
Politechnika Lubelska
Wydział Elektrotechniki i Informatyki
Instytut Informatyki

Algorytm równoległy dla problemu n-ciał

autor: Tomasz Nowicki

Lublin 2007

Spis treści

1. Wstęp.....	4
1.1. Wprowadzenie do zagadnienia i cel pracy.....	4
1.2. Zakres pracy.....	5
2. Przetwarzanie równoległe.....	6
2.1. Klasyfikacje systemów równoległych.....	6
2.1.1. Taksonomia Flynna.....	6
2.1.2. Klasyfikacja Erlangera (Erlanger Classification Scheme ECS).....	9
2.1.3. Klasyfikacja oparta o organizację zasobów.....	9
2.1.4. Klasyfikacja oparta o integrację mikroprocesorów.....	11
2.1.5. Klasyfikacja oparta o stopień rozproszenia.....	12
2.2. Techniki dekompozycji.....	12
2.2.1. Dekompozycja geometryczna.....	13
2.2.2. Dekompozycja iteracyjna.....	14
2.2.3. Dekompozycja rekurencyjna.....	15
2.2.4. Dekompozycja spekulatywna.....	15
2.2.5. Dekompozycja funkcjonalna.....	16
2.3. Rodzaje równoległości.....	17
2.3.1. Równoległość homogeniczna.....	17
2.3.2. Równoległość heterogeniczna.....	18
2.4. Szybkość i efektywność systemów równoległych.....	19
2.4.1. Lokalność procesu.....	19
2.4.2. Granularność obliczeń.....	19
2.4.3. Opóźnienie startu procesów.....	20
2.4.4. Dobór granularności obliczeń.....	20
2.4.5. Przyspieszenie.....	21
2.4.6. Zrównoleglenie programu.....	21
2.4.7. Efektywność wykorzystania procesorów.....	22
3. Klastry komputerowe.....	24
3.1. Struktury klastrowe.....	24
3.1.1. Klastry symetryczne.....	24
3.1.2. Klastry asymetryczne.....	25
3.1.3. Klastry rozwinięte (extended clusters).....	25
3.2. Popularne systemy klastrowe.....	27
3.2.1. Beowulf.....	27
3.2.2. Oscar.....	27
3.2.3. NPCI Rocks.....	27

4. Biblioteka MPI.....	28
4.1. Charakterystyka biblioteki MPI.....	28
4.2. Funkcje biblioteczne MPI.....	29
4.2.1. Organizacja procesów.....	29
4.2.2. Rodzaje Komunikacji.....	30
4.2.3. Komunikacja typu punkt-punkt (point – to – point).....	31
4.2.4. Komunikacja zbiorowa typu rozsyłanie (broadcast).....	32
4.2.5. Komunikacja zbiorowa typu rozrzucenie (scatter).....	33
4.2.6. Komunikacja zbiorowa typu zebranie (gather).....	33
4.2.7. Komunikacja zbiorowa typu redukcja (reduction).....	34
4.2.8. Komunikacja zbiorowa typu bariera (barrier).....	35
5. Realizacja klastra komputerowego.....	36
5.1. Sposób realizacji klastra.....	36
5.2. Szczegóły implementacyjne.....	38
5.2.1. Synchronizacja czasu.....	38
5.2.2. Dzielony zasób dyskowy.....	39
5.2.3. Komunikacja sieciowa.....	39
5.2.4. Odzworowanie nazw węzłów.....	40
5.2.5. Biblioteka Mpich.....	40
5.2.6. Wydajność komunikacji.....	40
5.2.7. Opis zbudowanego systemu wieloprocesorowego.....	41
6. Problem n – ciało.....	42
6.1. Uogólniony problem n-ciała.....	42
6.1.1. Sformułowanie problemu.....	42
6.1.2. Algorytm cząstka – cząstka (particle – particle PP).....	44
6.1.3. Algorytm cząstka – siatka (particle – mesh PM).....	45
6.1.4. Algorytm cząstka – cząstka – cząstka – siatka (particle – particle – particle – mesh P3M).....	45
6.2. Przykłady problemów technicznych, w których występuje problem n – ciało.....	46
6.2.1. Grawitacja.....	46
6.2.2. Tworzenie korelacji przestrzennych.....	48
6.2.3. Metoda wirów dyskretnych.....	50
7. Metoda wirów dyskretnych.....	51
7.1. Podstawowe równania.....	51
7.2. Kinematyka wirowa.....	52
7.2.1. Wir dyskretny.....	52
7.2.2. Wzajemne oddziaływanie wirów.....	53
7.2.3. Kinematyka wirowa.....	54
7.2.4. Szczególne przypadki.....	56
7.2.5. Szybki algorytm.....	58

7.3.Zastosowanie DVM.....	62
7.3.1.Opływ budynku.....	63
7.3.2.Przewidywanie dynamicznej odpowiedzi konstrukcji z uwzględnieniem efektów aeroelastycznych.....	63
8.Implementacja algorytmu.....	65
8.1.Założenia.....	65
8.2.Zrealizowane w ramach pracy programy komputerowe.....	67
8.2.1.Formaty plików używanych przez programy.....	67
8.2.2.Program vorsym_s.....	68
8.2.3.Program vorsym_q.....	68
8.2.4.Program vorsym_p.....	70
8.2.5.Program vsread.....	73
8.2.6.Program vrb2txt.....	74
8.2.7.Program vrb2bmp.....	74
8.3.Przeprowadzone symulacje.....	74
8.4.Wyniki symulacji.....	76
8.4.1.Wpływ ilości subdomen na czas obliczeń.....	76
8.4.2.Czasy trwania symulacji.....	77
8.4.3.Przyspieszenie i efektywność wykorzystania procesorów.....	82
9.Podsumowanie i wnioski.....	87
10.Literatura.....	88
11.Dodatek.....	90

1. Wstęp

1.1. Wprowadzenie do zagadnienia i cel pracy

Kilkudziesięcioletnia historia technik komputerowych może utwierdzić w przekonaniu, iż nieustannie przyśpieszający ich rozwój nigdy nie nadąży za również nieustannie zwiększającymi się potrzebami ludzkości. Wydaje się, że nowe możliwości osiągnięte za sprawą taniej - a zatem dostępnej obecnie każdemu obywatelowi krajów rozwiniętych – technologii komputerowej, stymulują nowe potrzeby. Rosnące oczekiwania użytkowników komputerów wymuszają stosowanie efektywniejszych technik obliczeniowych począwszy od wydajniejszych układów elektronicznych poprzez konstruowanie lepszych algorytmów aż do organizowania istniejących zasobów komputerowcy w spójne systemy wykonujące zadania z dużą wydajnością.

Tendencją ostatnich lat jest zwrócenie się w stronę ostatniego z wymienionych rozwiązań. Podyktowane jest to wyczerpaniem się innych łatwych w eksploatacji zasobów. Elektronika osiągnąwszy skalę nanometrów, tj. skalę pojedynczych atomów, dotarła do fizycznej bariery. Algorytmy są domeną twórczej działalności umysłu ludzkiego, trudno więc jest przewidzieć kiedy pojawią się nowsze i bardziej wydajne. Zaś łączenie pojedynczych jednostek obliczeniowych w systemy jest dziś łatwe ze względu na niskie koszty takiego rozwiązania. Obecnie średniozamożny konsument jest w stanie nabyć komputer z wielordzeniowym mikroprocesorem, komputerowy klaster można zbudować w oparciu o istniejące laboratoria. Zakładając wykorzystanie wyłącznie Otwartego Oprogramowania (*Open Source Software*), koszt finansowy takiego rozwiązania jest zerowy. Również przemysł inwestuje w rozwiązania klastrowe tworząc tym samym wirtualne laboratoria, które wypierają tradycyjne. Można spodziewać się, że w najbliższej przyszłości będą pojawiać się nowe wersje znanych programów komputerowych w wersjach równoległych.

Moja praca stara się nadążyć za istniejącym nurtem. W swym zakresie objęła budowę klastra komputerowego przy wykorzystaniu dostępnego laboratorium komputerowego i Otwartego Oprogramowania, napisania równoległego programu komputerowego oraz porównania możliwości klastra z możliwościami pojedynczego

komputera. Jako problem do rozważań wybrałem problem n-ciał – zagadnienie, które często występuje w technice i jest istotną barierą w przypadku implementacji wielu atrakcyjnych od strony fizycznej modeli teoretycznych.

1.2. Zakres pracy

Praca swoim zakresem obejmuje:

- Zapoznanie się z technikami programistycznymi stosowanymi w programowaniu równoległym
- Realizacja klastra komputerowego
- Studia literatury dotyczącej komputerowych symulacji bezpośrednich z zastosowaniem cząstek
- Realizacja programów komputerowych wykonujących symulację przy użyciu elementów wirowych w wariancie szeregowym jak i równoległym
- Wykonanie symulacji komputerowych
- Ocena uzyskanych wyników

2. Przetwarzanie równoległe

2.1. Klasyfikacje systemów równoległych

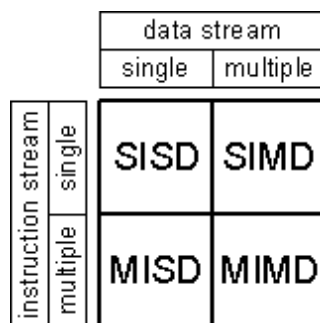
Systemy równoległe mogą być klasyfikowane na wiele sposobów. Pięć najczęściej spotykanych podziałów to:

- klasyfikacja Flynna
- klasyfikacja Erlangera
- klasyfikacja oparta o organizację zasobów
- klasyfikacja oparta o integrację mikroprocesorów
- klasyfikacja oparta o stopień rozproszenia

2.1.1. Taksonomia Flynna

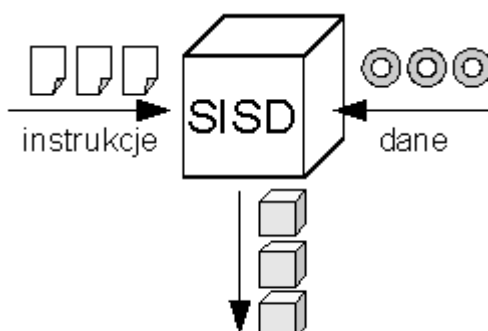
Podstawą tej klasyfikacji [1] [3] jest liczba strumieni danych i liczba strumieni rozkazów, rozróżnialnych w pracy systemu (rys. 2.1). Oddzielne grupy systemów charakteryzują się krotnością układów potrzebnych do obsługi tych strumieni. Rozróżnione zostały 4 warianty krotności strumieni:

- S I S D (*Single Instructon Stream – Single Data Stream*) - pojedynczy strumień rozkazów – pojedynczy strumień danych
- S I M D (*Single Instruction Stream – Multiple Data Stream*) – pojedynczy strumień rozkazów – wielokrotny strumień danych
- M I S D (*Multiple Instruction Stream – Single Data Stream*) – wielokrotny strumień rozkazów – pojedynczy strumień danych
- M I M D (*Multiple Instruction Strem - Multiple Data Stream*) – wielokrotny strumień rozkazów – wielokrotny strumień danych



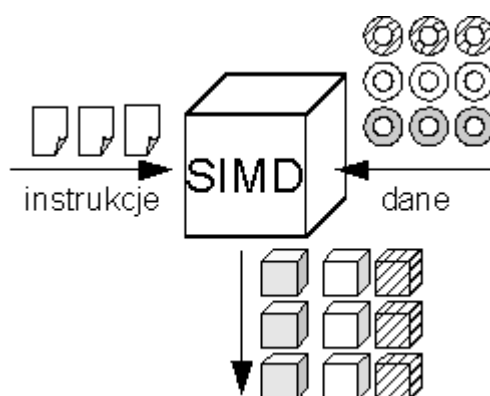
Rys 2.1: Taksonomia Flynna - wszystkie możliwe kombinacje krotności strumieni. (Opracowanie własne)

Przypadek pojedynczego strumienia danych i pojedynczego strumienia instrukcji (rys. 2.2) jest typowy dla dużej rodziny klasycznych komputerów, w których rozkazy wykonywane są w sposób sekwencyjny. Wszystkie pojedyncze systemy mikroprocesorowe oparte o architekturę von Numana należą do tej kategorii.



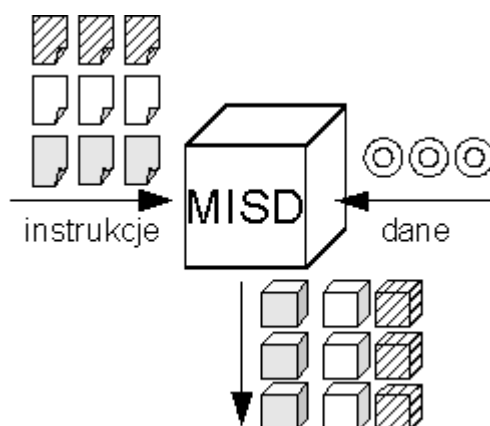
Rys 2.2: Taksonomia Flynna – przypadek pojedynczego strumienia rozkazów i pojedynczego strumienia danych. (Opracowanie własne)

Przypadek pojedynczego strumienia danych i wielokrotnego strumienia instrukcji (rys. 2.3) wykorzystywany jest wszędzie tam, gdzie zachodzi potrzeba przetworzenia dużych ilości danych o jednakowej strukturze. Systemy takie nazywane są często macierzami mikroprocesorów lub komputerami wektorowymi. Każdy mikroprocesor wykonuje identyczne operacje (identyczny strumień instrukcji) na innej części danych (różne strumienie danych). Do grupy SIMD zalicza się zarówno mikroprocesory posiadające jedną jednostkę zarządzającą i wiele jednostek arytmetycznych jak i procesory pracujące w trybie strumieniowym (*pipeine*). Do tej grupy zaliczają się procesory *Pentium MMX* lub *SSE*.



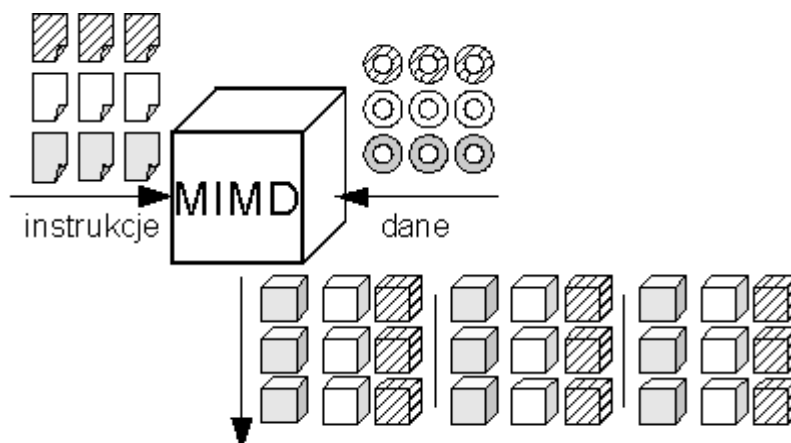
Rys 2.3: Taksonomia Flynna – przypadek pojedynczego strumienia instrukcji i wielokrotnego strumienia danych. (Opracowanie własne)

Architektura wielokrotnego strumienia danych i pojedynczego strumienia instrukcji (rys. 2.4) jest realizowana rzadko. Przykładem jest system CDC-6000. W tym przypadku każda z jednostek przetwarzających wykonuje inny cykl instrukcji na tym samym strumieniu danych.



Rys 2.4: Taksonomia Flynna – przypadek wielokrotnego strumienia instrukcji i pojedynczego strumienia danych. (Opracowanie własne)

Przypadek wielokrotnego strumienia danych i wielokrotnego strumienia instrukcji (rys. 2.5) wymaga asynchronicznego działania jednostek przetwarzających, tzn. macierzy mikroprocesorów. Oznacza to, że jednostki te muszą mieć możliwość niezależnego działania i muszą mieć swobodny dostęp zasobów. Obecnie jest to najbardziej rozpowszechniona grupa komputerów obliczeniowych o dużej mocy.



Rys 2.5: Taksonomia Flynna – przypadek wielokrotnego strumienia instrukcji i wielokrotnego strumienia danych.

2.1.2. Klasyfikacja Erlangera (*Erlanger Classification Scheme ECS*)

Klasyfikacja ta [15] została wprowadzona przez Handlera (1983) opisuje ona system równoległy poprzez podanie ilości jednostek sterujących i jednostek przetwarzających. W opisie tej klasyfikacji architektura komputera składa się z:

- ilości jednostek sterujących
- ilości jednostek przetwarzających
- ilości bitów na jednostkę

Klasyfikacja ta może mieć zastosowanie zarówno do opisu superkomputera jak i klastra zbudowanego z osobnych komputerów. Na przykład opis $ECS=(32,2,32)$ oznacza system równoległy złożony z 32 procesorów, z których każdy posiada 2 jednostki arytmetyczko-logiczne i wszystkie procesory są 32 bitowe.

