorname{sgn} y \end{pmatrix}, \tag{6.2}$$

gdzie $r = \sqrt{x^2+y^2}$, $\phi = 0.63\pi$ oraz $u = \arctan(x/y)$, a współrzędna z każdej cząstki pozostała bez zmian. Następnie uruchomiono symulację i zmierzono czas relaksacji kropli, który zależy od wartości napięcia powierzchniowego modelowanego płynu [105]. Przy braku lepkości kropla oscylowałaby wokół położenia równowagi w nieskończoność. W symulowanym zjawisku wprowadzona była sztuczna lepkość, która tłumiła oscylacje. Zmierzono przez to czas relaksacji. Przebieg relaksacji przedstawiono na rysunku (6.2). Uruchomione zostały cztery takie symulacje, każda dla innej wartości sił napięcia powierzchniowego.

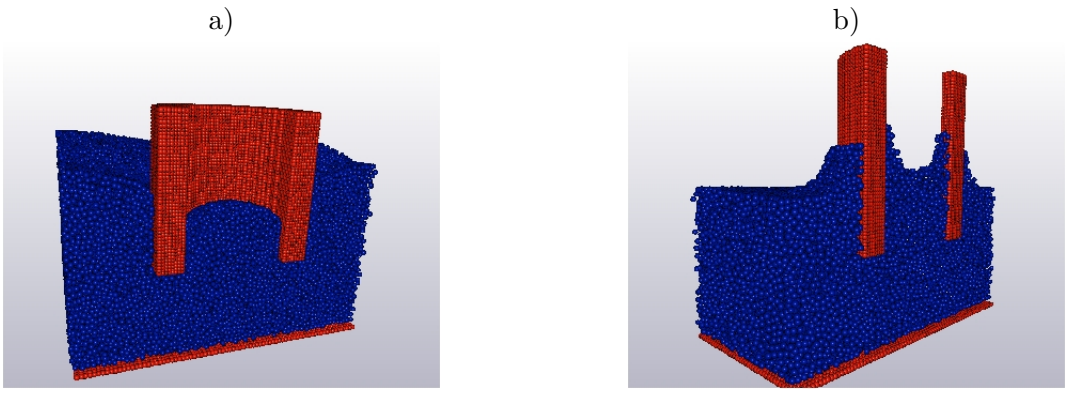
Czas relaksacji zależy od wartości współczynnika napięcia powierzchniowego jak $\sim \gamma^{-1/2}$ [105]. Wykres, jaki otrzymano z symulacji, przedstawiony na rysunku (6.3), pokazuje, że otrzymana zależność jest zgodna z obserwacjami.



Rysunek 6.2: Przebieg relaksacji kropli płynu.



Rysunek 6.3: Zależność czasu relaksacji kropli płynu od wartości siły przyciagającej modelującej napięcie powierzchniowe.



Rysunek 6.4: Otrzymane meniski dla różnych wartości napięcia powierzchniowego: a) menisk wypukły, b) menisk wklęsły.

Powstawanie menisków w kapilarze

Drugim zjawiskiem, w którego symulacji wykorzystano zaimplementowany model napięcia powierzchniowego w metodzie SPH jest powstawanie menisków w kapilarze. W zależności od tego, czy wzajemne przyciąganie cząstek płynu było silniejsze lub słabsze od ich przyciągania przez cząstki ścianek, otrzymano meniski o różnych kształtach. Wyniki jakościowe przedstawiono na rysunku (6.4). Dla przypadku, w którym przyciąganie wzajemne cząstek cieczy było silniejsze od ich przyciągania przez cząstki ścian, otrzymano menisk wypukły, który jest przedstawiony na rysunku (6.4a).

W przypadku, gdy przyciąganie cząstek SPH przez cząstki ścian było silniejsze od ich wzajemnego przyciągania, otrzymano menisk wklęsły, który przedstawiono na rysunku (6.4b). Obydwa rysunki pokazują, że zaproponowana postać sił modelujących napięcie powierzchniowe dobrze symulują to zjawisko.

6.2 Modelowanie przepływu cieczy nie-newtonowskiej

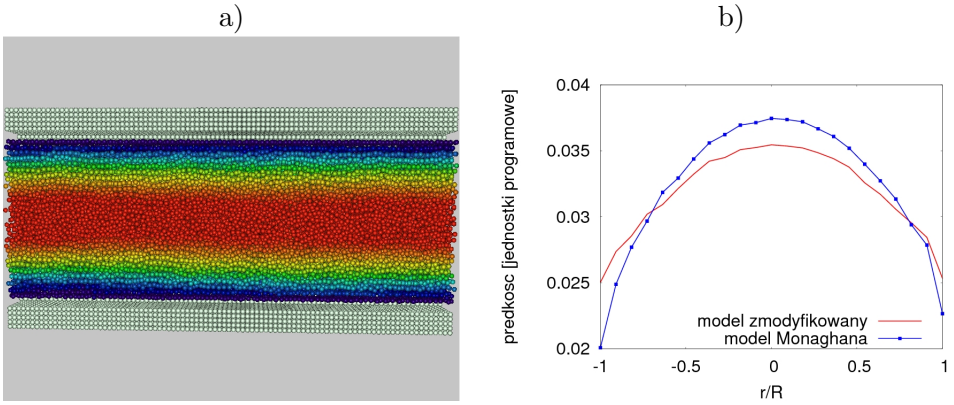
Zdecydowaną większość płynów występujących w przyrodzie stanowią płyny nie-newtonowskie. Stąd konieczność ich łatwego modelowania w różnych skalach przestrzenno-czasowych. Ich podstawową cechą charakterystyczną jest nieliniowy zależność współczynnika lepkości cieczy od lokalnej prędkości ścinania.

Aby zaproponować model zdolny do symulowania przepływów płynów nie-newtonowskich, zaproponowano modyfikację modelu sztucznej lepkości Monaghana [100]. Modyfikacja polega na zamianie równania (3.13) na równanie:

$$\mu_{ij} = \frac{h\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}}{r_{ij}^2 + \eta^2} \cdot \frac{\exp v_{ij} - 1}{v_{ij}}. \tag{6.3}$$

Zgodnie z tym równaniem lepkość działająca pomiędzy dwoma cząstkami zależy nieliniowo od ich prędkości względnej. Dzięki temu możliwe jest otrzymanie nie-newtonowskiego charakteru modelowanego płynu. Objawia się to w symulacji poprzez zmianę profilu prędkości przepływu, który odpowiada współczynnikowi lepkości zależącemu nieliniowo od prędkości ścinania [57].

Zjawiskiem, które zostało zamodelowane w celu sprawdzenia nie-newtonowskiego charakteru symulowanego płynu był przepływ płynu przez naczynie



Rysunek 6.5: Symulacja przepływu płynu w naczyniu cylindrycznym: a) rozkład prędkości, b) profile prędkości dla dwóch różnych modeli lepkości.