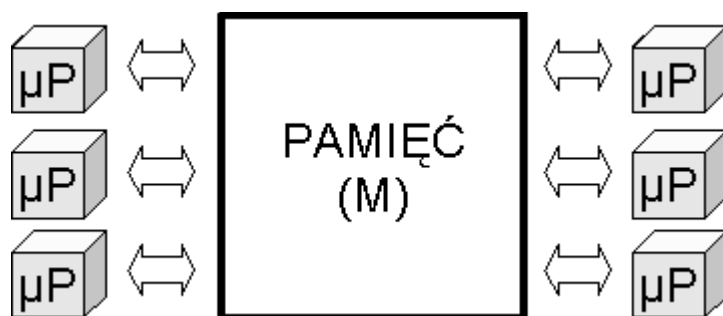
2.1.3. Klasyfikacja oparta o organizację zasobów

Klasyfikacja ta [1] [3] oparta jest na sposobie użycia pamięci przez procesory składające się na system równoległy. Możliwe jest wyróżnienie 3 przypadków:

- systemy z pamięcią dzieloną
- systemy z pamięcią rozproszoną

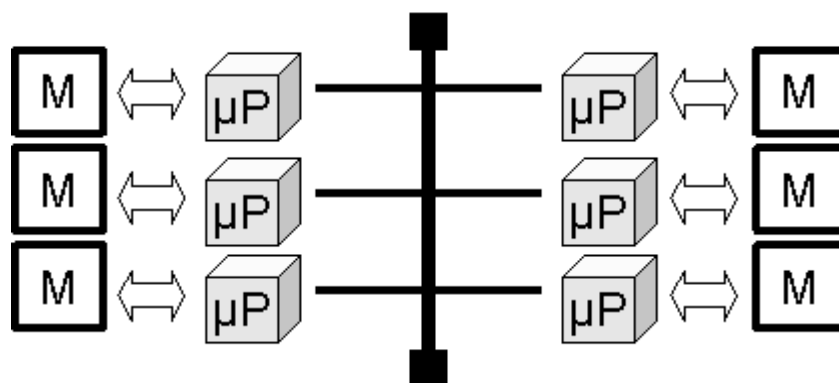
- systemy z pamięcią rozproszoną i dzieloną

Najważniejszą cechą architektury z pamięcią dzieloną jest istnienie globalnej przestrzeni adresowej do której każdy z procesorów ma równy dostęp (rys. 2.6). Komunikacja pomiędzy procesami zachodzi poprzez korzystanie z fizycznie tych samych struktur danych. Największą wadą takich systemów jest konieczność zabezpieczania się przed kolizjami. Zastosowanie mają semaforey, flagi, itp. Jest to przyczyna słabej skalowalności takich systemów.



Rys 2.6: System mikroprocesorowy ze wspólną, fizyczną przestrzenią adresową. (Opracowanie własne)

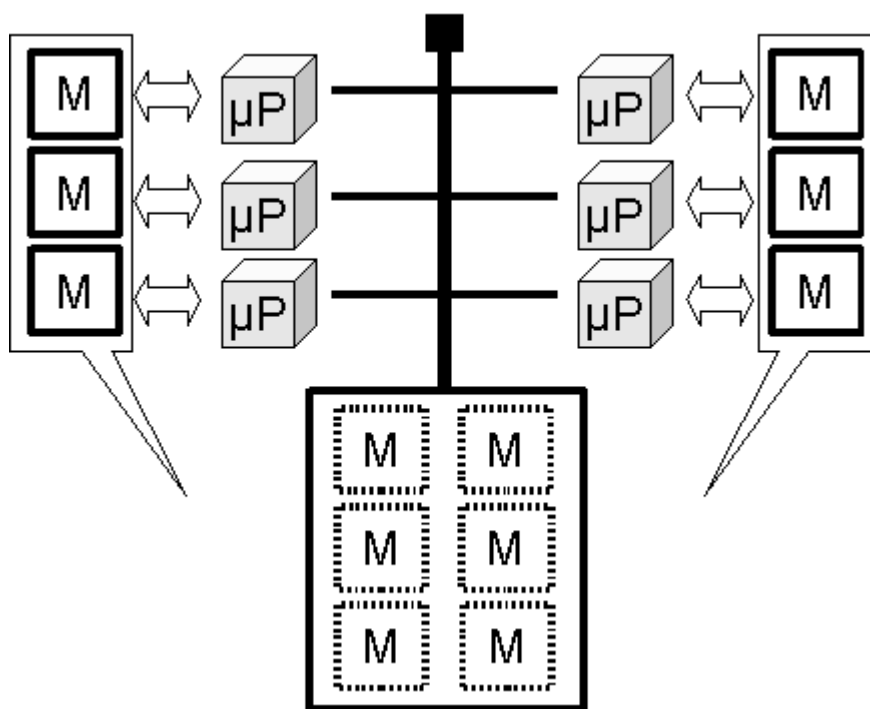
W systemie z pamięcią rozproszoną każdy procesor posiada własną pamięć niedostępną innym procesorom (rys. 2.7). Komunikacja międzyprocesowa odbywa się poprzez sieć lub magistralę systemową. Wymiana danych wymaga nawiązania połączenia pomiędzy procesorami. W takiej architekturze każdy procesor wyposażony w pamięć nazywany jest węzłem.



Rys 2.7: System mikroprocesorowy z fizycznie rozproszoną przestrzenią adresową. (Opracowanie własne)

System z pamięcią rozproszoną może zostać wzbogacony o wirtualną pamięć dzieloną (rys. 2.8). Najczęściej dzielony zasób jest realizowany programowo i jest częścią biblioteki równoległej. Użytkownik (programista) posługuje się dzielnym

zasobem tak samo jak w przypadku architektury z pamięcią dzieloną. Przy czym węzły mogą dalej posiadać własne zasoby pamięci lub nie. Przykładami takich systemów są *SGI Origin 2000* i *Cray*. Systemy takie pozwalają uruchamiać na fizycznie rozproszonych zasobach programy napisane dla środowisk z pamięcią dzieloną.



Rys 2.8: System mikroprocesorowy z fizycznie rozproszoną przestrzenią adresową tworzącą wspólną wirtualną przestrzeń adresową. (Opracowanie własne)

2.1.4. Klasyfikacja oparta o integrację mikroprocesorów

Klasyfikacja ta jest oparta o stopień elektronicznej integracji układów mikroprocesorowych, co pociąga za sobą rodzaj magistrali używanej do komunikacji między procesorowej. Istnieje możliwość wyróżnienia 4 grup:

- zintegrowana
- scentralizowana
- rozproszona
- mieszana

Systemy zintegrowane obejmują komputery z mikroprocesorami wielordzeniowymi. W takich systemach mikroprocesory (rdzenie) komunikują się

bezpośrednio poprzez szybką wewnętrzną magistralę mikroprocesorową. W komputery wyposażone w płyty główne obsługujące wiele procesorów tworzą grupę systemów scentralizowanych. Tutaj zastosowanie ma magistrala płyty głównej. Systemy rozproszone tworzone są przy użyciu zewnętrznych sieci komputerowych (np. *Ethernet*) i stanowią najczęściej zbiór samodzielnych komputerów połączonych siecią. Integracja mieszana jest dowolną kombinacją poprzednich.

2.1.5. Klasyfikacja oparta o stopień rozproszenia

Klasyfikacja ta [3] jest oparta o geograficzne rozproszenie współpracujących komputerów. Obecnie rozróżnia się 3 grupy:

- klastry komputerowe (*computer clusters*)
- sieci obliczeniowych (*computation grids*)
- zdalni klienci (*peer-to-peer computation*)

Klasterami komputerowymi nazywamy systemy tworzone w zasięgu sieci LAN. Klastery najczęściej mieszczą się w pojedynczych pomieszczeniach lub budynkach.

Systemy obliczeniowe łączone poprzez sieci rozległe WAN noszą miano sieci obliczeniowych poprzez analogię do sieci energetycznych. Sieci obliczeniowe są zbiorem super-węzłów, tzn. superkomputerów lub klasterów komputerowych.

Organizacja zdalnych klientów polega na istnieniu centralnego serwera obliczeniowego i dowolnej ilości dowolnie rozproszonych klientów obliczeniowych. Centralny serwer dzieli zadanie na małe fragmenty i wysyła je do aktualnie dostępnych klientów. Klient wykonuje obliczenia i wynik przesyła do serwera. Udanym przykładem takiej organizacji jest system *SETI@Home*.

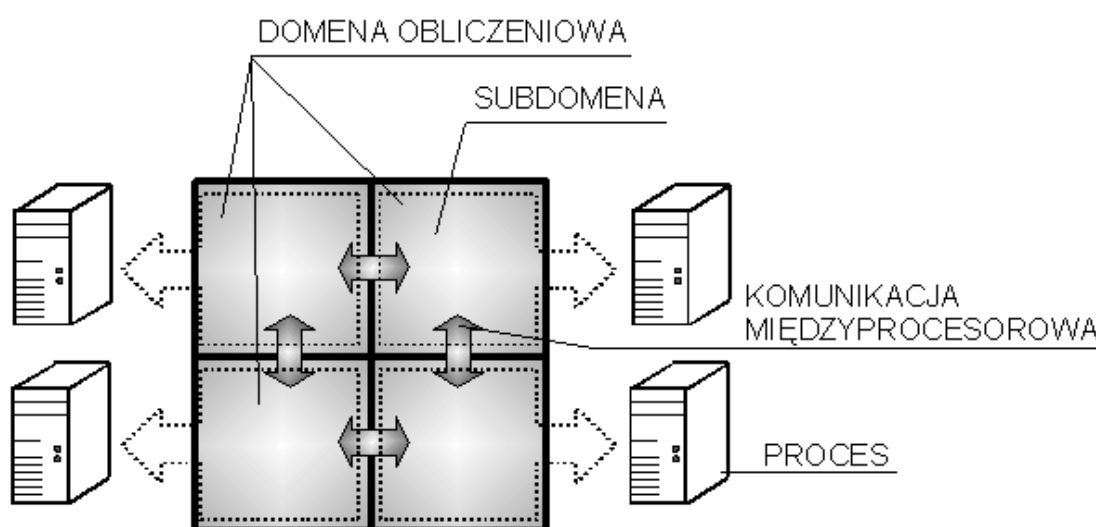
2.2. Techniki dekompozycji

Zrównoleglenie programu komputerowego zależy od specyfiki problemu, który ten program rozwiązuje. Sposób podziału zagadnienia na podsystemy, które mogą być wykonywane równocześnie ciągle pozostaje domeną twórczej aktywności człowieka. Przy dokonywaniu tego podziału wyłania się wiele problemów. Istnieją jednak ogólne kategorie sposobów dekompozycji, którymi można się kierować w

większości spotykanych problemach.

2.2.1. Dekompozycja geometryczna

Dekompozycja geometryczna [1] [3] (rys 2.9) ma zastosowanie w programach, które symulują zachowanie się systemów opisanych poprzez położenie na płaszczyźnie lub w przestrzeni. Przykładem może tutaj być różnego rodzaju pola, np.: pole elektromagnetyczne, pole prędkości przepływu itp. W tego typu zagadnieniach obszar zainteresowań (domena obliczeniowa) zostaje poddana podziałowi na subdomeny. Każdy fragment wynikający z podziału jest następnie obsługiwany przez oddzielny proces. Procesy pracują nad swoimi obszarami jednocześnie i wymieniają dane dotyczące wspólnych brzegów ich subdomen. Tego typu strategia jest typowa w przypadku rozwiązywania równań różniczkowych metodami różnicowymi (metoda różnic skończonych, elementów skończonych lub objętości skończonych).



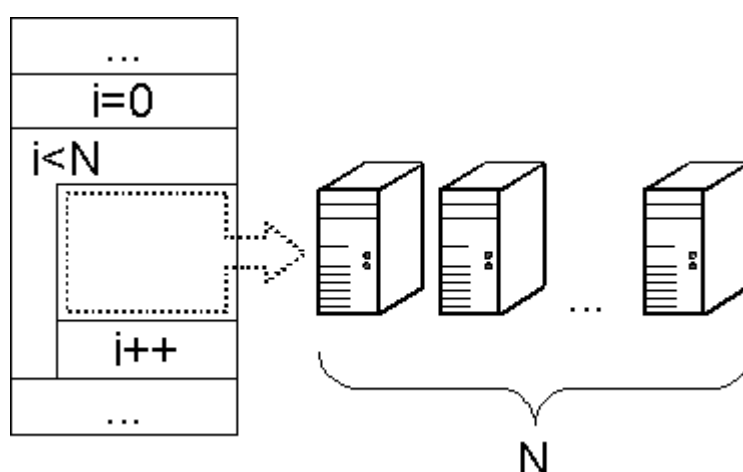
Rys 2.9: Dekompozycja geometryczna – podział domeny obliczeniowej. Symbol komputera oznacza niezależny proces obliczeniowy. (Opracowanie własne)

Dekompozycja geometryczna przynosi najlepsze efekty w przypadku kiedy obliczenia w każdym punkcie struktury zależ tylko i wyłącznie od położenia punktów w stosunku do punktu środkowego (punktu dla którego wykonywane są obliczenia). Punkty odległe o dany (stosunkowo mały) promień od punktu środkowego noszą nazwę *halo*. Od wielkości tego promienia zależy ilość danych, które muszą być użyte w obliczeniach i tym samym przekazane od innych procesów. Im ten promień jest mniejszy tym tych danych jest mniej i zrównoleglenie przynosi lepsze efekty.

2.2.2. Dekompozycja iteracyjna

W przypadku dekompozycji iteracyjnej [1] [3] (rys 2.10) ma zastosowanie fakt, że wiele programów sekwencyjnych zawiera pętlę. Ten rodzaj dekompozycji dotyczy tych przypadków, gdzie kolejne przebiegi pętli nie korzystają ze struktur danych będących przedmiotem zmiany poprzednich przebiegów pętli lub kiedy jest możliwe stworzenie hierarchicznej centralnej puli zadań. Jeżeli zadania nie wykorzystują wspólnych danych, hierarchia jest dowolna. W przypadku wykorzystywania wspólnych struktur danych lub zasobów hierarchia musi być starannie skonstruowana. Uruchamianie i zatrzymywanie zadań jest w takim przypadku kontrolowane przez osobny proces. Proces zarządzający jest odpowiedzialny za tworzenie kolejek do wspólnych zasobów i takie ułożenie operacji, aby eliminować zbędne obliczenia, tj. takie na które w danym momencie nie ma popytu.

Dekompozycja iteracyjna jest najbardziej efektywna w przypadku, gdy liczba pętli iteracyjnych jest znacznie większa niż liczba dostępnych procesorów. Jej efektywność zależy od programu zarządzającego. Klasycznym przykładem zastosowania tego rodzaju dekompozycji jest zrównoleglenie symulacji metody Monte Carlo. Ten rodzaj dekompozycji jest realizowany często poprzez dyrektywy kompilatora odpowiednio zaprojektowanych języków programowania np.: *High Performance Fortran* lub *OpenMP*.

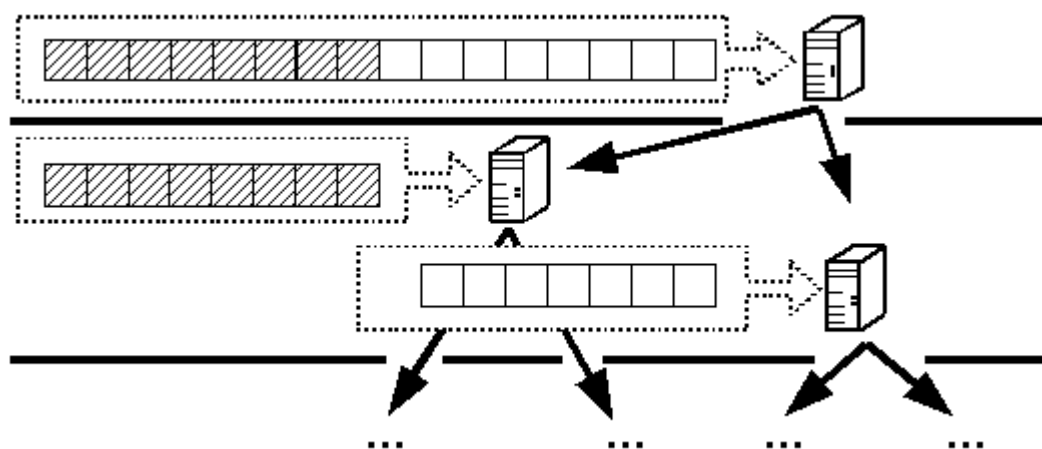


Rys 2.10: Dekompozycja iteracyjna – rozdział obiegów pętli. Symbol komputera oznacza niezależny proces obliczeniowy. (Opracowanie własne)

2.2.3. Dekompozycja rekurencyjna

Dekompozycja rekurencyjna [1] [3] (rys 2.11) jest podobna do iteracyjnej pod względem organizacji procesów – również wymaga zastosowania procesu zarządzającego. Jednak w przypadku dekompozycji rekurencyjnej źródłem nowego zadania (procesu) jest proces już istniejący (właśnie utworzony). W ten sposób oryginalny problem może zostać podzielony na większą ilość podproblemów, które są rozwiązywane przeciwsobnie. Wyniki rekurencji „zderzają się” ze sobą, prowadząc do rozwiązania problemu. Największym problemem tej techniki jest w niektórych zadaniach szybki (często wykładniczy) wzrost liczby procesów. Może to prowadzić do wyczerpania zasobów systemu komputerowego, np.: może nastąpić przepełnienie stosu.

Dekompozycja ta może być zastosowana do zrównoleglenia algorytmu szybkiego sortowania (*quick sort*). Główny proces otrzymuje zadanie posortowania wektora liczb. Proces ten dzieli wektor na dwie części i uruchamia kolejne dwa procesy, które mają to samo zadanie wykonać dla mniejszych wektorów.



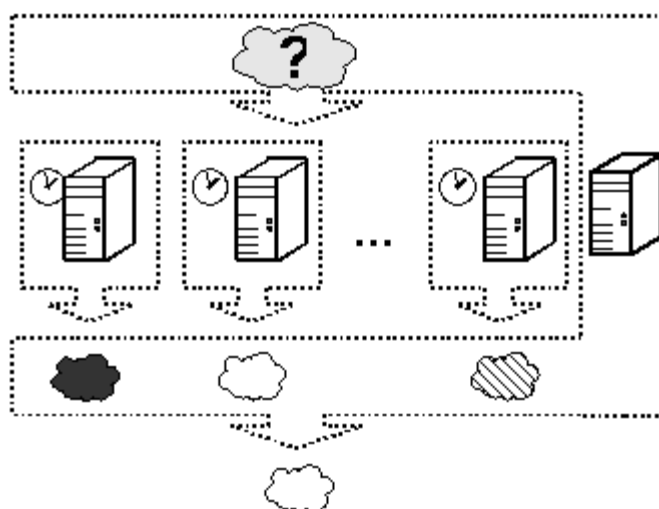
Rys 2.11: Dekompozycja rekurencyjna – drzewo procesów. Symbol komputera oznacza niezależny proces obliczeniowy. (Opracowanie własne)

2.2.4. Dekompozycja spekulatywna

Dekompozycja spekulatywna [1] [3] (rys 2.12) ma zastosowanie w przypadku grupy problemów, które mogą być rozwiązane różnymi metodami (najczęściej przybliżonymi). Problem należący do takiej grupy rozwiązywany jest niezależnie przez N różnych programów. Następnie wybierane jest rozwiązanie najlepsze według

zadanego kryterium. Zatem dekompozycja polega wyłącznie na zastosowaniu N technik rozwiązań jakiegoś problemu. Wykonywanie tych technik równoległe skraca czas analiz i pozwala przerwać obliczenia w sytuacji kiedy jeden z procesów zakończył działanie z możliwym do zaakceptowania wynikiem.

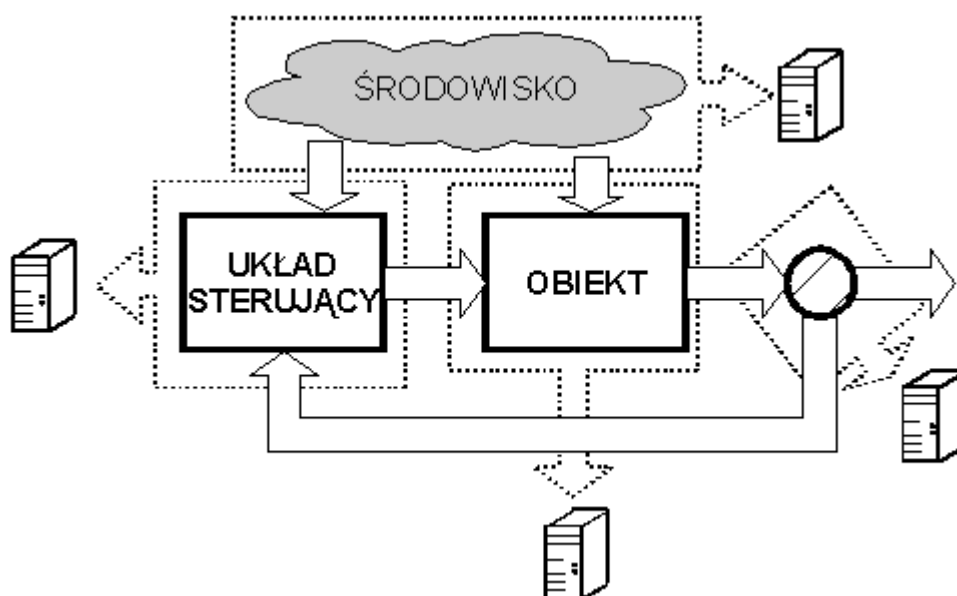
Przykładem zastosowanie mogą być zagadnienia symbolicznych przekształceń wyrażeń matematycznych lub algorytmy genetyczne.



Rys 2.12: Dekompozycja spekulatywna – selekcja optymalnego rozwiązania. Symbol komputera oznacza niezależny proces obliczeniowy. (Opracowanie własne)

2.2.5. Dekompozycja funkcjonalna

Dekompozycja funkcjonalna [1] [3] (rys. 2.13) może być zastosowana w przypadku modelowania systemów opartych o paradygmat producent – konsument. Podzespoły takiego systemu korzystają z wyników działania innych podsystemów lub / i są źródłem danych dla innych podsystemów. W przypadku takich problemów podziału zadania jest łatwy ponieważ odzwierciedla fizyczną strukturę zagadnienia. Zastosowanie współbieżności pozwala prowadzić symulacje w czasie rzeczywistym. Ten rodzaj dekompozycji jest często stosowany w zagadnieniach automatyki.



Rys 2.13: Dekompozycja funkcjonalna – podział problemu na producentów i konsumentów. Symbol komputera oznacza niezależny proces obliczeniowy. (Opracowanie własne)

2.3. Rodzaje równoległości

Przedstawione w poprzednim rozdziale techniki dekompozycji prowadzą do równoległych algorytmów służących do rozwiązania danego problemu. Miarą skomplikowania takiego algorytmu może być rodzaj równoległości.

Algorytmu równoległe homogeniczne należą do najłatwiejszych, zaś algorytmy heterogeniczne do trudniejszych w implementacji.

2.3.1. Równoległość homogeniczna

Równoległość homogeniczna [1] ma miejsce, gdy zadanie, które ma być wykonane, może zostać podzielone na identyczne co do wielkości (homogeniczne) podzadania. Każde takie podzadanie jest rozwiązywane sekwencyjnie przez oddzielny proces. Przy założeniu, że wszystkie procesy rozpoczną się równocześnie i zakończą się równocześnie, wówczas czas rozwiązania całego problemu określa wzór (2.1).

$$T_P = \frac{T}{P} + T_{kom} \quad (2.1)$$

gdzie:

P – ilość procesów

T_P – czas rozwiązania zadania przez P procesów

T – czas rozwiązania problemu przez jeden proces

T_{kom} – czas potrzebny na wymianę komunikatów.

Równoległość homogeniczna może być realizowana bez stosowania środków synchronizacji procesów.

Przypadki pełnej homogeniczności algorytmu występują rzadko. Dekompozycja geometryczna lub iteracyjna może w szczególnych przypadkach prowadzić do homogenicznych algorytmów równoległych.

2.3.2. Równoległość heterogeniczna

Równoległość heterogeniczna [1] ma miejsce wówczas, gdy zadanie, które ma być wykonane musi zostać podzielone na nieidentyczne (w szczególności różne co do wielkości) podzadania. Podzadania nie muszą być uruchamiane równocześnie. O czasie realizacji całej pracy decyduje proces o najdłuższym czasie działania (2.2).

$$T_P = \text{MAX}(T_1, T_2, T_3, \dots) \quad (2.2)$$

gdzie:

T_P – czas rozwiązania zadania przez P procesów

T_i – czas rozwiązania podzadania o numerze i .

Realizacja równoległości heterogenicznej wymaga stosowania środków synchronizacji procesów. Do takich środków zalicza się:

- semafor
- flagi
- tokeny.

Semafor służy do otwierania i zatrzymywania procesów. Flagi są służyć do sygnalizowania stanu procesów lub struktur danych. Tokeny przekazują dane między procesami.

2.4. Szybkość i efektywność systemów równoległych

Szybkość wykonania programu sekwencyjnego zależy wyłącznie od ilości obliczeń wykonywanych przez program. W przypadku programu równoległego czas jego wykonania jest sumą czasu obliczeń i czasu komunikacji pomiędzy poszczególnymi procesami.

2.4.1. Lokalność procesu

Lokalnością procesu [1] nazywa się stosunek ilości odniesień do pamięci lokalnej do ilości odniesień do pamięci zewnętrznej (2.3).

$$\text{lokalność procesu} = \frac{\text{ilość odniesień do pamięci lokalnej}}{\text{ilość odniesień do pamięci zewnętrznej}} \quad (2.3)$$

Przypadek, gdy proces realizuje zadanie odwołując się tylko do lokalnych zasobów, oznacza 100% lokalność działań. Jest to przypadek najkorzystniejszy, ale pozostaje wyłącznie w sferze teorii. Zaś przypadek, kiedy wszystkie odniesienia dotyczą zasobów zewnętrznych względem procesu oznacza zero lokalności i jest to przypadek najmniej korzystny.

Przedstawione w poprzednich rozdziałach techniki dekompozycji służą do takiego podziału zadania, aby maksymalnie zwiększyć lokalność procesów, tzn. ograniczyć komunikację pomiędzy procesami.

Optymalizacja algorytmu równoległego polega głównie na maksymalizowaniu lokalności procesów.

2.4.2. Granularność obliczeń

Granularnością obliczeń komputerowych [1] nazywa się ilość operacji, które są wykonywane pomiędzy zdarzeniami komunikacji międzyprocesorowej. Granulą nazywa się zbiór tych instrukcji. W zależności od wielkości granuli możemy programy równoległe dzielić na:

- programy o małej ziarnistości (kilka instrukcji kodu)
- programy o dużej ziarnistości (kilkadziesiąt instrukcji kodu)

Od granularności zależy możliwość zrównoważenia obciążenia procesorów. Od tego parametru zależy czy na procesory systemu równoległego spływa w przybliżeniu równy potok obliczeń. Zrównoważony potok obliczeń zapewnia najbardziej efektywne wykorzystanie dostępnych zasobów. Natomiast brak zrównoważenia oznacza występowanie wąskich gardeł, czyli istnienia przeciążonych zasobów przy jednoczesnym występowaniu zasobów nieużywanych.

Najkorzystniejszy jest przypadek drobnej granulacji przy jednoczesnej możliwości dowolnej dystrybucji granul. Wówczas możliwe jest dynamiczne przydzielanie granul procesorom w trybie priorytetu zapotrzebowań.

2.4.3. Opóźnienie startu procesów

Kolejnym czynnikiem silnie wpływającym na efektywność programu równoległego jest czas opóźnienia przy startowaniu odległych procesów [1]. Czas potrzebny na wysłanie komunikatów jest równy sumie dwóch czasów:

- czasu zużywanego na ustawienie parametrów przesyłu
- czasu zużywanego na przesłanie komunikatu

O ile ustawienie paramentów przesyłu jest zazwyczaj niezależne od wielkości komunikatu, tak czas zużywany na jego przesłanie zależy od ilości przesłanych danych.

Czas zużywany na przesyłanie komunikatów jest – z punktu widzenia użytkownika – czasem traconym. W związku z tym preferowana powinna być duża granulacja, przy której ogranicza się ilość przesyłanych komunikatów

2.4.4. Dobór granularności obliczeń

Z poprzednich dwóch rozdziałów wynika, że programista stoi przed konieczności doboru odpowiedniej granularności programu. Mała granularność pozwala na lepsze zrównoważenie obciążenia, lecz zwiększa opóźnienia wywołane komunikacją. Duża granularność ogranicza opóźnienia związane z komunikacją, ale może prowadzić do występowania wąskich gardeł.

Każdy program będzie zatem charakteryzował się wartością współczynnika obliczeń – komunikacja (*computation to communication ratio*) (2.4).

$$C_c = \frac{\text{ilość obliczeń}}{\text{ilość komunikacji}} \quad (2.4)$$

Poprzez ilość obliczeń należy rozumieć ilość operacji mikroprocesora mierzoną liczbą instrukcji arytmetycznych. Ilość komunikacji to liczba przychodzących komunikatów.

Wskazane jest uzyskanie dużego współczynnika C_c .

2.4.5. Przyspieszenie

Miarą korzyści jaka wypływa ze zrównoleglenia algorytmu jest przyspieszenie aplikacji [1]. Praktyczną miarą przyspieszenia obliczeń wykonywanych w systemie wyposażonego w P_2 procesorów względem systemu z P_1 procesorami jest stosunek czasów wykonania tego samego zadania (2.5).

$$S(P_1, P_2, N) = \frac{T(P_1, N)}{T(P_2, N)} \quad (2.5)$$

gdzie:

S – przyspieszenie

T – czas wykonywania zadania

N – umowny rozmiar zadania

P_i – ilość procesorów dostępnych w systemie o numerze i

Dla $P_2 > P_1$ oczekujemy $S > 1$.

2.4.6. Zrównoleglenie programu

Miarą zrównoleglenia programu [1] jest przyspieszenie liczone względem programu sekwencyjnego (2.6).

$$S_P(N) = \frac{T_I(N)}{T_P(N)} \quad (2.6)$$

gdzie:

$S_P(N)$ - zrównoleglenie programu

$T_I(N) = T(I, N)$ - czas wykonania programu sekwencyjnego

$T_P(N) = T(P, N)$ - czas wykonania programu równoległego na P procesorach

2.4.7. Efektywność wykorzystania procesorów

Przyśpieszenie programu S_P nie opisuje efektywności wykorzystania pojedynczych procesorów [1] [3]. Efektywnością taką jest stosunek osiągniętego przyśpieszenia do liczby procesorów (2.7).

$$e(P) = \frac{S_P(N)}{P} = \frac{T_1(N)}{P \cdot T_P(N)} \quad (2.7)$$

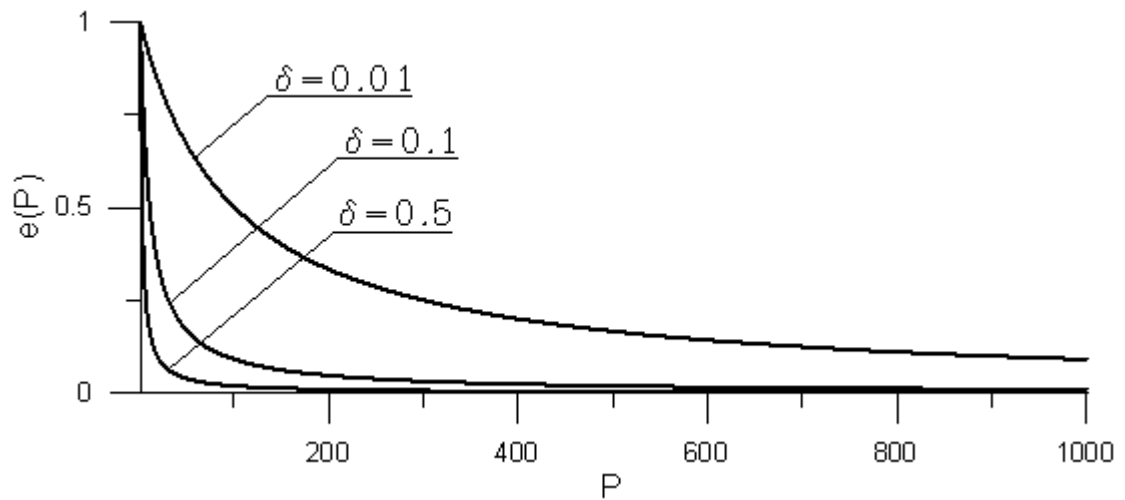
W rzeczywistych zagadnieniach dynamika wzrostu prędkości programu zależy od stosunku ilości szeregowych części programu do części równoległych. Przyjmując, że program zawiera dwa rodzaje obliczeń: szeregowe (operacje wejścia/wyjścia, komunikacja, dostęp do wspólnej zmiennej) i takie, które mogą być zrównoleglone i wykonane na dowolnej liczbie procesorów i oznaczając przez δ ($0 < \delta < 1$) ilość obliczeń szeregowych, zaś przez $(1 - \delta)$ ilość obliczeń zrównoleglonych można zapisać:

$$S_P(N) = \frac{T_1(N)}{T_P(N)} = \frac{T_1(N)}{\delta T_1(N) + (1 - \delta) \frac{T_1(N)}{P}} = \frac{1}{\delta + \frac{1 - \delta}{P}} \quad (2.8)$$

tak więc:

$$e(P) = \frac{1}{P \delta + (1 - \delta)} \quad (2.9)$$

Prawo to zostało sformułowane po raz pierwszy przez Gene Amdahla w 1967 roku (IBM). Wynika z niego, że niezależnie od tego ile procesów użyjemy w obliczeniach wzrost prędkości programu będzie zawsze ograniczony przez parametr δ . Ze wzoru widać również, że efektywność wykorzystania procesora przy wzroście ilości procesorów będzie maleć do zera (rys. 2.14). Zjawisko to jest określane mianem malejących powrotów (*diminishing returns*).



Rys 2.14: Teoretyczna efektywność programu równoległego w zależności od ilości procesorów P dla różnych wartości części szeregowych programu δ . (Opracowanie własne)

Należy jednak zaznaczyć, że rozważania przedstawione w tym rozdziale zakładały niezależność proporcji pomiędzy częścią szeregową i równoległą programu od wielkości zadania N . Innymi słowy $\delta = \text{const}(N)$. Założenie takie w praktyce nie jest prawdziwe, tzn. proporcja ta zmienia się wraz ze zmianą wielkości zadania. Prawo Amdahla przedstawione w powyższej postaci również nie uwzględnia w sposób jawny czasu potrzebnego wykonanie komunikacji pomiędzy procesami. W przypadku sieci z dzieloną magistralą można uznać, że ten czas wchodzi do części szeregowej

3. Klastry komputerowe

Klastrem komputerowym nazywamy zespół pojedynczych komputerów pracujących razem. Klaster komputerowy składa się z 3 elementów:

- zbioru pojedynczych komputerów
- sieci komputerowej
- oprogramowania, które umożliwia komunikację pomiędzy komputerami

Rozwój wydajnych klastrów komputerowych nastąpił po upowszechnieniu się szybkich technologii lokalnych sieci komputerowych. Zaś rozpowszechnienie się tego rodzaju systemów było poprzedzone ugruntowaniem się pozycji Otwartego Oprogramowania. Klastry komputerowe znajdują zastosowanie w przypadku zadań dla których jest możliwy podział na części takie, że każda część jest przydzielona oddzielnemu komputerowi. Każdy komputer składający się na klaster nazywany jest węzłem. Węzeł dysponuje indywidualnymi zasobami (mikroprocesor, pamięć, system operacyjny, biblioteki). Klastry komputerowe budowane na bazie komputerów używanych również do innych celów (np. laboratoria komputerowe) nazywane są często sieciami stacji roboczych.

3.1. Struktury klastrowe

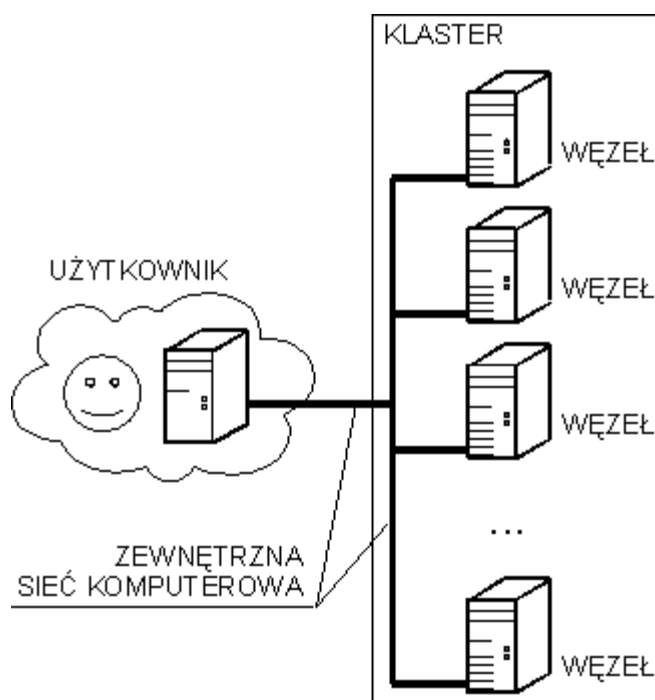
Klasyfikacja struktur klastrowych opiera się o role jakie pełnią poszczególne komputery w klastrze oraz o specyfikę sieci, które je łączy. Kolejne (bardziej skomplikowane) struktury powstają poprzez udoskonalanie struktur prostszych wraz z ich powiększaniem.