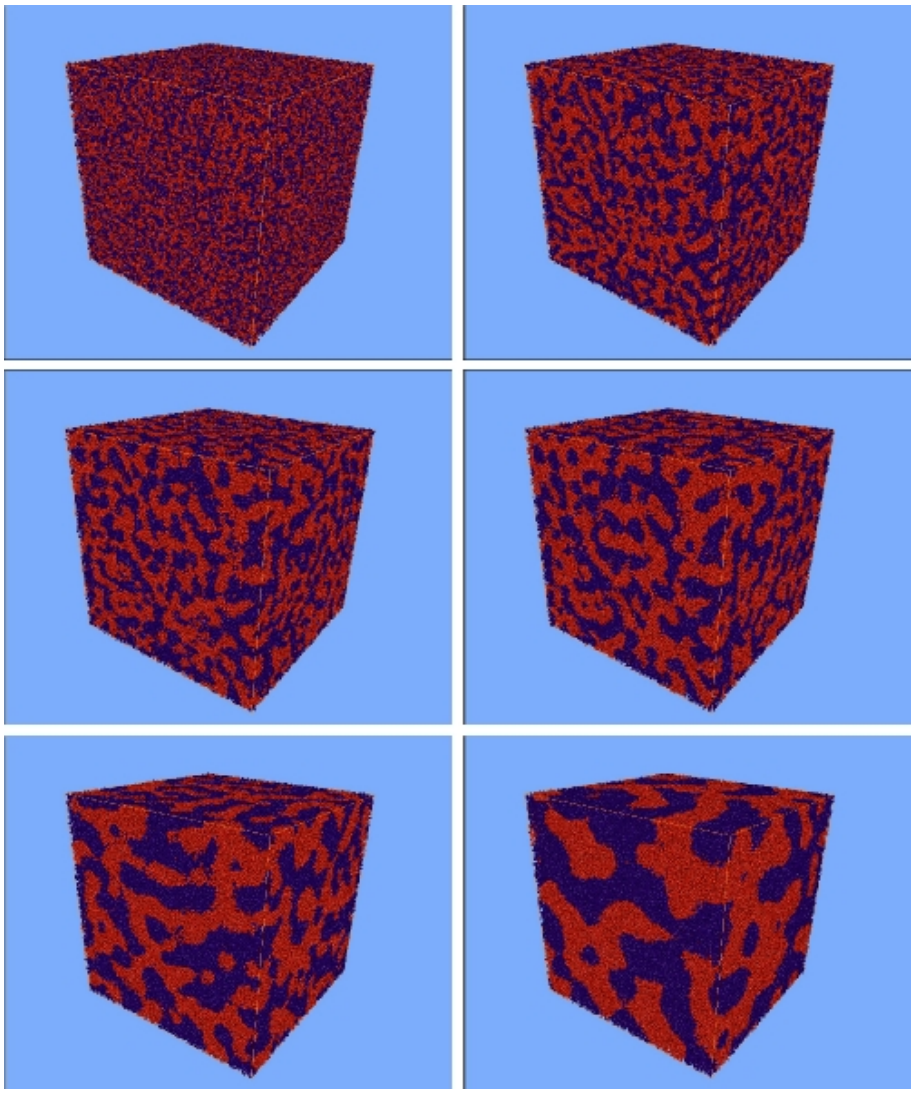
cylindryczne. Początkowo, wszystkie cząstki płynu wypełniające naczynie spoczywały. Następnie, każdej z nich nadano prędkość początkową wzdłuż osi naczynia i przeprowadzono symulację do momentu zatrzymania się cząstek. Przeływ płynu przez naczynie był hamowany przez siły dyssypatywne działające pomiędzy cząstkami ścian i cząstkami płynu. Oddziaływania te były modelowane poprzez model DPD z tą różnicą, że człon brownowski tego modelu został wyłączony. Przykładowy otrzymany rozkład prędkości w naczyniu został przedstawiony na rysunku 6.5.a. Na rysunku tym przy pomocy kolorów oznaczono prędkości cząstek wzdłuż osi naczynia. Kolor czerwony oznacza prędkość największą, a kolor niebieski najmniejszą. Uruchomiono kilka symulacji, każdą dla innego modelu lepkości. Przy porównaniu modelu lepkości Monaghana z modelem zmodyfikowanym według wzoru (6.3) zauważono, że profil prędkości z modelu zmodyfikowanego różni się o profilu z modelu Monaghana zgodnie z oczekiwaniami. Jest on bardziej płaski od profilu otrzymanego z modelu Monaghana, to znaczy prędkość płynu przy ściankach była szybsza, a zdala od nich wolniejsza niż dla płynu z modelem lepkości Monaghana. Potwierdza to nie-newtonowski charakter modelowanego płynu [57]. Profile prędkości dla dwóch modeli: Monaghana i zmodyfikowanego równaniem (6.3) zostały przedstawione na rysunku 6.5.b.

Poprzez wykorzystanie modyfikacji opisanej wzorem (6.3) wprowadzona została nieliniowa zależność współczynnika μ_{ij} od prędkości v_{ij}. Równanie to jest jedynie przykładem tego typu modyfikacji i pokazuje możliwości dalszych badań prowadzonych w tym zakresie. Wykorzystane zostały również inne równania, dla których jednak otrzymano bardzo podobne rezultaty.

6.3 Separacja faz symulowana przy pomocy modelu DPD

Separacja faz jest zjawiskiem, w którym wieloskładnikowy płyn przechodzi od stanu jednorodnego do stanu z wyróżnionymi i oddzielonymi przestrzennie fazami. Teoretyczne podstawy tego zjawiska podane zostały przez Cahna i Hilliarda w pracy [26]. Zjawisko to jest powszechnie spotykane w wielu problemach nauki i techniki. Stąd zrozumienie jego mechanizmów i co za tym idzie, jego poprawna symulacja , mają tak istotne znaczenie.

W modelu DPD zjawisko to zostało zamodelowane poprzez umieszczenie w pudle ubliczeniowym cząstek dwóch rodzajów. Współczynniki składowych sił oddziaływania pomiędzy cząstkami zostały dobrane tak, aby cząstki tego samego rodzaju odpychały się siłą o mniejszej wartości, niż cząstki różnego rodzaju. Dodatkowo, oddziałujące cząstki różnych typów dyssypują energię bardziej niż cząstki tego samego typu. Dzięki temu na granicy



Rysunek 6.6: Kolejne widoki modelowanego układu w symulacji separacji faz metodą DPD.

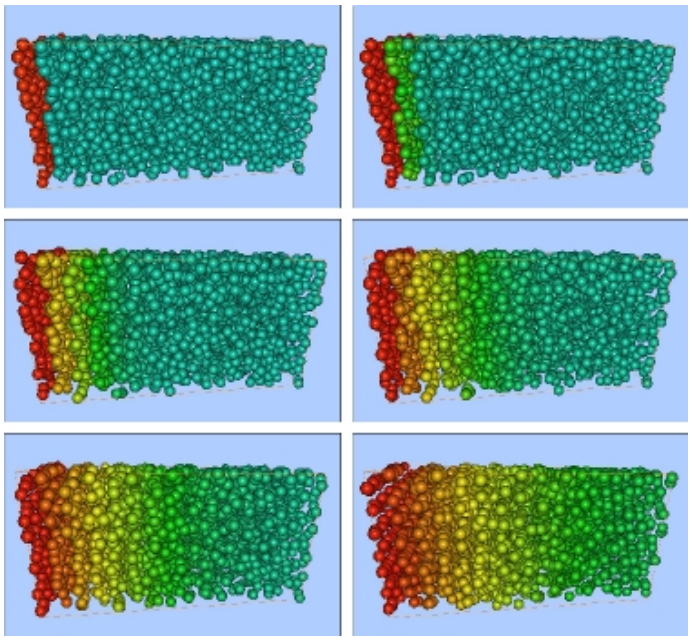
dwóch faz powstaje siła napięcia powierzchniowego i obydwie fazy nie mieszają się ze sobą.