3.1.1. Klastry symetryczne

Klastry symetryczne [3] (rys. 3.1) reprezentują najprostszą strukturę klastrową. Żaden z węzłów nie jest wyróżniony. Każdy może funkcjonować jako oddzielny komputer. Węzły połączone są ogólnie dostępną siecią komputerową. Użytkownik może bezpośrednio połączyć się z każdym węzłem. Klaster taki może korzystać z

zewnętrznych serwerów konfiguracyjnych lub serwery takie mogą funkcjonować na dowolnie wybranych węzłach.

Wadą tej struktury jest konieczność wykonywania czynności konfiguracyjnych oddzielnie dla każdego z komputerów. Zaś wykorzystanie zewnętrznej sieci obniża bezpieczeństwo systemu oraz ma niekorzystny wpływ na szybkość pracy klastra.



Rys 3.1: Klastrowy symetryczny. (Opracowanie własne)

3.1.2. Klastry asymetryczne

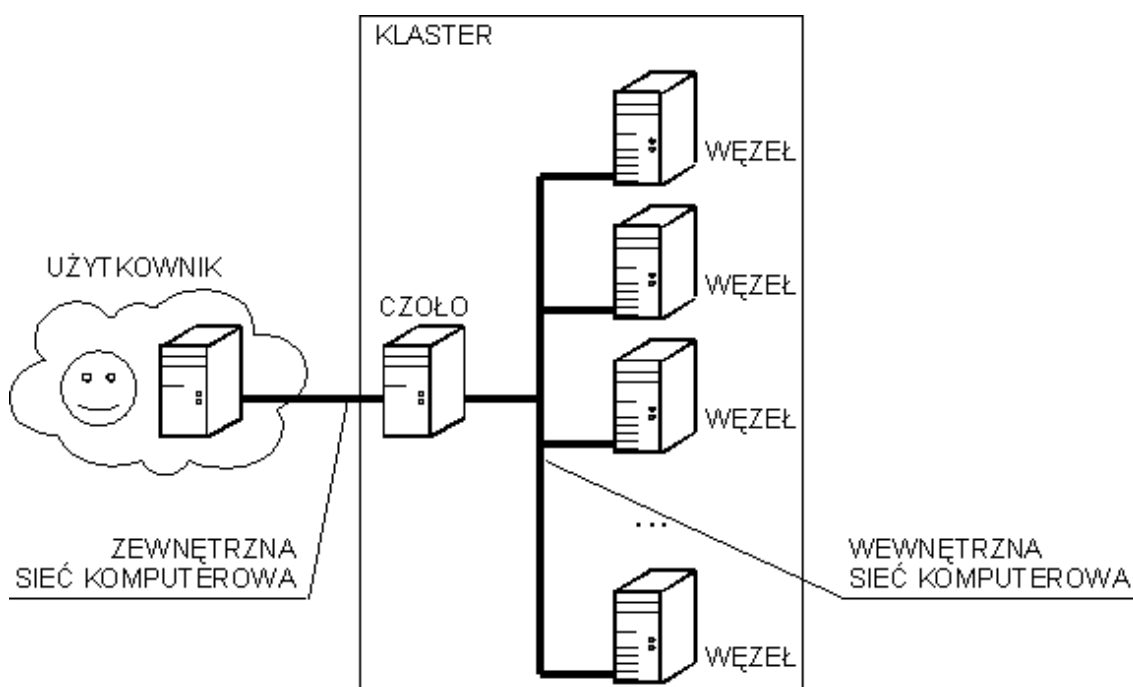
Najważniejszą cechą klastra asymetrycznego [3] (rys. 3.2) jest istnienie wewnętrznej sieci komputerowej i wyróżnionego węzła. Wyróżniony węzeł -czoło (*head*) pełni funkcje bramy sieciowej. Użytkownik może nawiązać bezpośrednie połączenie wyłącznie z czołem klastra. Funkcje konfiguracyjne pełni wyróżniony węzeł.

Wadą struktury asymetrycznej jest duże obciążenie jednego z węzłów. Przy dużej ilości pozostałych węzłów czoło systemu będzie opóźniać pracę całego systemu.

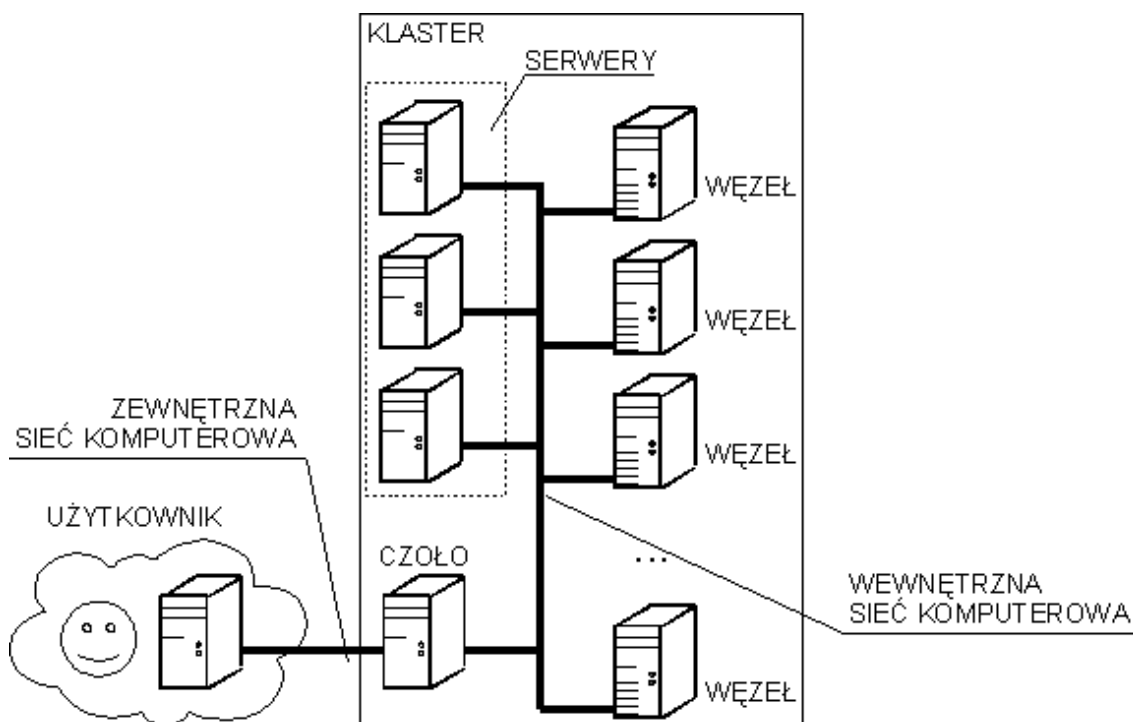
3.1.3. Klastry rozwinięte (*extended clusters*)

Klastry rozwinięte [3] (rys. 3.3) są wyposażone dodatkowo w dedykowane serwery

dostarczające usług sieciowych wyłącznie na potrzeby klastra. Serwerami takimi mogą być: DHCP, DNS, NTP, NFS.



Rys 3.2: Klaster asymetryczny. (Opracowanie własne)



Rys 3.3: Klaster rozwinięty. (Opracowanie własne)

3.2. Popularne systemy klastrowe

Bieżący paragraf opisuje najpopularniejsze oprogramowanie do budowy klastrów dostępne na licencji Otwartego Oprogramowania. Przedstawione pakiety służą do budowania systemów asymetrycznych. Jednak przy odpowiedniej konfiguracji mogą być użyte to tworzenia innych struktur.

3.2.1. *Beowulf*

Beowulf [14] jest zbiorem oprogramowania przeznaczonego dla komputerów klasy PC z systemem operacyjnym *Linux* (oryginalnie *Slackware*). Oprogramowanie to pozwala zbudować na takiej bazie skalowalny klaster obliczeniowy. Prace nad systemem rozpoczęto w 1994 roku w NASA. Celem ich było stworzenie alternatywy dla drogich superkomputerów. *Beowulf* zdobył dużą popularność w środowiskach akademickich i naukowych. Obecnie jest jednym z najczęściej implementowanych rozwiązań.

3.2.2. *Oscar*

Oscar (*Open Source Cluster Application Resources*) [13] jest oprogramowaniem klastrowym tworzonym przez *Open Cluster Group*. System bazuje na systemie operacyjnym *Linux*, lecz nie jest związany z żadną jego dystrybucją. Oprogramowanie instalowane jest na jednym z węzłów i następnie klonowane na pozostałe węzły. Dostarczane skrypty pozwalają w sposób instalować oprogramowanie w sposób automatyczny przy minimalnej ingerencji użytkownika. System zapewnia duży poziom automatyki przy instalacji i zarządzaniu klastrem.

3.2.3. *NPCI Rocks*

NPCI Rocks (*National Partnership for Advanced Computational Infrastructure*) działa w systemie *Red Hat Linux*. Dostarczany jest jako zmodyfikowane dystrybucje tego systemu, które instalują się razem z niezbędnymi bibliotekami i narzędziami. Dostępne są również automatyczne konfiguratorzy. System zapewnia najwyższy stopień automatyzacji i wygody użytkownika.

4. Biblioteka MPI

4.1. Charakterystyka biblioteki MPI

Biblioteka MPI (*Message Passing Interface*) [10] [11] jest obecnie podstawą każdego systemu klastrowego. MPI jest standardem opisującym sposób wymiany komunikatów w modelu obliczeń równoległych. Model ten oparty jest na dwóch głównych przesłankach:

- Obliczenia równoległe wykonywane są przez oddzielne procesy, z których każdy operuje na lokalnych strukturach danych. Nie jest możliwe, aby procesy odczytywały bądź zapisywały struktury danych przynależne innym procesom
- Wymiana danych pomiędzy procesami jest możliwa poprzez wymianę komunikatów. Wymiana komunikatów jest możliwa jedynie poprzez jawne wywoływanie instrukcji wysłania i odebrania komunikatu.

Model MPI nie precyzuje jaki system równoległy powinien być użyty do wykonywania procesów równoległych. W związku z tym implementacje MPI mogą być realizowane zarówno dla systemów wieloprocessorowych z dzieloną pamięcią jak i dla rozproszonych klastrów komputerowych.

Konieczność jawnej obsługi komunikatów daje możliwość dużej kontroli nad przepływem danych pomiędzy procesami. Implementacje MPI to biblioteki funkcji C lub FORTRAN-u, które pozwalają w prosty sposób realizować komunikację pomiędzy równoległe działającymi procesami.

W czasie pisania pracy znane były dwa standardy MPI. Standard MPI-1 został ogłoszony w 1994 roku. Swoją specyfikacją obejmuje:

- nazwy funkcji
- sekwencje wywołań funkcji
- rezultaty funkcji

Specyfikacja nie określa szczegółów implementacji.

Standard MPI – 2 będąc nadzbiorem MPI – 1 dodatkowo określa:

- równoległe funkcje realizujące wejście i wyjście
- mechanizm uruchamiania programów
- interfejs dla C++
- interfejs dla FORTRAN 90

Biblioteka MPI jest tak zaprojektowana, aby zapewniać:

- możliwość przenoszenia oprogramowania pomiędzy wszystkimi platformami, gdzie zaimplementowano tę bibliotekę
- możliwość implementacji standardu na szerokim spektrum platform sprzętowych
- obsługę heterogenicznych systemów komputerowych

4.2. Funkcje biblioteczne MPI

Program oparty o bibliotekę MPI wykonywany jest w wielu instancjach. Poszczególne instancje są programami szeregowymi, które komunikują się między sobą poprzez wywoływanie funkcji bibliotecznych.

Funkcje biblioteczne mogą być podzielone na 4 kategorie:

- funkcje służące do nawiązania, zarządzania i kończenia komunikacja
- funkcje służące do komunikacji typu punkt-punkt
- funkcje służące do komunikacji zbiorczej
- funkcje służące do tworzenie struktur danych niezależnych od platformy

W kolejnych podrozdziałach przedstawione są wybrane funkcje realizujące komunikację pomiędzy procesami.

4.2.1. Organizacja procesów

Organizacja procesów pozwala wyróżnić grupy procesów i pozwala wskazać proces w grupie. Grupy mogą mieć topologię liniową, tablicową lub tworzyć graf.

Komunikator jest strukturą danych opisującą grupę procesów. Stąd grupa procesów często jest nazywana komunikatorem. Pojedynczy proces może należeć do wielu grup. Proces nie może nie należeć do żadnej grupy. Każdy proces należy do predefiniowanej grupy o nazwie `MPI_COMMON_WORLD`. Procesy mogą się komunikować tylko wtedy kiedy należą do wspólnej grupy.

Utworzenie nowej grupy realizują instrukcje:

- `MPI_Group_incl` – tworzenie nowej grupy poprzez wskazanie procesów należących do istniejącej grupy
- `MPI_Group_excl` – tworzenie nowej grupy poprzez eliminację procesów należących do istniejącej grupy
- `MPI_Comm_create` – tworzenie nowego komunikatora którego członkami są wskazane procesy należące do istniejącej grupy

Usunięcie grupy, tzn. uwolnienie struktur danych z nią związanych, realizuje komenda `MPI_Group_free`.

Proces może uzyskać informacje o rozmiarze grupy (ilość procesów do niej należących) poprzez wywołanie instrukcji: `MPI_Comm_size`.

W obrębie grupy każdy proces jest identyfikowany poprzez unikatowy numer (*rank*). Procesy numerowane są w sposób ciągły począwszy od 0. Jeżeli proces należy do wielu grup wówczas w każdej grupie identyfikowany jest przez (zazwyczaj) inny numer. W przypadku wirtualnych topologii liniowej numer jest za pomocą funkcji bibliotecznych przeliczany na indeksy adekwatne dla danej topologii.

Proces może uzyskać informację o swoim numerze wywołując instrukcję: `MPI_Comm_rank`.

4.2.2. Rodzaje Komunikacji

Ze względu na czas dopełnienia komunikacji wyróżniamy

- komunikacja blokująca
- komunikacja nieblokująca

Blokująca instrukcja wysłania lub odbioru zostaje zakończona wówczas, kiedy komunikacja zostanie dopełniona. Po dopełnionej komunikacji program wykonuje

kolejną instrukcję. Natomiast nieblokujące instrukcje wysłania lub odbioru składają się z dwóch części: ustawienia żądania komunikacji i potwierdzenia komunikacji. Pomiędzy tymi dwoma instrukcjami może wystąpić dowolna ilość innych instrukcji.

Ze względu na ilość procesów biorących udział w komunikacji wyróżniamy;

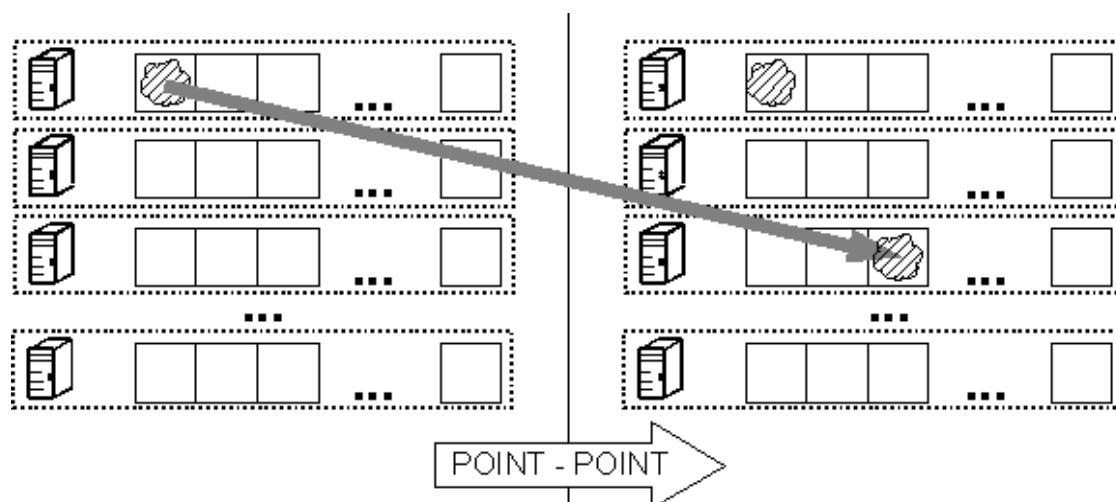
- komunikację typu punkt – punkt
- komunikację typu zbiorowego

Komunikacja typu zbiorowego (*braodcast, scatter, gather, reduction*) jest najczęściej realizowana poprzez użycie instrukcji służących do komunikacji punkt – punkt. Jednak jest to tak zorganizowane przez funkcje biblioteczne, że jest niewidoczne dla programisty.

4.2.3. Komunikacja typu punkt-punkt (*point – to – point*)

Komunikacja typu punkt – punkt (rys 4.1) jest podstawowym i najprostszym sposobem komunikacji bezpośredniej pomiędzy procesami. Jeden z procesów wysyła komunikat a drugi go odbiera. Procesy w jawny sposób określają się. Proces wysyłający określa numer adresata. Adresat zgłasza oczekiwanie na komunikat od procesu podając jego numer. Transfer danych jest możliwy kiedy oba procesy należą do tej samej grupy (komunikator). Komunikacja typu punkt – punkt jest możliwa w trybie blokującym i nieblokującym.

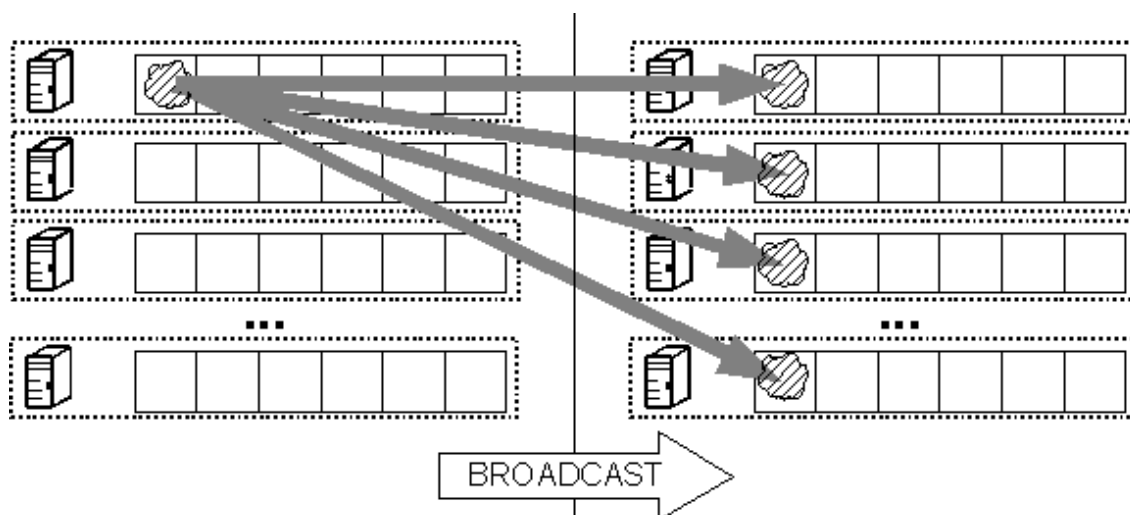
Tryb blokujący obsługuje para instrukcji: `MPI_Send` i `MPI_Recv`. Instrukcje te wywoływane są odpowiednio przez nadawcę i odbiorcę komunikatu. Tryb nieblokujący obsługują instrukcje: `MPI_Isend`, `MPI_IRecv`. Tryb nieblokujący wymaga potwierdzenie zakończenia transmisji. Służą do tego instrukcje: `MPI_Wait` lub `MPI_Test`.



Rys 4.1: Komunikacja typu punkt – punkt. Symbol komputera oznacza pojedynczy proces, zaś prostokąty oznaczają struktury danych należące do odpowiednich procesów. (Opracowanie własne)

4.2.4. Komunikacja zbiorowa typu rozsyłanie (*broadcast*)

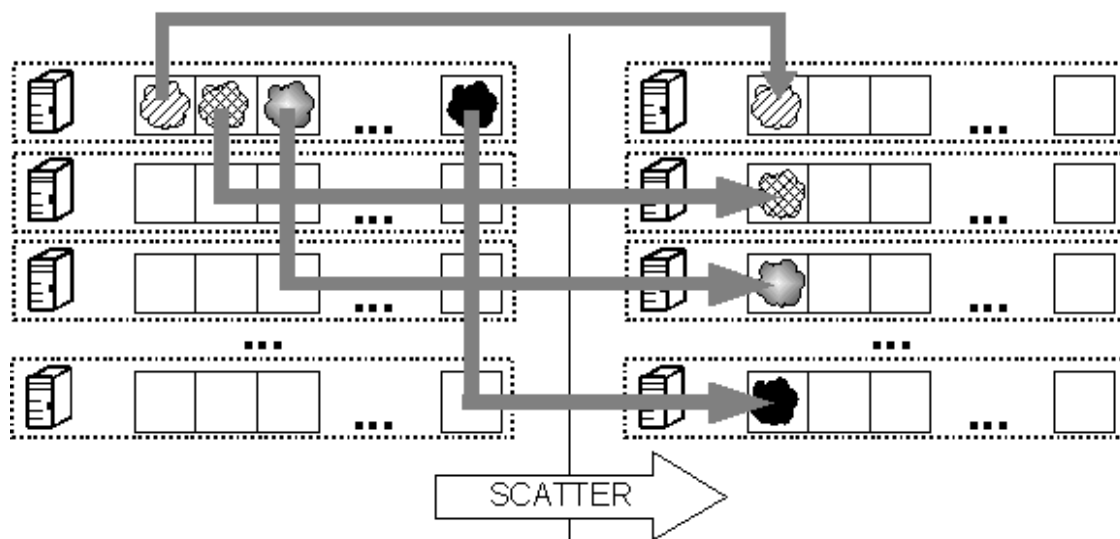
Istotą komunikacji typu rozsyłanie (rys 4.2) jest przesłanie kopii danych przez jeden proces wszystkim procesom tworzącym grupę rozgłoszeniową. Rozsyłanie realizuje instrukcja `MPI_Bcast`.



Rys 4.2: Komunikacja zbiorowa typu rozsyłanie. Symbol komputera oznacza pojedynczy proces, zaś prostokąty oznaczają struktury danych należące do odpowiednich procesów. (Opracowanie własne)

4.2.5. Komunikacja zbiorowa typu rozrzucenie (*scatter*)

Istotą komunikacji typu rozrzucenie (rys. 4.3) jest dystrybucja danych jednego procesu do pozostałych procesów w grupie. Przy czym każdy z procesów otrzymuje od procesu wysyłającego inną porcję danych. Komunikacja tego typu jest realizowana przez instrukcję `MPI_Scatter`.

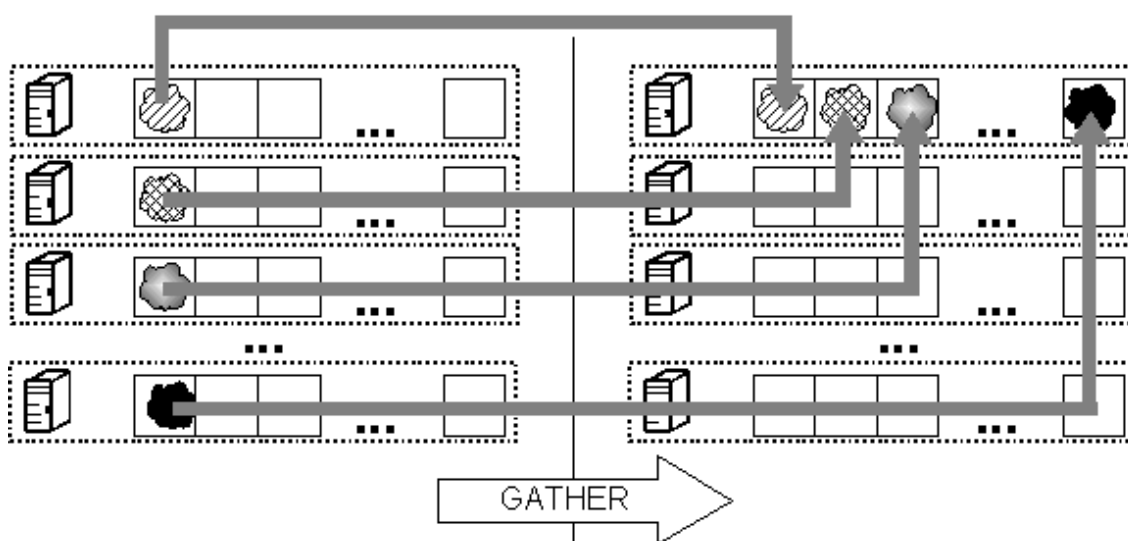


Rys 4.3: Komunikacja zbiorowa typu rozrzucenie. Symbol komputera oznacza pojedynczy proces, zaś prostokąty oznaczają struktury danych należące do odpowiednich procesów. (Opracowanie własne)

4.2.6. Komunikacja zbiorowa typu zebranie (*gather*)

Komunikacja typu zebranie (rys. 4.4) jest odwróceniem komunikacji typu rozrzucenie. Zachodzi wówczas, gdy wszystkie procesy należące do danej grupy przekazują dane jednemu procesowi. Przy czym proces będący adresatem zapisuje przychodzące dane w osobnych miejscach. Komunikację tego rodzaju realizuje funkcja `MPI_Gather`.

Odmianą komunikacji typu *gather* jest komunikacja *all gather*. Polega ona na tym, że dane gromadzone są w pamięci wszystkich procesów. Ten rodzaj komunikacji realizowany jest przez instrukcję `MPI_Allgather`. Instrukcja ta wywołuje instrukcję `MPI_Gather` dla wszystkich procesów w grupie.



Rys 4.4: Komunikacja zbiorowa typu zebranie. Symbol komputera oznacza pojedynczy proces, zaś prostokąty oznaczają struktury danych należące do odpowiednich procesów. (Opracowanie własne)

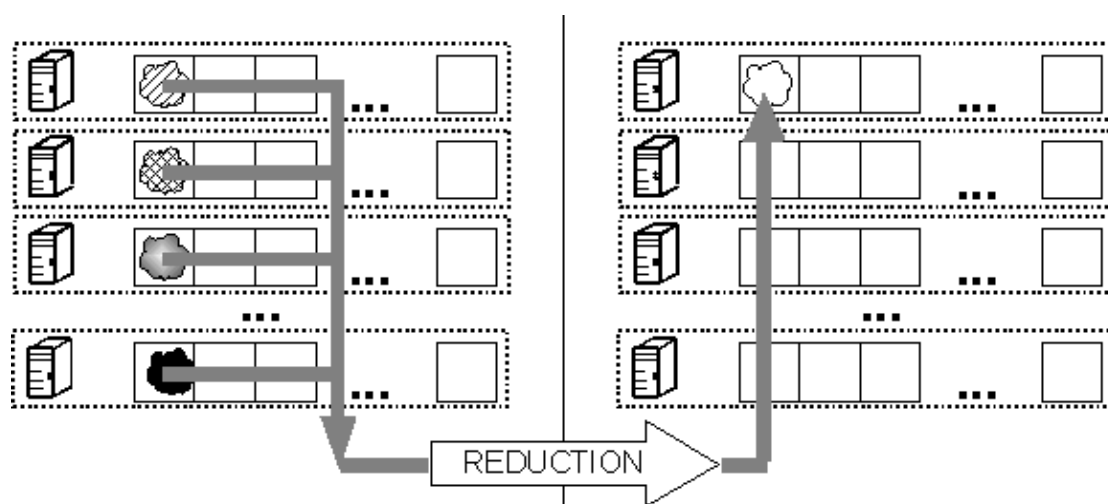
4.2.7. Komunikacja zbiorowa typu redukcja (*reduction*)

Istotą operacji redukcji (rys. 4.5) jest wykonanie operacji na danych należących do różnych procesów i zapisanie wyniku tej operacji w pamięci jednego z procesów. Operacją wykonywaną na danych może być:

- wyznaczenie wartości minimalnej
- wyznaczenie wartości minimalnej wraz z indeksem określającym położenie tego elementu
- wyznaczenie wartości maksymalnej
- wyznaczenie wartości maksymalnej wraz z indeksem określającym położenie tego elementu
- suma
- iloczyn
- suma logiczna
- bitowa suma logiczna
- iloczyn logiczny

- bitowy iloczyn logiczny
- logiczny xor
- bitowy xor

Operację redukcji realizuje polecenie: `MPI_Reduce`.



Rys 4.5: Komunikacja zbiorowa typu redukcja. Symbol komputera oznacza pojedynczy proces, zaś prostokąty oznaczają struktury danych należące do odpowiednich procesów. (Opracowanie własne)

4.2.8. Komunikacja zbiorowa typu bariera (*barrier*)

Bariera służy do synchronizacji procesów. Gwarantuje, że wszystkie procesy, których ona dotyczy, rozpoczną wykonywanie kolejnej instrukcji w jednakowym czasie.

Bariera realizowana jest przy użyciu instrukcji `MPI_Barrier`.

5. Realizacja klastra komputerowego

Klaster komputerowy został zrealizowany w Laboratorium Metod Numerycznych Wydziału Inżynierii Budowlanej i Sanitarnej Politechniki Lubelskiej (rys. 5.1).



Rys 5.1: Laboratorium Numeryczne Wydziału Inżynierii Budowlanej i Sanitarnej Politechniki Lubelskiej. (Fot. T. Nowicki)

Laboratorium to jest wyposażone w 12 jednoprocessorowych komputerów PC połączonych siecią komputerową *Fast Ethernet*. Jednostki komunikują się wykorzystując protokół TCP/IP w wersji 4. Komputery są wyposażone w mikroprocesory *AMD Athlon XP 1600+* 1.4 GHz i w 256 MB pamięci operacyjnej. Podczas projektowania systemu klastrowego należało uwzględnić następujące ograniczenia:

- komputery mają zainstalowany system operacyjny *Windows XP*
- laboratorium jest używane przez nieinformatyków
- w laboratorium występują częste awarie sprzętowe
- nie są przewidziane żadne nakłady finansowe na realizację projektu

5.1. Sposób realizacji klastra

Zdecydowałem, że nie będę stosował gotowych systemów klastrowych a cały system zostanie skonstruowany z podstawowego oprogramowania, które zostanie skonfigurowane indywidualnie.

Klaster został zbudowany w oparciu o system operacyjny *Ubuntu Linux 6.10*. System ten jest drugim obok *Windows XP* systemem operacyjnym. Domyślnie uruchamianym systemem jest system Windows. Rozwiązanie to pozwoliło na całkowite

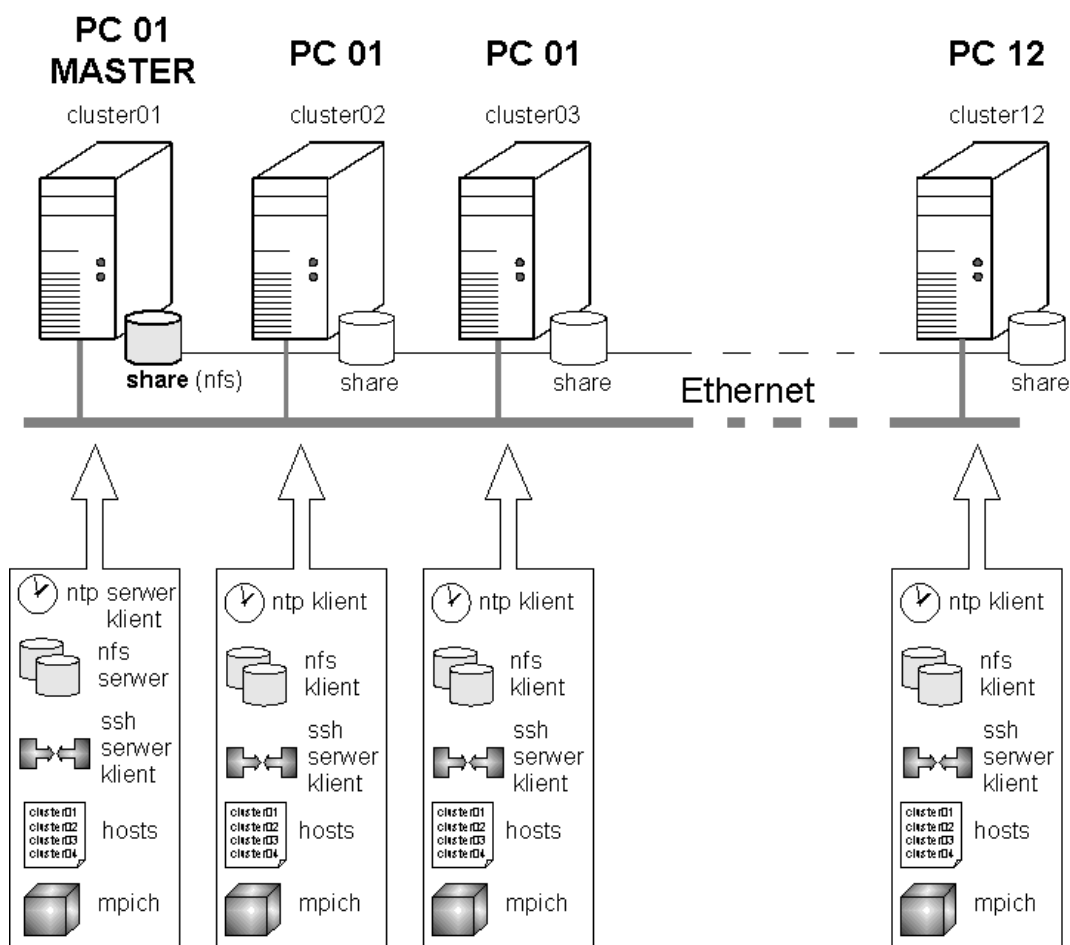
odseparowanie od siebie oprogramowania używanego do celów obliczeniowych od oprogramowania dydaktycznego. Nie wzbudzało też podejrzeń innych użytkowników laboratorium.

Zrealizowany klaster ma strukturę symetryczną. Jeden z komputerów został wybrany jako węzeł zarządzający. Zarządzanie ogranicza się jednak do dostarczania pozostałym komputerom następujących usług:

- synchronizacja czasu
- utrzymywanie dzielonego zasobu dyskowego.

W trakcie korzystania z klastra wszystkie węzły mogą być traktowane jako identyczne, a rolę węzła zarządzającego może przejąć dowolny inny komputer. Taka struktura systemu została wybrana ze względu na dużą awaryjność dostępnych komputerów.

Jako oprogramowanie klastrowe została użyta biblioteka *Mpich 2.0*. Programy niezbędne do działania klastra zostały obrazowo zestawiona na rysunku 5.1. Całość oprogramowania użyta do budowy klastra jest dostępna w zasobach Internetu na licencji Otwartego Oprogramowania i jest darmowa.



Rys 5.2: Schemat konfiguracji autorskiego klastra komputerowego.

5.2. Szczegóły implementacyjne

Poniższe paragrafy opisują szczegóły związane z realizacją klastra.

5.2.1. Synchronizacja czasu

W celu synchronizacji zegarów systemowych na węzłach klastra na jednym z węzłów został zainstalowany serwer czasu ntp (*Network Time Protocol*). Węzeł z serwerem czasu synchronizowany jest przy użyciu internetowych serwerów czasu. Zaś pozostałe węzły synchronizowane są lokalnie.

Do synchronizacji czasu zostały wybrane następujące publicznie dostępne serwery ntp:

2.pl.pool.ntp.org

0.europe.pool.ntp.org

2.europe.pool.ntp.org

5.2.2. Dzielony zasób dyskowy

Wyróżniony węzeł *cluster01* udostępnia katalog poprzez protokół NFS (*Network File System*). Tutaj umieszcza się plik do egzekucji na wszystkich węzłach oraz dane do przetworzenia. Tutaj zapisywany jest plik z wynikami.