W prezentowanej symulacji liczba cząstek każdego typu była taka sama. Poczatkowy układ symulacji wygenerowano losowo przypisując każdej cząstce jeden z dwóch typów. W trakcie trwania symulacji zaczęły powstawać skupiska cząstek tego samego typu, których rozmiar powiększał się z czasem trwania symulacji. Wyniki te są zgodne z tymi zaprezentowanymi w pracy [42]. Układ osiągnął stan równowagi gdy wartość energii kinetycznej układu jako całości przestała maleć.

Wybrane kroki symulacji zostały przedstawione na rysunku (6.6).

6.4 SDPD - przepływ ciepła

Metoda SDPD umożliwia symulowanie zjawisk analogicznych jak metody DPD oraz SPH. Wynika to stąd, że jest ona uogólnieniem zarówno metody DPD, jak i SPH. Jednak dodatkowo w porównaniu do wymienionych metod, potrafi ona również poprawnie modelować zjawiska fizyczne, w których istotne są termodynamiczne własności symulowanego płynu. W celu ukazania tej możliwości przeprowadzono testową symulację przepływu ciepła w jedno-



Rysunek 6.7: Przepływ ciepła w jednorodnym płynie zamkniętym w prostopadłościennym pudle obliczeniowym.

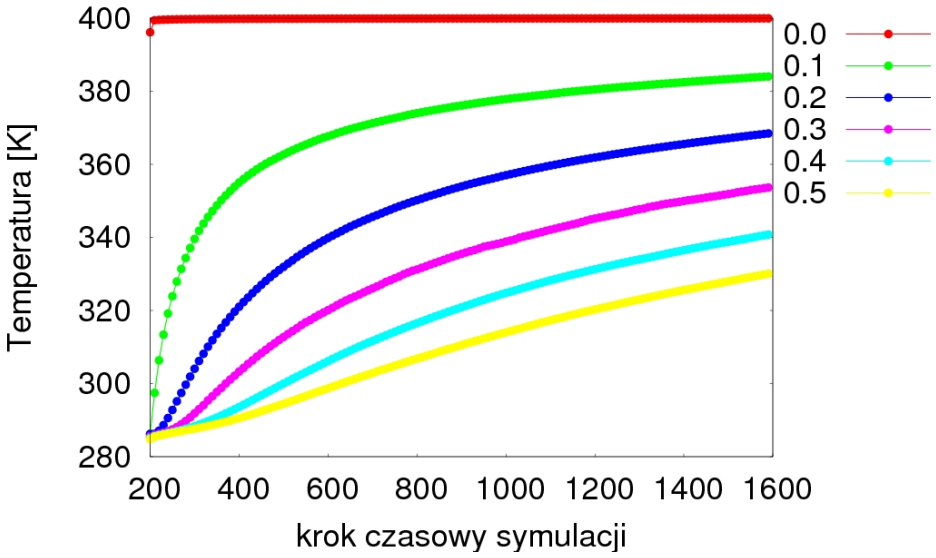
rodnym płynie zamkniętym w prostopadłościennym pudle obliczeniowym. Wybrane kroki symulacji przedstawione zostały na rysunku (6.7), na którym temperaturę poszczególnych cząstek układu oznaczono kolorem.

Aby przetestować fizyczne podstawy zaimplementowanego modelu w czasie trwania symulacji mierzono tempreaturę płynu w 5 punktach oddalonych od źródła ciepła kolejno o 0.0, 0.1, 0.2, 0.3, 0.4 oraz 0.5 długości pudła obliczeniowego. Wyniki przedstawione zostały na rysunku (6.8). Zależność analityczna temperatury od czasu w zadanym położeniu w pudle obliczeniowym może być podana jedynie w formie nieskończonego szeregu [103, 135], dla którego trudno przedstawić wykres w postaci analogicznej jak na rysunku. Wyniki z rysunku można jednak porównać z wynikami otrzymanymi z innych symulacji podobnych zjawisk []. Z porównania tego wynika, że otrzymane i porównywane wyniki są niemal identyczne.

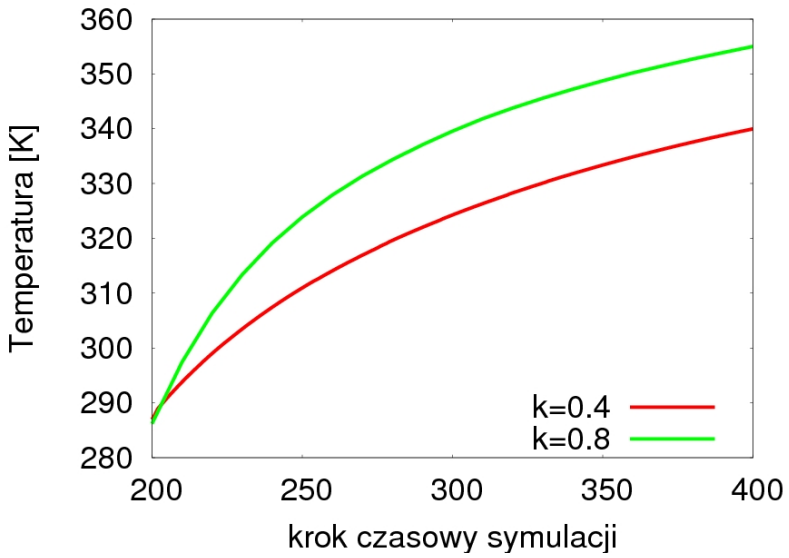
Dodatkowo, przeprowadzono symulacje dla dwóch różnych wartości parametru κ (współczynnik przewodnictwa cieplnego) i ich wyniki przedstawiono na rysunku (6.9). Wyniki te są zgodne z oczekiwaniami - im większa wartość parametru κ, tym szybsza propagacja ciepła w modelowanym płynie. Uzyskane wyniki wskazują, że wpływ parametru κ na symulację jest zgodny z oczekiwaniami.

6.5 Porównanie zaimplementowanych metod cząstek

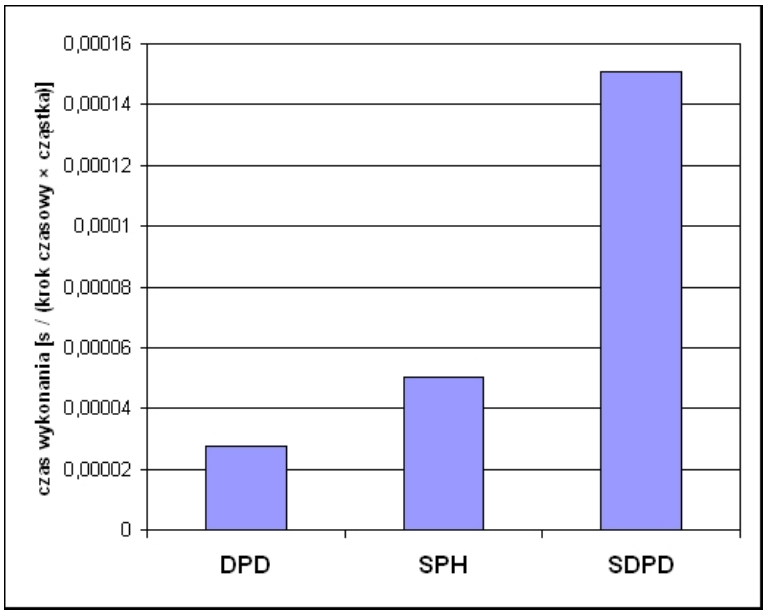
Przedstawione w niniejszej pracy wyniki zostały otrzymane z wykorzystaniem trzech metod cząstek: DPD, SPH, oraz SDPD. Każdą z nich charakteryzuje skala przestrzenno-czasowa, w której metoda może być stosowana. Dodatkowo, każda z wymienionych metod umożliwia modelowanie zjawisk fizycznych o określonej złożoności. Jedynie metoda SDPD jest zdolna do symulacji fizycznych poprawnie modelujących termodynamiczny aspekt zjawiska. Jest to możliwe dzięki ujęciu równań metody w ramach formalizmu GENERIC. Ma to jednak swoje koszty. Na rysunku (6.10) przedstawiono porównanie trzech metod pod względem ich kosztów obliczeniowych wyrażonych w jednostkach: sekunda / (krok obliczeniowy $\times$



Rysunek 6.8: Zależność temperatury od kroku czasowego dla sześciu różnych położeń.