Sieciowy system plików powinien być zainstalowany i skonfigurowany jako pierwsza dostępna usługa sieciowa. Pozwala on w wygodny sposób organizować dane dostępne dla wszystkich węzłów. Ułatwia również sporządzanie kopii zapasowych.

W prezentowanej konfiguracji dzielony zasób jest montowany automatycznie przez węzły. W związku z tym zachodzi konieczność, aby węzły *cluster02*, *cluster03*, ..., *cluster12* były uruchamiane po pełnym uruchomieniu się węzła *cluster01*.

Konfiguracja dzielonego zasobu zezwala na zapis i odczyt wszystkim węzłom.

5.2.3. Komunikacja sieciowa

Każdy z węzłów posiada serwer i klienta Ssh (*Secure Shell*). Konfiguracja ssh pozwala na łączenie się każdego węzła z każdym bez podawania hasła. Wykorzystano w tym celu klucze publiczne. Każdy z węzłów ma użytkownika o nazwie 'cluster' należącego do grupy 'cluster'. Użytkownik 'cluster' na każdym węźle loguje się do systemu podając takie samo hasło. Rozwiązanie takie upraszcza konfigurację usługi ssh.

Usługa ssh jest używana do uruchamiania procesów na węzłach klastra. W przypadku obliczeń rozproszonych wadą ssh jest szyfrowanie transmisji. Prowadzi to niewątpliwie do opóźnień obliczeń. Jednakże w przypadku węzłów połączonych za pomocą zewnętrznej sieci użycie transmisji jawnej pogorszyłoby drastycznie bezpieczeństwo całego systemu.

5.2.4. Odzworowanie nazw węzłów

W implementacji klastra nie został zastosowany serwer konfiguracyjny DHCP i DNS. Węzły mają statycznie przypisane adresy IP.

Odzworowanie nazw symbolicznych (*cluster01*, *cluster02*, ..., *cluster12*) na adresy liczbowe zostało zrealizowane poprzez odpowiednie wpisy w pliku konfiguracyjnym */etc/hosts*.

5.2.5. Biblioteka Mpich

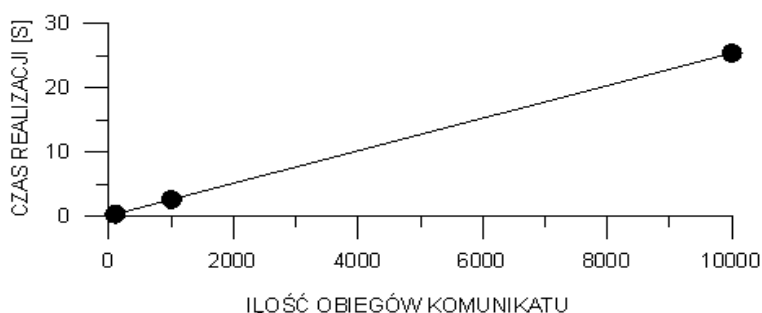
Na każdym węźle lokalnie został zainstalowany biblioteka *Mpich 2*. Jest to implementacja MPI-2 rozwijana przez *Argonne National Laboratory*. Oprócz implementacji MPI-2 pakiet jest wyposażony w środowisko uruchomieniowe i skrypty ułatwiające kompilację oprogramowania.

5.2.6. Wydajność komunikacji

Szybkość komunikacji została sprawdzona za pomocą programu narzędziowego *mpdringtest*. Wyniki zestawia tabela 5.1 oraz rysunek 5.3. W zakresie pomiarowym nie stwierdzono ponadliniowych opóźnień.

Tabela 5.1. Wyniki testu szybkości komunikacji węzłów klastra.

<i>Ilość obiegów</i>	<i>Czas wykonania [s]</i>
100	0.29184103121
1 000	2.56493902206
10 000	25.4008171556



Rys 5.3: Szybkość komunikacji węzłów klastra w zależności od ilości komunikatów.

5.2.7. Opis zbudowanego systemu wieloprocesorowego

Odwołując się do taksonomii Flynna zbudowany klaster należy opisać jako system MIMD. Ze względu na organizację pamięci jest to system z pamięcią rozproszoną. W klasyfikacji Erlangera system otrzyma opis: (12, 1, 32). Przy czym koprocesor numeryczny nie został potraktowany jako odrębna jednostka obliczeniowa.

6. Problem n – ciał

Problem n – ciał (*n – body problem*) zadziwiająco często występuje w zagadnieniach fizycznych a przez to w technice. Mamy z nim do czynienia wówczas, gdy zjawisko fizyczne może być modelowane dyskretnym zbiorem cząstek, które wzajemnie oddziałują na siebie. Koncepcja takiego postępowania jest atrakcyjna ze względu na swoją prostotę. Jednak prostota ta jest zrównoważona przez czasochłonność obliczeń, tj. konieczność rekalkulacji wzajemnych oddziaływań wszystkich elementów na każdym kroku symulacji. Zatem naturalnym sposobem wybrnięcia z tego problemu jest zastosowanie do obliczeń klastra komputerowego.

6.1. Uogólniony problem n-ciał

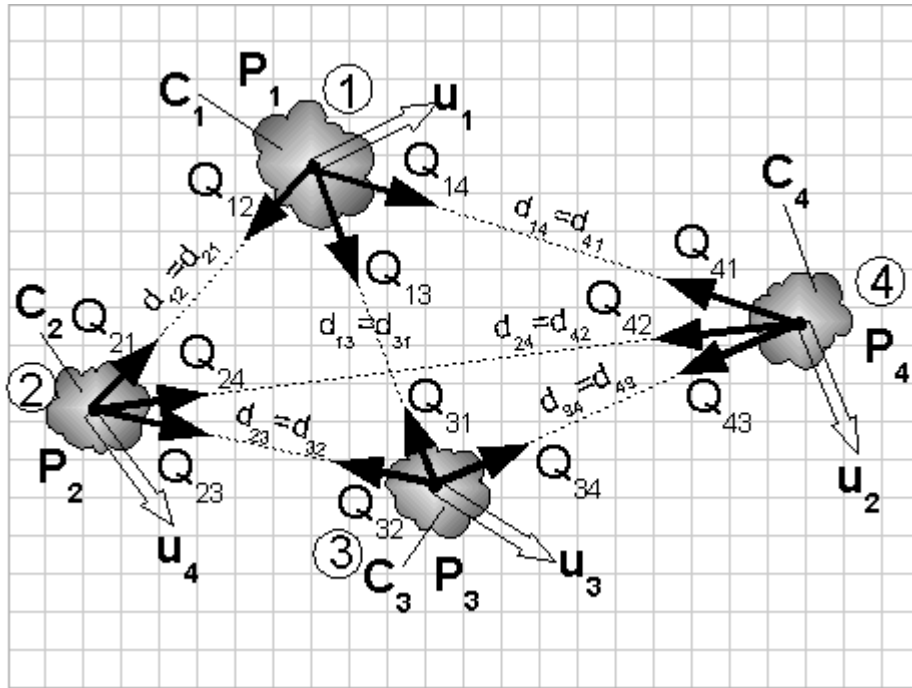
6.1.1. Sformułowanie problemu

W celu postawienia uogólnionego problemu n-ciał [6] należy (rys. 6.1):

- obrać przestrzeń metryczną z metryką d
- wzbogacić obraną przestrzeń metryczną o czas t
- obrać zbiór obiektów i określić ich początkowe parametry
- obrać funkcje oddziaływania pomiędzy obiektami i oraz j $Q_{ij}=Q(C_i, C_j, d_{ij})$
- obrać funkcję prędkości $u=u(Q, u, P, \dots)$

Każdy z rozpatrywanych obiektów opisany jest następującymi charakterystykami:

- położeniem w przestrzeni $P_i=(x_1, x_2, x_3, \dots)$
- natężeniem obranej cechy C_i
- prędkością $u_i=\frac{d P_i}{dt}$



Rys 6.1: Położenie obiektów w chwili czasowej t_0 . Wzajemne oddziaływania obiektów. Oznaczenia: P_i – położenie obiektu i ; C_i – natężenie cechy dla obiektu i ; u_i - prędkość obiektu i ; d_{ij} – odległość pomiędzy obiektami i oraz j ; Q_{ij} – oddziaływanie obiektu j na obiekt i . (Opracowanie własne)

Obiekty stale przemieszczają się w przestrzeni z prędkościami V_i . W każdym położeniu oddziałują wzajemnie na siebie. Pod wpływem wzajemnych oddziaływań zmieniają się prędkości obiektów.

W dowolnej chwili t_0 na obiekt i działa uogólniona siła od obiektu j równa Q_{ij} .

Zatem wypadkowe oddziaływanie od wszystkich obiektów na obiekt i wynosi Q_i :

$$Q_i = \sum_{j=1}^n Q_{i,j} \quad (5.1)$$

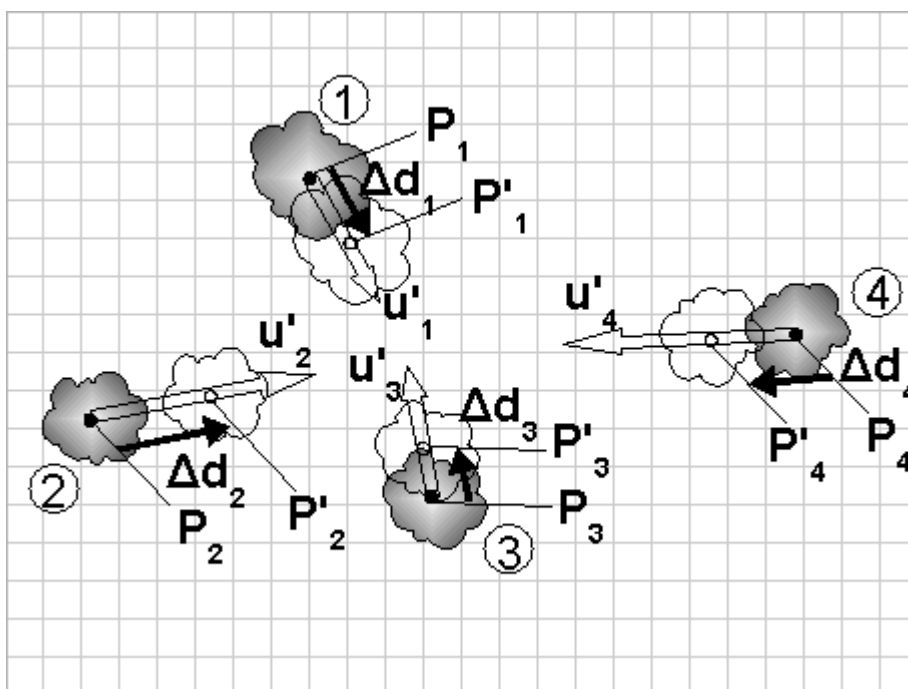
Oddziaływanie to zmienia prędkość obiektu i w chwili czasowej t_0 :

$$u_i^{t_0} = u(Q_i, u_i^{t_0 - \Delta t}, \dots) \quad (5.2)$$

Następnie obiekt zmienia swoje położenie (rys. 6.2):

$$P_i^{t_0 + \Delta t} = P_i^{t_0} + \Delta t \cdot u_i^{t_0} \quad (5.3)$$

Obiekty będące w innych położeniach oddziałują na siebie innymi siłami. Zatem cykl obliczeń należy powtórzyć.



Rys 6.2: Zmiana położenia obiektów dla czasu $t_0 + \Delta t$. Oznaczenia: P'_i – nowe położenie obiektu; Δd_i – przemieszczenie obiektu i . (Opracowanie własne)

Należy określić położenie obiektów w dowolnej chwili czasowej.

Okazuje się, że tak sformułowany problem nie ma w ogólności analitycznego rozwiązania. Położenie obiektów może być określone jedynie poprzez bezpośrednią symulację numeryczną.

Wyróżnia się 3 główne grupy metod rozwiązywania problemu n – ciał:

- metoda cząstka – cząstka
- metoda cząstka – siatka
- metoda cząstka – cząstka – cząstka – siatka.

6.1.2. Algorytm cząstka – cząstka (*particle – particle PP*)

Algorytm cząstka – cząstka [6] bazuje na bezpośrednim zliczaniu oddziaływań wg. założonej funkcji. W związku z tym dla każdego z n obiektów należy wykonać n obliczeń. Zatem problem charakteryzuje się złożonością obliczeniową rzędu $O(n^2)$.

Złożoność obliczeniowa tego rzędu prowadzi do długich czasów symulacji trudnych do zaakceptowania w praktycznych obliczeniach inżynierskich. Zatem algorytm ten, pomimo że jest najprostszy i najbardziej dokładny jest stosowany

najrzadziej.

6.1.3. Algorytm cząstka – siatka (*particle – mesh PM*)

Algorytm cząstka – siatka [6] jest metodą najmniej dokładną, ale najszybszą. Wymaga ona wprowadzenia siatki w obrębie domeny obliczeniowej. Następnie w oczkach tej siatki wyznacza się potencjalne pole oddziaływań wywołane poprzez istniejące w węzle siatki cząstki. Wiąże się to z koniecznością rozwiązania równanie Poissona. Na podstawie znanego pola potencjalnego (znanego w wybranych punktach siatki) określa się oddziaływanie poprzez interpolację w punktach, w których znajdują się cząstki.

Algorytm składa się z 3 głównych kroków:

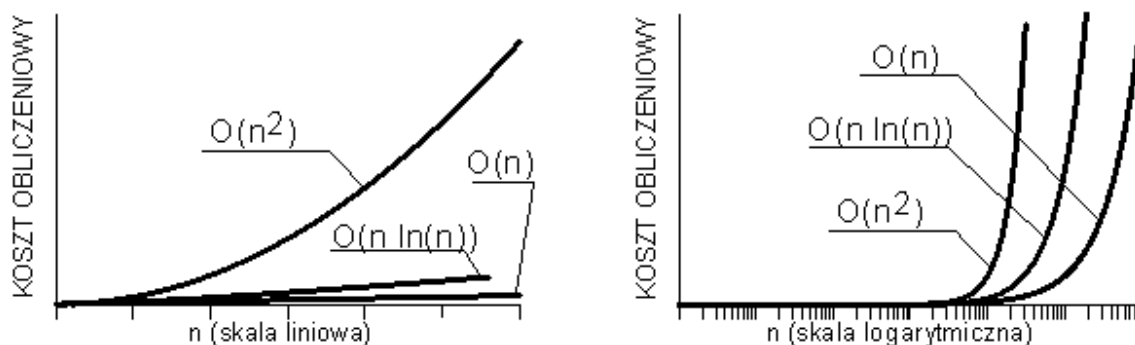
- przypisanie mocy cząstek do siatki (na podstawie położenia cząstek)
- rozwiązanie równanie Poissona na przyjętej siatce
- wyznaczenie sił dla cząstek na podstawie pola potencjalnego

Złożoności obliczeniowe kroku pierwszego i ostatniego są rzędu $O(n_1)$, gdzie n_1 oznacza liczbę cząstek. Krok drugi zależy liniowo od ilości punktów siatki, w których rozwiązuje się równie Poissona i jego złożoność obliczeniowa jest rzędu $O(n_2)$, gdzie n_2 oznacza ilość węzłów siatki. Zatem problem charakteryzuje się złożonością obliczeniową rzędu $O(n)$.

6.1.4. Algorytm cząstka – cząstka – cząstka – siatka (*particle – particle – particle – mesh P³M*)

Algorytm cząstka – cząstka – cząstka – siatka [6] jest połączeniem dwóch poprzednich metod. Metoda ta wykorzystuje zarówno siatkę jak i bazuje na bezpośrednim zliczaniu oddziaływań. Podobnie jak w metodzie cząstka-siatka wyznaczane jest pole potencjalne w całym obszarze domeny obliczeniowej. Następnie dla każdej z n cząstek wyróżnia się bliższe i dalsze sąsiedztwo. Cząstki pozostające w bliższym sąsiedztwie oddziałują bezpośrednio na rozważaną cząstkę (zastosowanie metody cząstka – cząstka). Oddziaływania cząstek z dalszego sąsiedztwa zliczane są pośrednio poprzez uwzględnienie potencjału (metoda cząstka – siatka). Koszt obliczeniowy zależy od proporcji wielkości sąsiedztwa bliższego do sąsiedztwa dalszego. Może zmieniać się od $O(n)$ do $O(n^2)$. Przy optymalnym doborze wielkości sąsiedztw koszt obliczeniowy wynosi $O(n \cdot \ln(n))$.

Na rysunku 6.3. zostały przedstawione za pomocą wykresów omawiane koszty obliczeniowe.



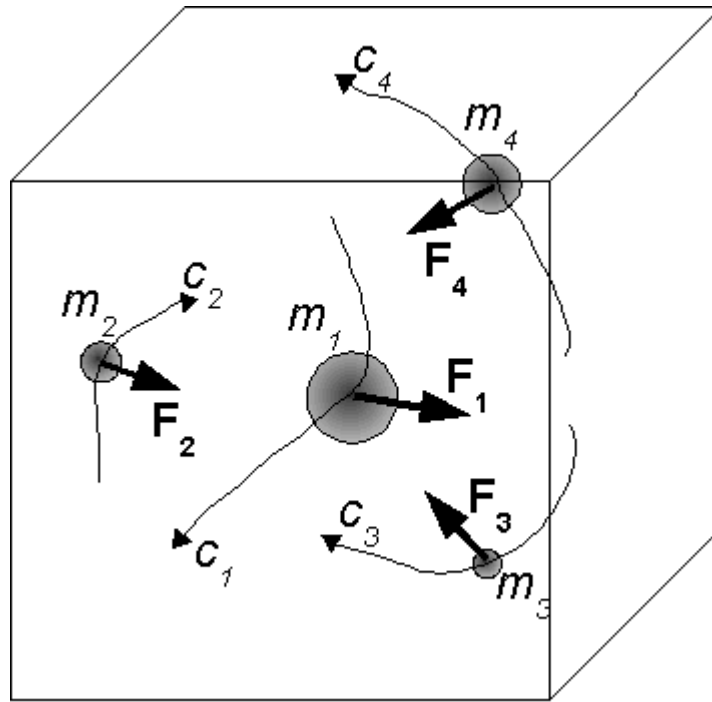
Rys 6.3: Porównanie kosztów obliczeniowych. (Opracowanie własne)

6.2. Przykłady problemów technicznych, w których występuje problem n – ciał

Duża grupa zagadnień technicznych może być rozwiązana drogą bezpośrednich symulacji komputerowych. W wielu takich symulacjach występuje problem n – ciał. Poniżej zostały przedstawione 3 takie problemy.

6.2.1. Grawitacja

Klasyczny problem n -ciał [6] został sformułowany dla ciał posiadających masę i oddziaływających na siebie siłami przyciągania grawitacyjnego. Ciała rozpatrywane są w trójwymiarowej przestrzeni euklidesowej (rys. 6.4)



Rys 6.4: Masy w euklidesowej przestrzeni trójwymiarowej; Oznaczenia: m_i – masa, c_i – trajektoria masy m_i , F_i - siła wypadkowa działająca na masę m_i . (Opracowanie własne)

Poszukujemy trajektorii dowolnie wybranego ciała: $c_i(t)=?$

Siła grawitacji działająca na ciało i od ciała j określa prawo powszechnego ciążenia:

$$F_{ij}=F_{ji}=\gamma \cdot m_i \cdot m_j \frac{c_j(t)-c_i(t)}{\|c_j(t)-c_i(t)\|^3} \quad (5.4)$$

Trajektorię taką określa druga zasada dynamiki Newtona:

$$F_i=\frac{d^2 c_i(t)}{dt^2} \cdot m_i \quad (5.5)$$

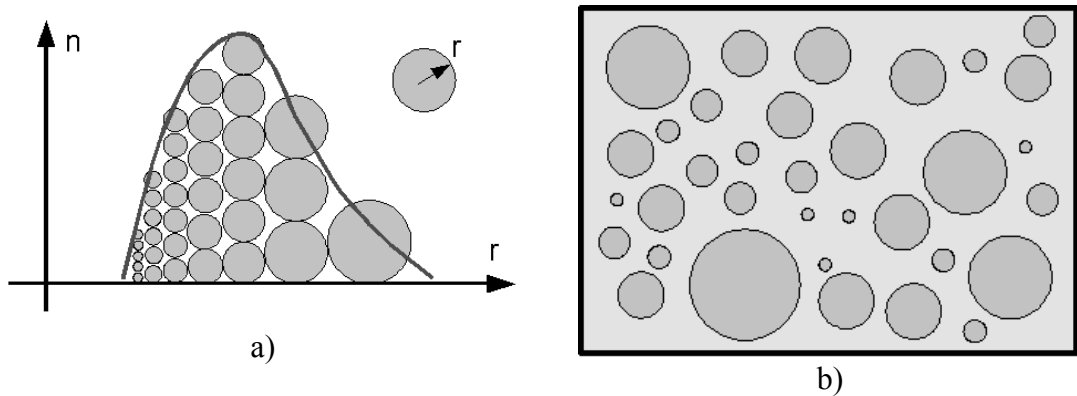
Zatem otrzymujemy równanie trajektorii

$$\frac{d^2 c_i(t)}{dt^2}=\gamma \sum_{j=1}^n m_j \frac{c_j(t)-c_i(t)}{\|c_j(t)-c_i(t)\|^3} \quad (5.6)$$

gdzie: γ oznacza stałą grawitacji.

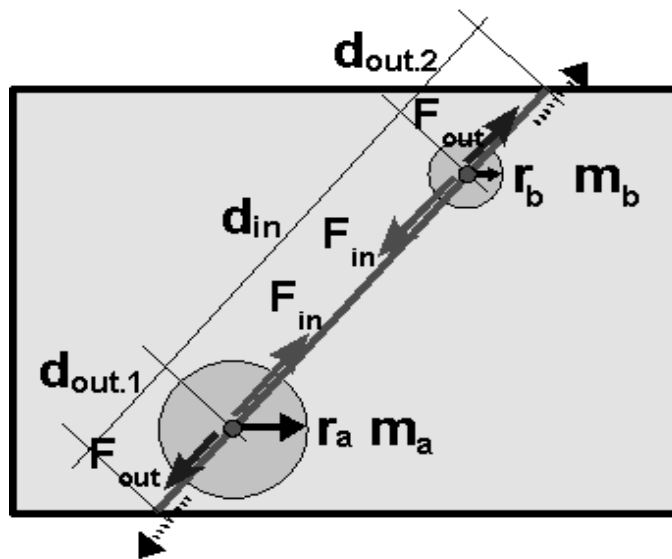
6.2.2. Tworzenie korelacji przestrzennych

Opisywane w tym rozdziale zagadnienie ma zastosowanie do tworzenia sztucznych próbek skał o pseudolosowej strukturze (rys 6.5). Próbka taka składa się z jednorodnego ośrodka i domieszek (ziaren). Narzucamy rozkład domieszek (rys. 6.5 a), tzn. krzywą przesiewową. Ziarna mogą być rozmieszczone w próbce na nieskończona ilość sposobów. Najprostszym sposobem na sklasyfikowanie takich rozkładów jest zastosowanie techniki „funkcji układającej”.



Rys 6.5: a) Zadany rozkład ziaren b) wygenerowana próbka. (Opracowanie własne)

Zakładamy, że przestrzeń próbki nie ma granic, tj. przypomina sferę. Następnie założone zostaje najczęściej sztuczne oddziaływanie pomiędzy ziarnami. Oddziaływanie to następuje wzdłuż linii wewnętrznej i zewnętrznej (rys. 6.6).



Rys 6.6: Wzajemne oddziaływanie ziaren. Oznaczenia: r_a , r_b – promienie ziaren; m_a , m_b – umowne masy ziaren; d_{in} – odległość wewnętrzna ziaren; d_{out} – odległość zewnętrzna ziaren, F – oddziaływanie.
(Opracowanie własne)

Funkcja opisująca oddziaływanie może być dobrana w dowolny sposób. Poniżej przedstawione zostały 3 przykłady:

$$F_1(f, m_a, m_b) = \alpha_1 \cdot f \cdot m_a \cdot m_b \quad (5.7)$$

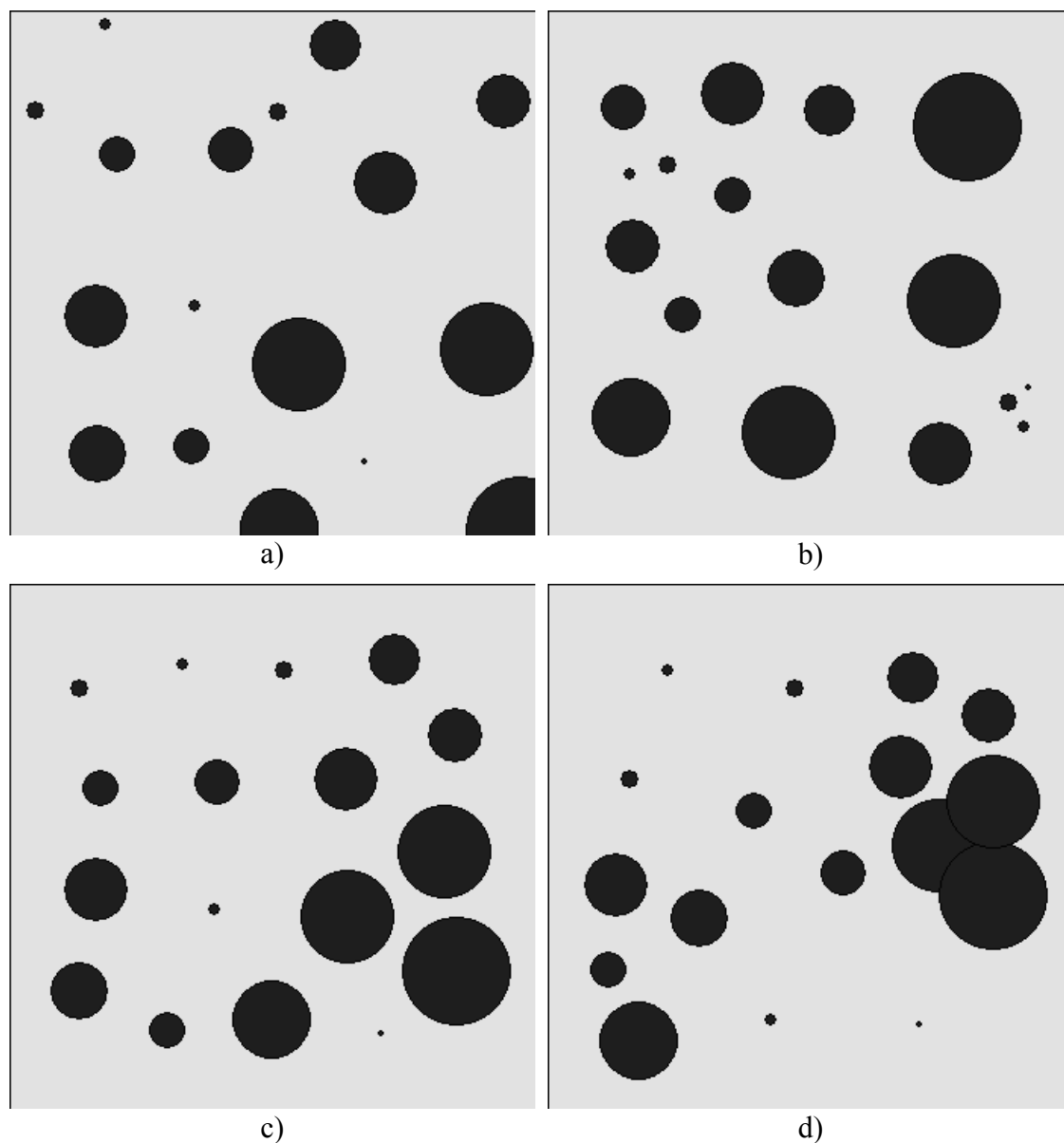
$$F_2(f, m_a, m_b) = \alpha_2 \cdot \frac{f}{m_a + m_b} \quad (5.8)$$

$$F_3(f, m_a, m_b) = \alpha_2 \cdot \frac{f}{m_a \cdot m_b} \quad (5.9)$$

gdzie: $f = 1 - \frac{r_a + r_b}{d}$; α - współczynnik; d – odległość między ziarnami.

W przytoczonym przykładzie wartość oddziaływania została przyjęta jako równa przemieszczeniu. Symulacja była prowadzona do momentu ustalenia się stanu równowagi. Wyniki symulacji przedstawia rysunek 6.7. Analiza wyników pokazuje, że funkcja F_1 prowadzi do równomiernego rozłożenia ziaren w obszarze próbki. Funkcja F_2 preferuje konglomeraty dużych drobin. Zaś funkcja F_3 powoduje tworzenie gron złożonych z ziaren o porównywalnej wielkości.

Wygenerowane próbki są następnie poddawane dalszym analizom. Okazuje się, że różne próbki wygenerowane za pomocą tej samej funkcji układającej wykazują podobieństwa.



Rys 6.7: a) Ułożenie początkowe (losowe). b) Efekt działania funkcji F_1 c) Efekt działania funkcji F_2 d) Efekt działania funkcji F_3 . (Opracowanie własne)

6.2.3. Metoda wirów dyskretnych

W metodzie wirów dyskretnych zagadnienie wzajemnego oddziaływania elementów wirowych odgrywa kluczowe znaczenie.

W prezentowanej pracy zagadnienie to zostało wybrane jako temat analiz. W związku z tym metodzie wirów dyskretnych został poświęcony oddzielny rozdział.

7. Metoda wirów dyskretnych

Metoda wirów dyskretnych (*Discrete Vortex Method DVM*) [4] [5] jest jedną z metod numerycznych służącą do symulacji przepływów turbulentnych. Narodziny DVM datuje się na lata '30 XX wieku. Jednakże szybki rozwój metod wirowych rozpoczął się w latach '80 XX wieku i był poprzedzony rozwojem technik komputerowych. Od tego czasu metoda ta była z powodzeniem stosowana głównie w zagadnieniach mechaniki płynów, fizyki plazmy, dynamiki cząstek i przenikania cieczy w ośrodkach porowatych.

DVM jest metodą numeryczną służącą rozwiązywaniu równania Naviera-Stokes (N-S). Bazuje ona na lagranżowskim modelu śledzenia cząstek. Zaliczana jest do metod bezpośredniej symulacji cyfrowej, gdyż rozwiązuje równanie N-S w postaci wirowej metodami bezpośredniej symulacji zjawisk fizycznych, a nie poprzez zastosowanie numerycznych metod siatkowych jak np.: metoda różnic skończonych, objętości kontrolnych czy elementów skończonych. Ze względu na samoadaptacyjne właściwości tej metody nie są konieczne dodatkowe równania opisujące uproszczone modele turbulencji. Innymi zaletami tej metody jest jej stabilność numeryczna i brak ograniczeń związana z geometrią zadania. Mniejsza jest też liczba niewiadomych, tj. parametrów śledzenia cząstek lub wirów, w stosunku do metod RANS (*Reynolds Average Navier-Stokes*) lub LES (*Large Eddy Simulation*). Wadą tej metody jest czasochłonność algorytmu odtwarzającego ciągle pole prędkości na podstawie wirów dyskretnych. Konieczność wykonania dużej ilości obliczeń uzasadnia zastosowanie klastra komputerowego.

7.1. Podstawowe równania

Równanie Naviera-Stokesa (N-S) dla płynu nieściśliwego dla zagadnienia dwuwymiarowego [7]:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \nabla) \mathbf{u} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} \quad (6.1)$$

gdzie: $\mathbf{u} = u_x \mathbf{e}_x + u_y \mathbf{e}_y$ - pole prędkości; p - pole ciśnienia; ν - lepkość płynu; t - czas.

Równanie N-S poddajemy dekompozycji obliczając rotację wektora \mathbf{u} i otrzymujemy równanie transportu wirów:

$$\frac{\partial \omega}{\partial t} + (\mathbf{u} \nabla) \omega = \nu \nabla^2 \omega \quad (6.2)$$

gdzie: $\omega = \nabla \times \mathbf{u} = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}$ oznacza (dla przypadku 2D) skalarne pole wirowości prędkości.

Równanie (6.2) składa się z dwóch części: adwekcyjnej (6.3) i dyfuzyjnej (6.4).

$$\frac{\partial \omega}{\partial t} + (\mathbf{u} \nabla) \omega = 0 \quad (6.3)$$

$$\frac{\partial \omega}{\partial t} = \nu \nabla^2 \omega \quad (6.4)$$

Równanie adwekcji oznacza znikanie pochodnej materialnej pola wirowości. Rozwiązywane jest ono zwykle metodą przyrostową we współrzędnych Lagrangea dla pojedynczych wirów. Rozdzielenie równania (6.2) w przedstawiony powyżej sposób jest określane w literaturze mianem *split algorithm*. Przepływ może być traktowany jako dwa niezależnie i równocześnie zachodzące zjawiska: adwekcji i dyfuzji. Oba równania są podstawą do dyskretyzacji pola przepływu i symulacji zmian tego pola w czasie.

7.2. Kinematyka wirowa

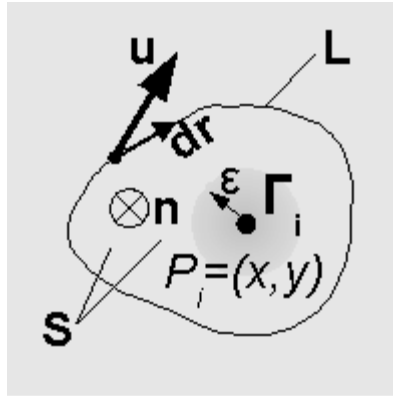
7.2.1. Wir dyskretny

Podstawowym obiektem DVM jest dyskretny element wirowy [4] [5]. Wir dyskretny charakteryzuje się:

- położeniem $\mathbf{P}_i = (x, y)$
- mocą Γ_i
- jądrem Biota-Savarta \mathbf{K}_ϵ

Moc wiru to wartość cyrkulacji pola prędkości wokół krzywej zamkniętej otaczającej powierzchnię S z określoną normalną \mathbf{n} (rys 7.1) :

$$\Gamma = \oint_L \mathbf{u} \cdot d\mathbf{r} = \iint_S \omega \cdot \mathbf{n} dS \quad (6.5)$$



Rys 7.1: Element wirowy utworzony w punkcie $P_i = (x, y)$ poprzez obliczenie cyrkulacji prędkości po krzywej zamkniętej L . (Opracowanie własne)

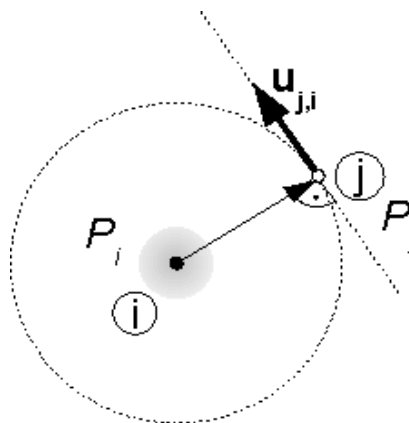
Elementy wirowe tworzone są na początku symulacji dla zadanego pola prędkości. Następnie przeprowadzana jest symulacja. Z elementem wirowym nie jest związana masa i jest to wyłącznie opis kinematyczny ruchu płynu.