Rysunek 6.9: Zależność temperatury od kroku czasowego dla dwóch różnych wartości parametru κ (jednostki programowe).



Rysunek 6.10: Koszt obliczeniowy opisywanych metod cząstek dla symulacji zachowania płynu jednorodnego.

cząstka). Wyniki te zostały otrzymane z symulacji zachowania się jednorodnego płynu w pudle obliczeniowym z periodycznymi warunkami brzegowymi.

Wyniki te pokazują, że zwiększenie fizycznej poprawności metod cząstek w metodzie SDPD skutkuje zwiększeniem czasu obliczeń. W przypadku porównania metod SDPD oraz DPD czas obliczeń jest dłuższy w przypadku metody SDPD o czynnik ok. 7.

6.6 Podsumowanie

W rozdziale przedstawione zostały wyniki zastosowań metod i implementacji omówionych w poprzednich rozdziałach w modelowaniu płynów rzeczywistych. Przedstawione wyniki dotyczą metod DPD oraz SPH, oraz metody SDPD. Wyniki otrzymane przy pomocy metody SDPD są o tyle wartościowe, że jak dotychczas nie ukazały się w literaturze wyniki otrzymane tą metodą podobne do opisanych.

Do głównych osiągnięć autora zaliczyć należy:

1. zaproponowanie modyfikacji modelu SPH umożliwiającej modelowanie zjawisk, w których uwodacznia się napięcie powierzchniowe symulowanego płynu,

2. zaproponowanie modyfikacji modelu SPH pozwalającej otrzymać cechy przepływu charakterystyczne dla płynu nie-newtonowskiego,

3. wykorzystanie zaproponowanych algorytmów do implementacji symulacji zjawiska separacji faz przy pomocy metody DPD,

4. wykorzystanie implementacji modelu SDPD do symulacji transportu ciepła w płynie jednorodnym

5. określenie zakresów zastosowań metod oraz przedstawienie ich kosztów obliczeniowych.

Rozdział 7

Podsumowanie i wnioski

W rozprawie zaprezentowano wyniki prac dotyczących realizacji algorytmów symulacji metodą cząstek na architekturach z pamięcią współdzieloną oraz rozproszoną. Przedstawiono propozycję algorytmów implementacji równoległej dla architektur komputerowych opartych na modelach z pamięcią rozproszoną i współdzieloną. Zaprezentowano wyniki symulacji zjawisk i czasy wykonania implementacji w zastosowaniu do symulacji płynów metodami DPD, SPH oraz SDPD.

W pracy udało się zrealizować następujące cele:

- zaprezentowano efektywną implementację nowoczesnych, teoretycznych modeli płynów z wykorzystaniem paradygmatów programowania dla architektur z pamięcią współdzieloną i rozproszoną,
- zaproponowano nowe sposoby implementacji równoległej algorytmów symulacji cząstek z wykorzystaniem środowisk MPI oraz OpenMP. Sposoby te kładą specjalny nacisk na efektywne wykorzystanie dostępnych zasobów obliczeniowych,
- przebadano efektywność implementacji przeznaczonych na architektury z pamięcią rozproszoną i współdzieloną. Wnioski wynikające z porównania obydwu implementacji wskazują na istotne istotne różnice pomiędzy nimi, które należy uwzględnić podczas konstruowania i wykonywania symulacji,
- zaproponowano sposób porównania definicji sąsiedztwa pomiędzy cząstkami. Na podstawie wykonanego porównania wysnuto wnioski dotyczące właściwej definicji w symulacjach zarówno płynów ściśliwych, jak i nieściśliwych,
- zaproponowano i przetestowano modyfikację wzorów określających napięcie powierzchniowe w metodzie SPH,
- zaproponowano i sprawdzono nowy model lepkości pozwalający na modelowanie przepływów płynów nie-newtonowskich,
- wykorzystano zaprezentowaną implementację do symulacji złożonych zjawisk z wykorzystaniem metod DPD, SPH oraz SDPD.

7.1 Perspektywy dalszych prac

Podczas przeprowadzania badań opisywanych w niniejszej rozprawie pojawiły się nowe kwestie wskazujące możliwe kierunki dalszych prac. Jako najważniejsze można wymienić:

- dalsze prace nad możliwościami implementacji równoległej metod cząstek z wykorzystaniem nowych środowisk i architektur zarówno komputerów, jak i procesorów,
- przebadanie możliwości implementacji hybrydowej na architekturach wyposażonych w oddzielne jednostki obliczeniowe składające się z procesorów wielordzeniowych,
- wzbogacenie opisywanej implementacji o dodatkowe funkcjonalności, takie jak bardziej złożone schematy całkowania, możliwość modelowania potencjałów dalekozasięgowych czy zdolność do przeprowadzania symulacji przy pomocy metod z dwóch różnych skal przestrzenno czasowych, co wymaga stosowania kilku kroków czasowych jednocześnie,
- przetestowanie metod spójnych termodynamicznie (formalizm GENERIC) do symulowania zjawisk fizycznych nie posiadających do tej pory sprawdzonej metody obliczeniowej,
- zastosowanie metody SDPD do symulacji przebiegu skomplikowanych i złożonych procesów oraz zjawisk, w których konieczne staje się uwzględnienie termodynamicznych aspektów modelowanego układu.

7.2 Wnioski końcowe

Niniejsza praca przedstawia wyniki badań nad możliwością implementacji równoległej metod cząstek do symulowania płynów dla architektur komputerowych z pamięcią rozproszoną i współdzieloną. Wydaje się, że przedstawione w pracy implementacje wyczerpały możliwości optymalizacji czasu wykonania symulacji. Dotyczy to zarówno wersji na architektury z pamięcią rozproszoną (środowisko MPI) jak i z pamięcią współdzieloną (środowisko OpenMP). Wyniki te prezentują możliwości imeplementacji równoległej i świadczą o konieczności korzystania z architektur wieloprocesorowych w symulacji płynów nowoczesnymi modelami opartymi na metodach cząstek.

Wydaje się, że w najbliższym czasie spójne termodynamicznie metody cząstek osiągną swoją dojrzałość, jak również powstające architektury komputerowe zdolne będą do przeprowadzania symulacji za pomocą tych metod dla układów składających się z dużej, niedostępnej dzisiaj liczby cząstek. Wymaga to stosowania sprawdzonych technik implementacji równoległej w celu optymalnego wykorzystania zasobów obliczeniowych przyszłych architektur. Niniejsza praca prezentuje istotne aspekty tych technik i zawiera wartościowe wnioski, które mogą być wykorzystane przy tworzeniu przyszłych implementacji równoległych symulacji metodami cząstek.

Dodatek A

Charakterystyka wykorzystywanego sprzętu

Przedstawione w pracy wyniki symulacji oraz pomiary dla niej wskaźników wykonania równoległego przeprowadzono na szeregu komputerów różniących się szczegółami architektury. W poniższym rozdziale znajduje się charakterystyka tych architektur wraz z wykorzystywanym na nich oprogramowaniem.

A.1 HP ProLiant DL585

Maszyna ta charakteryzuje się:

- 4 procesorami AMD Opteron^{TM}Model 865 1.8 GHz - 1 MB L2 dual core,
- 20 GB pamięci operacyjnej DDR taktowanej zegarem o częstotliwości 400 MHz.

Jej dokładna charakterystyka jest przedstawiona w [75]. Zgodnie z informacjami podawanymi przez producenta maszyna ta jest przeznaczona m.in. do wykonywania obliczeń wielkiej skali. W momencie udostępnienia możliwości wykorzystania w tej architekturze czterech procesorów Opteron dual core maszyna ta była jedną z najbardziej wydajnych w swoim przedziale cenowym.

W niniejszej pracy na komputerze tym uruchamiane były symulacje testujace wskaźniki wykonania równoległego przy wykorzystaniu środowiska MPI. W tym celu wykorzystano implementację środowiska LAM/MPI [23] w wersji 7.0.6 wraz z dostarczonym wraz z nim kompilatorem mpic++.

A.2 AMD Opteron 270 Dual Core

Komputer ten charakteryzuje się:

- 4 procesorami AMP Opteron^{TM}Model 270 2.0 GHz - 1 MB L2 dual core,
- 12 GB pamięci operacyjnej DDR taktowanej zegarem.

W niniejszej pracy na komputerze tym uruchamiane były symulacje testujące wskaźniki wykonania równoległego przy wykorzystaniu środowiska OpenMP. W tym celu wykorzystano kompilator PGI [114] w wersji 7.1-3.

A.3 SGI Altix 3700

SGI Altix 3700 jest serwerem obliczeniowym opartym na 64-bitowych procesorach Intel Itanium 2. System ten cechuje się dużą skalowalnością - może zawierać do 512 procesorów, pozwala na uruchamianie programów równoległych wykorzystujacych zarówno paradygmat oparty na przesyłaniu komunikatów jak i pamięci współdzielonej. Możliwe jest zaadresowanie do 24 TB pamięci operacyjnej. System SGI Altix 3700 jest w rzeczywistości klastrem składającym się z oddzielnych jednostek, z których każda zawiera dwie pary procesorów (tzw. C-Bricks). Każda para ma przypisaną własnę pamięć operacyjną. Poszczególne części są połączone ze sobą siecią NUMAlink^{TM}realizującą topologię „fat-tree". Mechanizm ten zapewnia dostęp do dowolnej pamięci w całym systemie dla dowolnego procesora. Dzięki temu, pomimo rozproszonego charakteru architektury, mechanizm ten realizuje interfejs pamięci współdzielonej. Ten model architektury nazywany jest ccNUMA.

Do obliczeń w niniejszej pracy wykorzystano klaster 10 maszyn SGI Altix 3700, których zasoby obliczeniowe udostepniane były przez wspólny system kolejkowy. Na klaster ten składają się:

- 1 komputer z 256 procesorami Intel Itanium 2 1.6 GHz i z pamięcią o rozmiarze 512 GB,
- 1 komputer z 48 procesorami Intel Itanium 2 1.3 GHz i z pamięcią o rozmiarze 96 GB,
- 8 komputerów z 16 procesorami Intel Itanium 2 1.5 GHz i z pamięcią o rozmiarze 32 GB.

W niniejszej pracy na komputerze tym uruchamiane były symulacje testujące wskaźniki wykonania równoległego przy wykorzystaniu środowisk zarówno OpenMP jak i MPI. W tym celu korzystano z kompilatora i bibliotek dostarczonych przez firmę Intel [79].

A.4 IBM Power4

IBM Power4 jest klastrem składającym się z 23 jednostek obliczeniowych, z których każda jest wieloprocesorowym komputerem opartym na architekturze z pamięcią współdzieloną. Wszystie procesory w klastrze są z rodziny procesorów IBM Power4. Wszystkie 23 jednostki obliczeniowe sa dostępne przez wspólny system kolejkowy. Na dostępne zasoby składają się:

- 1 komputer z 32 procesorami p690 1.3 GHz i z pamięcią o rozmiarze 64 GB,
- 1 komputer z 24 procesorami p690 1.3 GHz i z pamięcią o rozmiarze 24 GB,
- 1 komputer z 30 procesorami p690 1.7 GHz i z pamięcią o rozmiarze 128 GB,
- 1 komputer z 32 procesorami p690 1.7 GHz i z pamięcią o rozmiarze 64 GB,
- 8 komputerów z 8 procesorami p655 1.5 GHz i z pamięcią o rozmiarze 16 GB,
- 9 komputerów z 8 procesorami p655 1.7 GHz i z pamięcią o rozmiarze 16 GB.

W niniejszej pracy na kompuerze tym uruchamiane były symulacje testujące wskaźniki wykonania równoległego przy wykorzystaniu środowisk zarówno OpenMP jak i MPI. W tym celu korzystano z kompilatora firmy IBM [71].

Bibliografia

[1] J. Adamowski, *Metody Obliczeniowe Fizyki*, WFiIS, Akademia Górniczo-Hutnicza, dostępne na stronie internetowej `http://www.zftik.agh.edu.pl/mof/wyk/index.html`, 2008.

[2] B. J. Alder and T. E. Wainwright, *Studies in Molecular Dynamics. I. General Method*, Journal of Chemical Physics **31** (1959), 459–466.

[3] J. Ambjørn, J. Jurkiewicz, and R. Loll, *Reconstructing the Universe*, Physical Review D **72** (2005), no. 6, 64014.

[4] G. Amdahl, *Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities*, AFIPS Conference Proceedings, no. 30, 1967, pp. 483–485.

[5] Y. Aoyama and J. Nakano, *RS/6000 SP: Practical MPI Programming*, International Technical Support Organization, 1999.

[6] M.A. Arbib (ed.), *The Handbook of Brain Theory and Neural Networks*, Bradford Books, 2003.

[7] P. Bak, M. Paczuski, and M. Shubik, *Price variations in a stock market with many agents*, Physica A Statistical Mechanics and its Applications **246** (1997), 430–453.

[8] E. Barth, M. Mandziuk, and T. Schlick, *A separating framework for increasing the timestep in molecular dynamics*, Computer Simulation of Biomolecular Systems: Theoretical and Experimental Applications **3** (1997), 97–121.

[9] G. K. Batchelor, *An introduction to fluid dynamics*, Cambridge University Press, 1967.

[10] M. R. Bate, I. A. Bonnell, and V. Bromm, *The formation of a star cluster: predicting the properties of stars and brown dwarfs*, Monthly Notices of the Royal Astronomical Society **339** (2003), 577–599.

[11] M. R. Bate, I. A. Bonnell, and N. M. Price, *Modelling accretion in protobinary systems*, Monthly Notices of the Royal Astronomical Society **277** (1995), 362–376.

[12] J. Büchner, C. T. Dum, and M. Scholer, *Space Plasma Simulation*, Springer, 2003.

[13] W. Benz, *Smooth Particle Hydrodynamics - a Review*, Numerical Modelling of Nonlinear Stellar Pulsations Problems and Prospects (J. R. Buchler, ed.), 1990, pp. 269–+.

[14] W. Benz, W. L. Slattery, and A. G. W. Cameron, *Short note: Snapshots from a three-dimensional modeling of a giant impact*, Origin of the Moon (W. K. Hartmann, R. J. Phillips, and G. J. Taylor, eds.), 1986, pp. 617–+.

[15] H. J. C. Berendsen, *Biophysical applications of molecular dynamics*, Computer Physics Communications **44** (1987), 233–242.

[16] E. S. Boek, P. V. Coveney, H. N. W. Lekkerkerker, and P. van der Schoot, *Simulating the rheology of dense colloidal suspensions using dissipative particle dynamics*, Physical Review E **55** (1997), 3124–3133.

[17] B. M. Boghosian and W. Taylor, *Renormalized equilibria of a Schlögl model lattice gas*, Journal of Statistical Physics **81** (1995), 295–317.

[18] K. Boryczko, *Informatyczny model przepływu krwi w naczyniach włosowatych*, AGH - Uczelniane wydawnictwa naukowo-dydaktyczne, 2002.

[19] K. Boryczko, Marian Bubak, Jacek Kitowski, Jacek Moscinski, and Renata Slota, *Lattice Gas Automata and Molecular Dynamics on a Network of Computers*, HPCN Europe 1994: Proceedings of the nternational Conference and Exhibition on High-Performance Computing and Networking Volume I (London, UK), Springer-Verlag, 1994, pp. 177–180.

[20] K. Boryczko, W. Dzwinel, and D. A. Yuen, *Dynamical clustering of red blood cells in capillary vessels*, Journal of Molecular Modeling **9** (2003), no. 1, 16–33.

[21] K. Boryczko, W. Dzwinel, and D. Yuen, *Parallel Implementation of the Fluid Particle Model for Simulating Complex Fluids in the Mesoscale*, Concurency and Computation: practice and experience **14** (2002), 137–161.

[22] G. Burns, *The Local Area Multicomputer*, Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications, ACM Press, Marzec 1989.

[23] Greg Burns, Raja Daoud, and James Vaigl, *LAM: An Open Cluster Environment for MPI*, Proceedings of Supercomputing Symposium, 1994, pp. 379–386.

[24] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*, John Wiley and Sons, 2003.

[25] R. Butler and E. Lust, *Monitors, messages, and clusters: the p4 parallel programming system*, Journal of Parallel Computing **20** (1994), no. 4, 547–564.

[26] J.W. Cahn and J.E. Hilliard, *Free Energy of a Nonuniform System. I. Interfacial Free Energy*, The Journal of Chemical Physics **28** (2004), 258.

[27] R. Capuzzo-Dolcetta and R. Di Lisio, *A Criterion for the choice of the interpolation kernel in Smoothed Particle Hydrodynamics*, Arxiv preprint astro-ph/9907089 (1999).

[28] G. Carraro, C. Lia, and C. Chiosi, *Galaxy formation and evolution - I. The Padua tree-sph code (pd-sph)*, Monthly Notices of the Royal Astronomical Society **297** (1998), 1021–1040.

[29] H. B. Casimir, *On Onsager's Principle of Microscopic Reversibility*, Reviews of Modern Physics **17** (1945), 343–350.

[30] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*, The MIT Press, 2007.

[31] G. Ciccotti, D. Frenkel, and I. R. McDonald, *Simulation of liquids and solids*, North-Holland, 1987.

[32] G. Ciccotti and W. G. Hoover, *Molecular dynamics simulation of statistical-mechanical systems*, North-Holland, 1986.

[33] P. Cleary, J. Ha, V. Alguine, and T. Nguyen, *Flow modelling in casting processes*, Applied Mathematical Modelling **26** (2002), no. 2, 171–190.

[34] A. Colagrossi and M. Landrini, *Numerical simulation of interfacial flows by smoothed particle hydrodynamics*, Journal of Computational Physics **191** (2003), 448–475.

[35] G.H. Cottet and P.D. Koumoutsakos, *Vortex Methods: Theory and Practice*, Cambridge University Press, 2000.

[36] R. Couturier and C. Chipot, *Parallel molecular dynamics using on a shared memory machine*, Computer Physics Communications **124** (2000), 49–59.

[37] P. V. Coveney and K. E. Novik, *Computer simulations of domain growth and phase separation in two-dimensional binary immiscible fluids using dissipative particle dynamics*, Computer (1996), 7002–+.

[38] J. M. A. Danby, R. Kouzes, and C. Whitney, *Astrophysics Simulations*, John Wiley & Sons, 1995.

[39] M. S. Daw and M. I. Baskes, *Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals*, Physical Review B **29** (1984), 6443–6453.

[40] A. Drogoul and J. Ferber, *Multi-Agent Simulation as a Tool for Modeling Societies: Application to Social Differentiation in Ant Colonies*, Actes du Workshop MAAMAW **92** (1992).

[41] W. Dzwinel and D. Yuen, *A Two-Level, Discrete Particle Approach for Large-Scale Simulation of Colloidal Aggregates*, International Journal of Modern Physics C **11** (2000), no. 5, 1037–1061.

[42] W. Dzwinel and D. A. Yuen, *Matching Macroscopic Properties of Binary Fluids to the Interactions of Dissipative Particle Dynamics*, International Journal of Modern Physics C **11** (2000), 1–25.

[43] R. Eckhardt, *Stan Ulam, John von Neumann, and the Monte Carlo method*, Los Alamos Science **15** (1987), 131–137.

[44] R. Eigenmann and B. R. de Supinski (eds.), *OpenMP in a New Era of Parallelism, 4th International Workshop, IWOMP*, Springer Berlin / Heidelberg, May 2008.

[45] M. Ellero, M. Serrano, and P. Español, *Incompressible smoothed particle hydrodynamics*, Journal of Computational Physics **226** (2007), 1731–1752.

[46] P. Español, *Fluid particle model*, Physical Review E **57** (1998), 2930–2948.

[47] P. Español and P. Warren, *Statistical Mechanics of Dissipative Particle Dynamics*, Europhys. Lett. **30** (1995), no. 4, 191–196.

[48] Pep Español and Mariano Revenga, *Smoothed dissipative particle dynamics*, Phys. Rev. E **67** (2003), no. 2, 026705.

[49] P. Espanol, *Dissipative Particle Dynamics and Other Fluid Particle Models*, ICASE LARC Interdisciplinary Series in Science and Engineering **10** (2004), 213–236.

[50] C. Farrel, *Simulating ultracold matter: horizons and slow light*, Ph.D. thesis, University of St Andrews, December 2007.

[51] G.S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, Springer, 1996.

[52] M. Flynn, *Some Computer Organizations and Their Effectiveness*, IEEE Trans. Comput. **C-21** (1972), 948.

[53] M. B. Liu G. R Liu, *Smoothed Particle Hydrodynamics - a meshfree particle method*, ch. 4.4.4-4.4.5, pp. 129–132, World Scientific Publishing Company, 2003.

[54] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, 1994, dostępne na stronie `http://www.netlib.org/pvm3/book/pvm-book.html`.

[55] G. Geist, M. Heath, B. Peyton, and P. Worley, *A Users' Guide to PICL. A Portable Instrumented Communication Library*, Tech. Report TM-11616, Oak Ridge National Laboratory, October 1990.

[56] J. B. Gibson, K. Chen, and S. Chynoweth, *The Equilibrium of a Velocity-Verlet Type Algorithm for Dpd with Finite Time Steps*, International Journal of Modern Physics C **10** (1999), 241–261.

[57] FJH Gijsen, FN van de Vosse, and JD Janssen, *The influence of the non-Newtonian properties of blood on the flow in large arteries: steady flow in a carotid bifurcation model*, Journal of Biomechanics **32** (1999), no. 6, 601–608.

[58] R. A. Gingold and J. J. Monaghan, *Smoothed particle hydrodynamics - Theory and application to non-spherical stars*, Monthly Notices of the Royal Astronomical Society **181** (1977), 375–389.

[59] H. Grabert and M. S. Green, *Fluctuations and nonlinear irreversible processes*, Physical Review A **19** (1979), 1747–1756.

[60] R. Graham, *Noise in Nonlinear Dynamical Systems, Vol. 1, Theory of Continous Fokker-Planck Systems*, p. 225, Cambridge University Press, Cambridge, 1989.

[61] Richard L. Graham, Galen M. Shipman, Brian W. Barrett, Ralph H. Castain, George Bosilca, and Andrew Lumsdaine, *Open MPI: A High-Performance, Heterogeneous MPI*, Proceedings, Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (Barcelona, Spain), September 2006.

[62] A. Grama, A. Gupta, V. Kumar, and G. Karypis, *Introduction to Parallel Computing Design and Analysis of Algorithms*, Pearson Education, 2003.

[63] Y.N. Grigoryev, V. a Vshivkov, and MP Fedoruk, *Numerical "Particle-In-Cell"Methods - Theory and Applications*, Brill Academic Pub, 2002.

[64] M. Grmela and H. C. Öttinger, *Dynamics and thermodynamics of complex fluids. I. Development of a general formalism*, Physical Review E **56** (1997), 6620–6632.

[65] R. D. Groot and P. B. Warren, *Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation*, Journal of Chemical Physics **107** (1997), 4423–4435.

[66] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, Parallel Computing **22** (1996), no. 6, 789–828.

[67] J. Grotendorst, D. Marx, and A. Muramatsu (eds.), *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, Lecture Notes*, John von Neumann Institute for Computing, Jülich, 2002.

[68] J. Grotendorst, D. Marx, and A. Muramatsu (eds.), *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, Lecture Notes*, ch. Classical Molecular Dynamics, pp. 211–254, John von Neumann Institute for Computing, Jülich, 2002.

[69] J. Gustafson, *Reevaluating Amdahl's Law*, Communications of the ACM **31** (1988), no. 5, 532–533.

[70] Lars Hernquist and Neal Katz, *TREESPH: A unification of SPH with the hierarchical tree method*, The Astrophysical Journal Supplement Series **70** (1989), 419–446.

[71] S. Hikida, D. Paulmard, and M. Wong, *A brief introduction to IBM XL compilers*, IBM, 2005, dostępne na stronie internetowej `http://www-128.ibm.com/developerworks/library/pa-nldec04-xlcompilers/`.

[72] R. W. Hockney and J. W. Eastwood, *Computer Simulations Using Particles*, McGraw-Hill Inc., 1981.

[73] P. J. Hoogerbrugge and J.M.V.A. Koelman, *Simulating macroscopic hydrodynamic phenomena with dissipative partice dynamics*, Europhys. Lett. **19** (1992), no. 3, 155–160.

[74] W. G. Hoover, *Nonequilibrium molecular dynamics*, Nuclear Physics A **545** (1992), 523–536.

[75] *HP ProLiant DL585 server - product overview*, dostępne na stronie internetowej `http://h18004.www1.hp.com/products/servers/proliantdl585/index.html`.

[76] X. Y. Hu and N. A. Adams, *A multi-phase SPH method for macroscopic and mesoscopic flows*, Journal of Computational Physics **213** (2006), 844–861.

[77] YC Hu, H. Lu, AL Cox, and W. Zwaenepoel, *OpenMP for Networks of SMPs*, Journal of Parallel and Distributed Computing **60** (2000), no. 12, 1512–1530.

[78] P. J. in't Veld, S. J. Plimpton, and G. S. Grest, *Accurate and efficient methods for modeling colloidal mixtures in an explicit solvent using molecular dynamics*, Comp. Phys. Comm. **179** (1008), 320–329.

[79] Intel®Corporation, *Intel® C++ Compiler User and Reference Guide*, numer dokumentu: 304968-022US, dostępne na stronie internetowej `http://www.intel.com`.

[80] M. C. Thompson J. J. Monaghan and K. Hourigan, *Simulation of free surface flows with SPH*, ASME Symposium on Computational Methods in Fluid Dynamics, Lake Tahoe, vol. June 19-23, 1994.

[81] R. K. Kalia, T. J. Campbell, A. Chatterjee, A. Nakano, P. Vashishta, and S. Ogata, *Multiresolution algorithms for massively parallel molecular dynamics simulations of nanostructured materials*, Computer Physics Communications **128** (2000), 245–259.

[82] V. K. Kedrinskii and S. Y. Knyazeva, *Hydrodynamics od Explosion: Experiment and Models*, Springer, 2005.

[83] R. Keunings, *Micro–macro methods for the multiscale simulation of viscoelastic flow using molecular models of kinetic theory*, Rheology Reviews (2004), 67–98.

[84] G. Krawezik, *Performance comparison of MPI and three openMP programming styles on shared memory multiprocessors*, Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures (2003), 118–127.

[85] P. Lague, R.W. Pastor, and B.R. Brooks, *Pressure-Based Long-Range Correction for Lennard-Jones Interactions in Molecular Dynamics Simulations: Application to Alkanes and Interfaces*, Journal of Physical Chemistry B **108** (2004), no. 1, 363–368.

[86] S. Litvinov, M. Ellero, X. Hu, and N. A. Adams, *Smoothed dissipative particle dynamics model for polymer molecules in suspension*, Phys. Rev. E **77** (2008), 066703.

[87] G. R. Liu and M. B. Liu, *Smoothed Particle Hydrodynamics - a meshfree particle method*, World Scientific Publishing Company, 2003.

[88] J. Liu, X. Jin, and K.C. Tsui, *Autonomy Oriented Computing: From Problem Solving To Complex Systems Modeling*, Kluwer Academic Publishers, 2005.

[89] M. B. Liu, G. R. Liu, K. Y. Lam, and Z. Zong, *Smoothed particle hydrodynamics for numerical simulation of underwater explosion*, Computational Mechanics **30** (2003), 106–118.

[90] L. B. Lucy, *A numerical approach to the testing of the fission hypothesis*, Astronomical Journal **82** (1977), 1013–1024.

[91] P. W. Cleary M. L. Sawley and J. Ha, *Modelling of flow in porous media and resin transfer moulding using Smoothed Particle Hydrodynamics*, Second International Conference on CFD in the Minerals and Process Industries, CSIRO, Melbourne, Australia, vol. December 6-8, 1999.

[92] E. Muller M. Steinmetz, *On the capabilities and limits of the smoothed particle hydrodynamics*, Astron. Astrophys. **268** (1993), 391–410.

[93] S. Marri and S. D. M. White, *Smoothed particle hydrodynamics for galaxy-formation simulations: improved treatments of multiphase gas, of star formation and of supernovae feedback*, Monthly Notices of Royal Astronomical Society **345** (2003), 561–574.

[94] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Czerwiec 2008, dostępne na stronie `http://www.mpi-forum.org/docs/`.

[95] N. Metropolis, *The Beginning of the Monte Carlo method*, Los Alamos Science **15** (1987), 125–130.

[96] J. J. Monaghan, *Particle methods for hydrodynamics*, Comput. Phys. Rep. **3** (1985), 71–124.

[97] J. J. Monaghan, *Smoothed Particle Hydrodynamics*, Annu. Rev. Astron. Astrophys. **30** (1992), 543–74.

[98] J. J. Monaghan, *Simulating Free Surface Flows with SPH*, Journal of Computational Physics **110** (1994), 399–406.

[99] J. J. Monaghan, *Smoothed particle hydrodynamics*, Rep. Prog. Phys. **68** (2005), 1703–1759.

[100] J. J. Monaghan and R. A. Gingold, *Shock Simulation by the Particle Method SPH*, Journal of Computational Physics **52** (1983), 374–+.

[101] J. J. Monaghan, A Kos, and N. Issa, *Fluid Motion Generated by Impact*, Journal of Waterway, Port, Coastal and Ocean Engineering **129** (2003), 250–259.

[102] nCUBE Corporation, *nCUBE 2 Programmers Guide, r2.0*, Grudzień 1990.

[103] N. Noda, R. Hetnarski, and Y. Tanigawa, *Thermal Stresses*, Taylor & Francis, 2003.

[104] K. E. Novik and P. V. Coveney, *Using Dissipative Particle Dynamics to Model Binary Immiscible Fluids*, International Journal of Modern Physics C **8** (1997), 909–918.

[105] S. Nugent and HA Posch, *Liquid drops and surface tension with smoothed particle applied mechanics*, Physical Review E **62** (2000), no. 4, 4968–4975.

[106] Lars Onsager, *Reciprocal Relations in Irreversible Processes. I.*, Phys. Rev. **37** (1931), no. 4, 405–426.

[107] ———, *Reciprocal Relations in Irreversible Processes. II.*, Phys. Rev. **38** (1931), no. 12, 2265–2279.

[108] The OpenMP Architecture Review Board, *OpenMP Application Program Interface*, 3.0 ed., May 2008, dostępne na stronie `http://openmp.org/wp/openmp-specifications/`.

[109] *Strona główna konsorcjum 'The OpenMP Architecture Review Board'*, URL: `http://openmp.org`.

[110] E. S. Oran, C. K. Oh, and B. Z. Cybyk, *Direct Simulation Monte Carlo: Recent Advances and Applications*, Annual Review of Fluid Mechanics **30** (1998), 403–441.

[111] S. Orsino, R. Weber, and U. Bollettini, *Numerical Simulation of Combustion of Natural Gas with High-Temperature Air*, Combustion Science and Technology **170** (2001), no. 1, 1–34.

[112] H. C. Öttinger and M. Grmela, *Dynamics and thermodynamics of complex fluids. II. Illustrations of a general formalism*, Physical Review E **56** (1997), 6633–6655.

[113] Stephen Oxley, *Modelling the Capture Theory for the Origin of Planetary Systems*, Ph.D. thesis, University of York, December 1999.

[114] The Portland Group®, *PGI® User's Guide, Parallel Fortran, C and C++ for Scientists and Engineers*, numer dokumentu: 083161052, dostępne na stronie internetowej `http://www.pgroup.com/doc/`.

[115] D. Price, *Smoothed Particle Hydrodynamics*, Arxiv preprint astro-ph/0507472 (2005).

[116] T. Rabczuk, T. Belytschko, and SP Xiao, *Stable particle methods based on Lagrangian kernels*, Computer Methods in Applied Mechanics and Engineering **193** (2004), no. 12-14, 1035–1063.

[117] TJ Raeker and AE DePristo, *Theory of Chemical Bonding Based on the Atom-Homogeneous Electron Gas System*, International Reviews in Physical Chemistry **10** (1991), no. 1, 1–54.

[118] D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, The Art of Molecular Dynamics Simulation, by D. C. Rapaport, pp. 564. ISBN 0521825687. Cambridge, UK: Cambridge University Press, April 2004., April 2004.

[119] P. J. Rossky, J. D. Doll, and H. L. Friedman, *Brownian dynamics as smart Monte Carlo simulation*, The Journal of Chemical Physics **69** (1978), 4628–4633.

[120] R.K. Sawyer, *Social Emergence: Societies As Complex Systems*, Cambridge University Press, 2005.

[121] A. G. Schlijper, P. J. Hoogerbrugge, and C. W. Manke, *Computer simulation of dilute polymer solutions with the dissipative particle dynamics method*, Journal of Rheology **39** (1995), 567–579.

[122] I. J. Schoenberg, *Contributions to the problem od approximation of equidistant data by analytic functions: part A*, Q. Appl. Math. **IV** (1946), 45–99.

[123] M. Selhammar, *The Use of Non-Spherical Kernels in Smooth Particle Hydrodynamics*, Arxiv preprint astro-ph/9706071 (1997).

[124] M. Serrano, *Comparison between smoothed dissipative particle dynamics and Voronoi fluid particle model in a shear stationary flow*, Physica A Statistical Mechanics and its Applications **362** (2006), 204–209.

[125] M. Serrano and P. Español, *Thermodynamically consistent mesoscopic fluid particle model*, Physical Review E **64** (2001), no. 4, 046115–+.

[126] Y. Shi, *Reevaluating Amdahl?s Law and Gustafson?s Law*, Computer Sciences Department, Temple University (MS: 38-24), Oct (1996).

[127] M.S. Stay, J. Xu, T.W. Randolph, and V.H. Barocas, *Computer Simulation of Convective and Diffusive Transport of Controlled-Release Drugs in the Vitreous Humor*, Pharmaceutical Research **20** (2003), no. 1, 96–102.

[128] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, Oxford University Press, 2001.

[129] D. ter Haar (ed.), *Collected Papers of L. D. Landau*, Pergamon, Oxford, 1965.

[130] C. Thieulot and P. Español, *Non-isothermal diffusion in a binary mixture with smoothed particle hydrodynamics*, Computer Physics Communications **169** (2005), 172–176.

[131] H. Uehara, M. Tamura, and M. Yokokawa, *MPI performance measurement on the Earth Simulator*, NEC research & development **44** (2003), no. 1, 75–79.

[132] I. Vattulainen, M. Karttunen, G. Besold, and J. M. Polson, *Integration schemes for dissipative particle dynamics simulations: From softly interacting systems towards hybrid models*, Journal of Chemical Physics **116** (2002), 3967–3979.

[133] Loup Verlet, *Computer Experiments on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules*, Phys. Rev. **159** (1967), no. 1, 98.

[134] F.J. Vesely, *Computational Physics: An Introduction*, Kluwer Academic/Plenum Publishers, 2001.

[135] Eric W. Weisstein, *Heat Conduction Equation*, From Mathworld - A Wolfram Web Resource, (`http://mathworld.wolfram.com/HeatConductionEquation.html`).

[136] D.A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction*, Springer, 2000.

[137] P. Wróblewski, K. Boryczko, and M. Kopeć, *SPH - a comparison of neighbor search methods based on constant number of neighbors and constant cut-off radius*, TASK Quart. **11** (2007), 275–285.

[138] L.T. Zhang, W.K. Liu, S.F. Li, D. Qian, and S. Hao, *Survey of Multi-Scale Meshfree Particle Methods*, Meshfree Methods for Partial Differential Equations (2002), 441–458.