7.2.2. Wzajemne oddziaływanie wirów

Wir znajdujący się w punkcie i indukuje w punkcie j prędkość zgodnie z prawem Biota-Savarta (rys. 7.2) [4] [5]:

$$\mathbf{u}_{j,i} = \Gamma_i \cdot \mathbf{K}_\epsilon(\mathbf{P}_j - \mathbf{P}_i) \quad (6.6)$$

, gdzie $\mathbf{P} = (x, y)$



Rys 7.2: Indukcja prędkości w punkcie j przez wir znajdujący się w punkcie i . (Opracowanie własne)

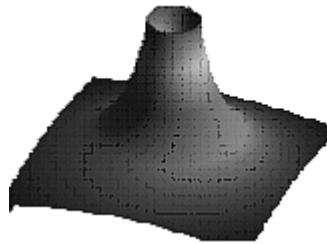
Zatem prędkość w punkcie \mathbf{j} , będąca wynikiem łącznego oddziaływania wszystkich wirów, wyraża się wzorem:

$$\mathbf{u}_j = \sum_{i=1}^n \mathbf{u}_{j,i} = \sum_{i=1}^n \Gamma_i \cdot \mathbf{K}_\epsilon(\mathbf{P}_j - \mathbf{P}_i) \quad (6.7)$$

Najprostszym jądrem Biota-Savarta jest jądro 2 rzędu:

$$\mathbf{K}_\epsilon = \begin{cases} \frac{(-y, x)}{2 \pi \epsilon} & r \leq \epsilon \\ \frac{(-y, x)}{2 \pi r^2} & r > \epsilon \end{cases} \quad (6.8)$$

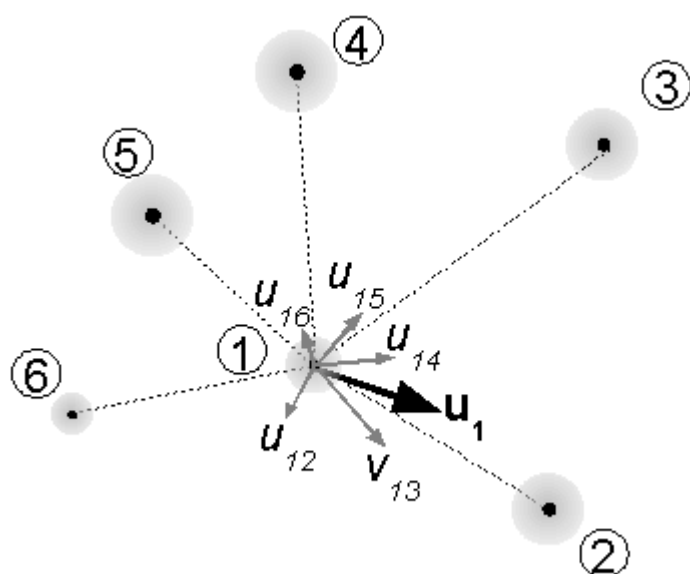
Typowy kształt modułu jądra jest przedstawiony na rysunku 7.3.



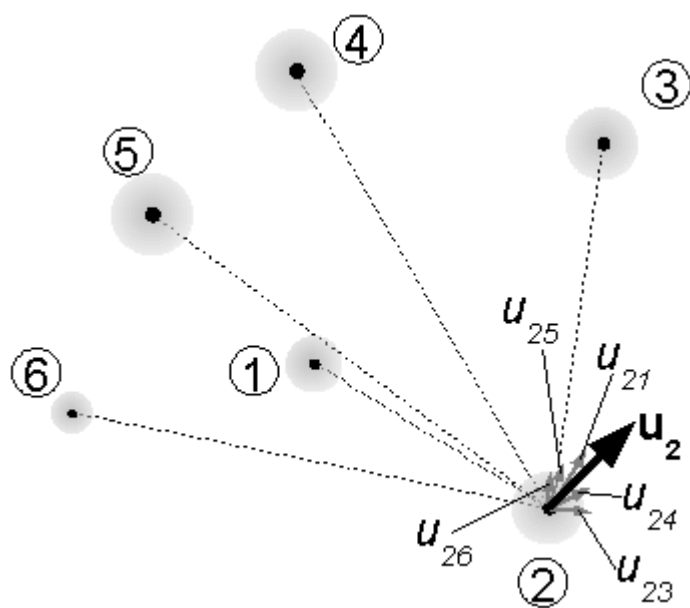
Rys 7.3: Typowy kształt modułu jądra Biota-Savarta. (Opracowanie własne)

7.2.3. Kinematyka wirowa

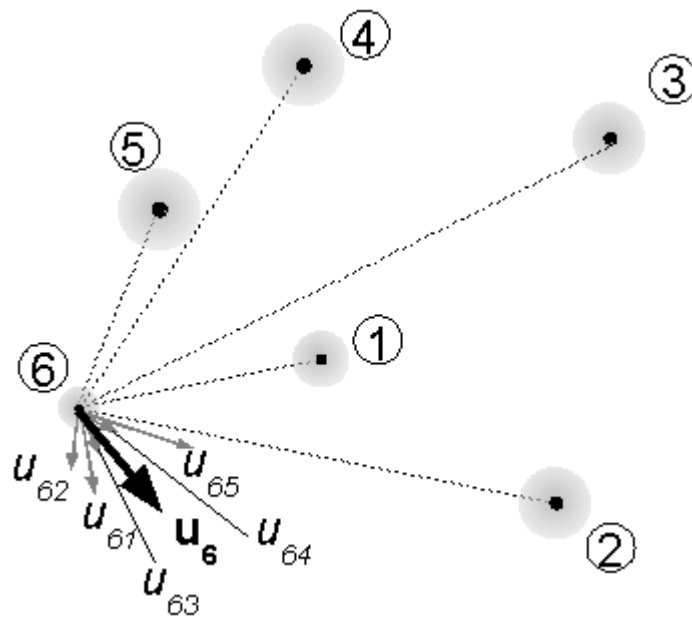
Rozwiązanie równania adwekcyjnego w sposób bezpośredni polega na wyznaczeniu prędkości w punkcie położenia każdego wiru i wykonania przesunięcia wirów. Typowa symulacja inżynierska składa się z kilkudziesięciu tysięcy takich kroków. Sposób przeprowadzenia 1 kroku symulacji obrazują rysunki od 7.4 do 7.7.



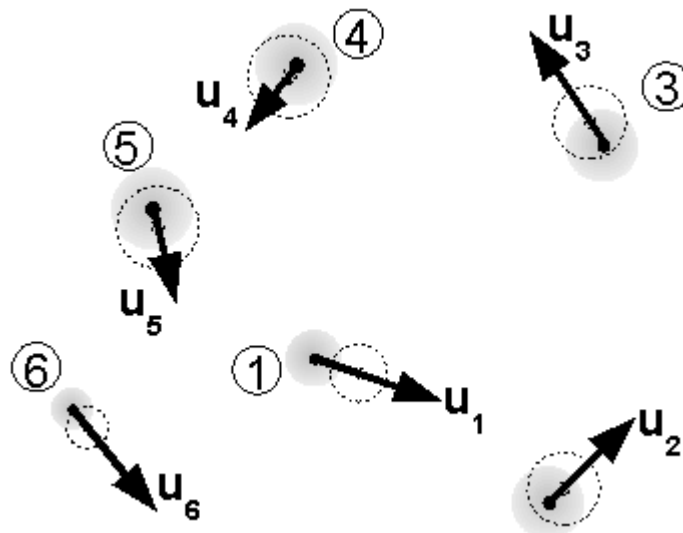
Rys 7.4: Prędkości indukowane w punkcie nr 1. (Opracowanie własne)



Rys 7.5: Prędkości indukowane w punkcie nr 2. (Opracowanie własne)



Rys 7.6: Prędkości indukowane w punkcie nr 6. (Opracowanie własne)



Rys 7.7: Przesunięcie się wirów pod wpływem prędkości wypadkowych. (Opracowanie własne)

7.2.4. Szczególne przypadki

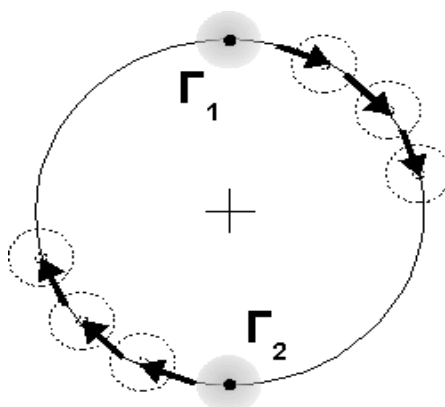
W przypadku przepływów turbulentnych położenie elementów wirowych jest szybkozmienne i chaotyczne. Występujące struktury makroskopowe (wiry

makroskopowe) są również trudne do sklasyfikowania. Istnieją jednak specjalne przypadki dla których łatwo przewidzieć kolejne stadia rozwoju. Należą do nich m.in. para wirów i linia wirowa.

Para wirów to dwa elementy wirowe o jednakowym natężeniu wirowości umieszczone względem siebie w ustalonej odległości. Trajektorią pary wirowej jest okrąg, którego średnicę w każdej chwili wyznacza para wirów (rys. 7.8).

Linia wirowa to struktura złożona z dowolnej ilości elementów wirowych o jednakowym natężeniu wirowości. Elementy w momencie rozpoczęcia symulacji tworzą prostą linię a odległości pomiędzy nimi są równe (rys. 7.9). Linia wirowa w trakcie symulacji dokonuje obrotu wokół swojego środka wraz z zawijaniem końców (rys. 7.10).

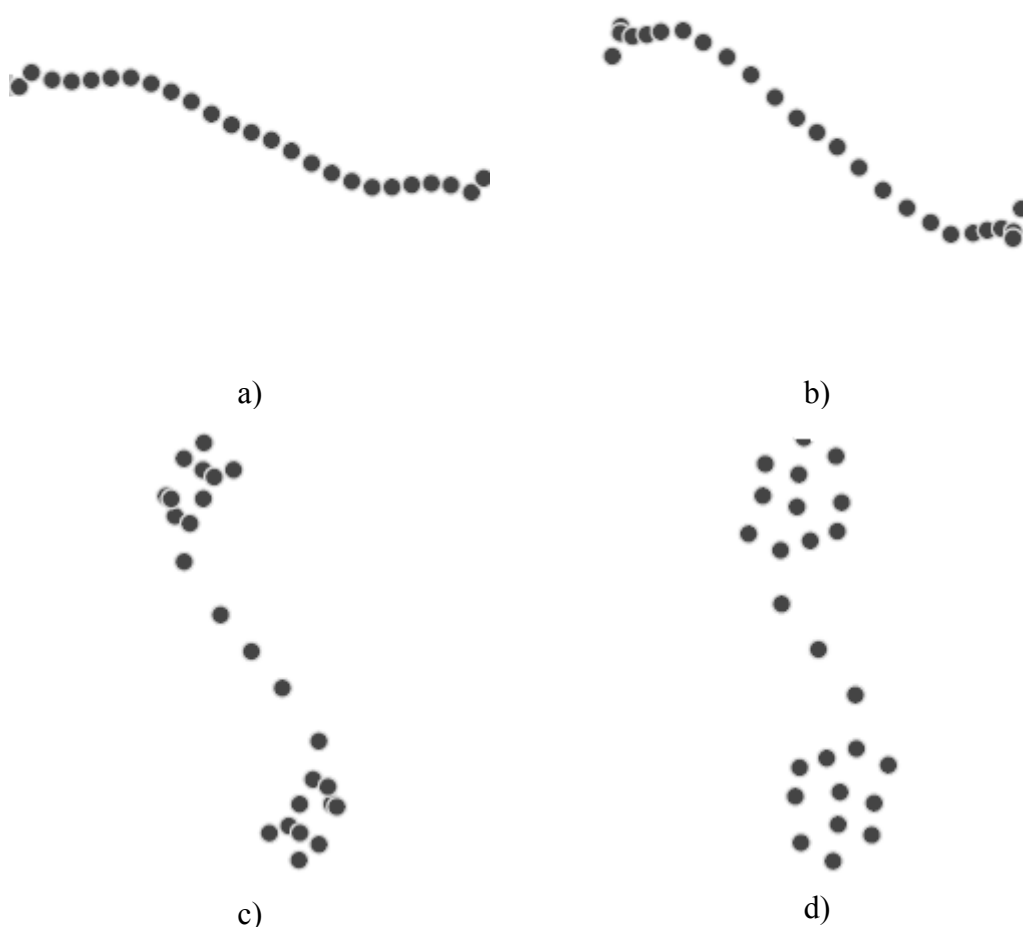
Oba powyższe przypadki szczególne posłużyły do przetestowania autorskich programów komputerowych.



Rys 7.8: Trajektoria pary elementów wirowych o jednakowej wirowości. (Opracowanie własne)



Rys 7.9: Linia wirowa. (Opracowanie własne)



Rys 7.10: Ewolucja linii wirowej: a, b, c, d – kolejne stadia. (Opracowanie własne)

7.2.5. Szybki algorytm

Jednym sposobem rozwiązania problemu czasu obliczeń jest rezygnacja z wzajemnych oddziaływań między wszystkimi elementami wirowymi [4] [7]. W tym celu domena obliczeniowa, tj. obszar symulacji, zostaje podzielona na subdomeny (cele) jak na rysunku 7.11. Każda subdomena posiada charakterystyki mówiące jak wszystkie wiry należące do danej subdomeny oddziałują (łącznie) na pojedynczy wir znajdujący się na zewnątrz tej domeny. W ten sposób wyeliminowana została konieczność obliczania bezpośrednich interakcji między wszystkimi wirami biorącymi udział w symulacji. Ciągłe jednak zachodzi potrzeba wyliczania bezpośrednich interakcji pomiędzy wirami należącymi do tej samej subdomeny.

Przedstawiony algorytm należy do typu cząstka – cząstka – cząstka – siatka. W przypadku metody wirów dyskretnych algorytm cząstka – siatka nie jest zalecany ze względu na brak możliwości ujęcia zjawisk drobnoskalowych.

Parametry charakteryzujące subdomenę to:

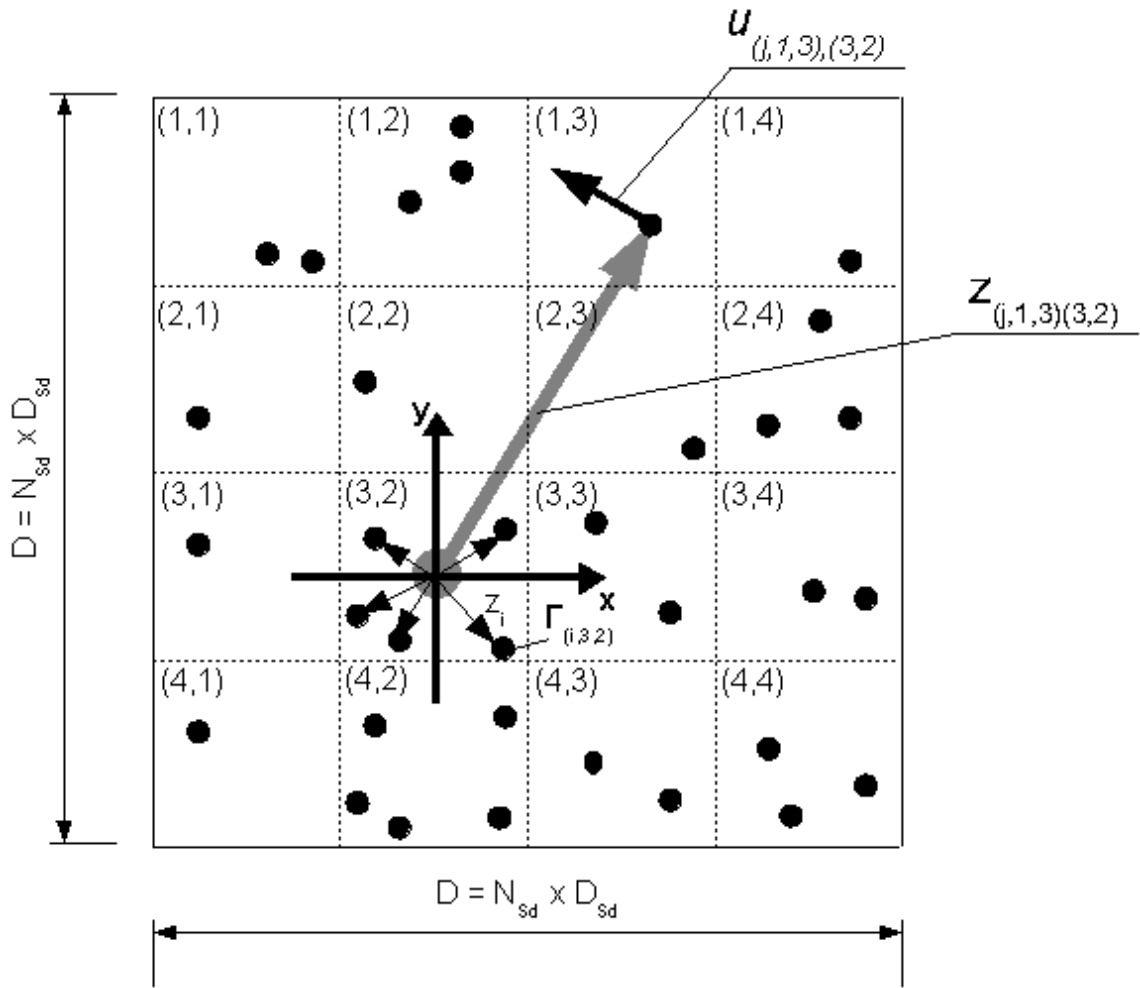
- położenie środka domeny (początek jej lokalnego układu współrzędnych)
- suma wirowości elementów do niej należących
- momenty wirowe względem początku lokalnego układu współrzędnych

Suma wirowości elementów w subdomenie (k,l) :

$$\Gamma_{0,(k,l)} = \sum_{i=1}^m \Gamma_i \quad (6.9)$$

gdzie:

m – aktualna ilość elementów wirowych znajdujących się w subdomenie (k,l)



Rys 7.11: Podział domeny symulacji na $N_{sub}=N_{sd} \times N_{sd}$ subdomen; $\mathbf{u}_{(j,1,3),(3,2)}$ oznacza prędkość wyindukowaną w punkcie j subdomeny (1,3) od wszystkich wirów znajdujących się w subdomenie (3,2). (Opracowanie własne)

Moment wirowy rzędu r wyraża się wzorem:

$$\mathbf{Z}_{r,(k,l)} = \mathbf{Z}_x + \mathbf{Z}_y \mathbf{i} = \frac{\sum_{i=1}^m \Gamma_{i,k,l} (\mathbf{z}_{i,k,l})^r}{\Gamma_{0,(k,l)}} \quad (6.10)$$

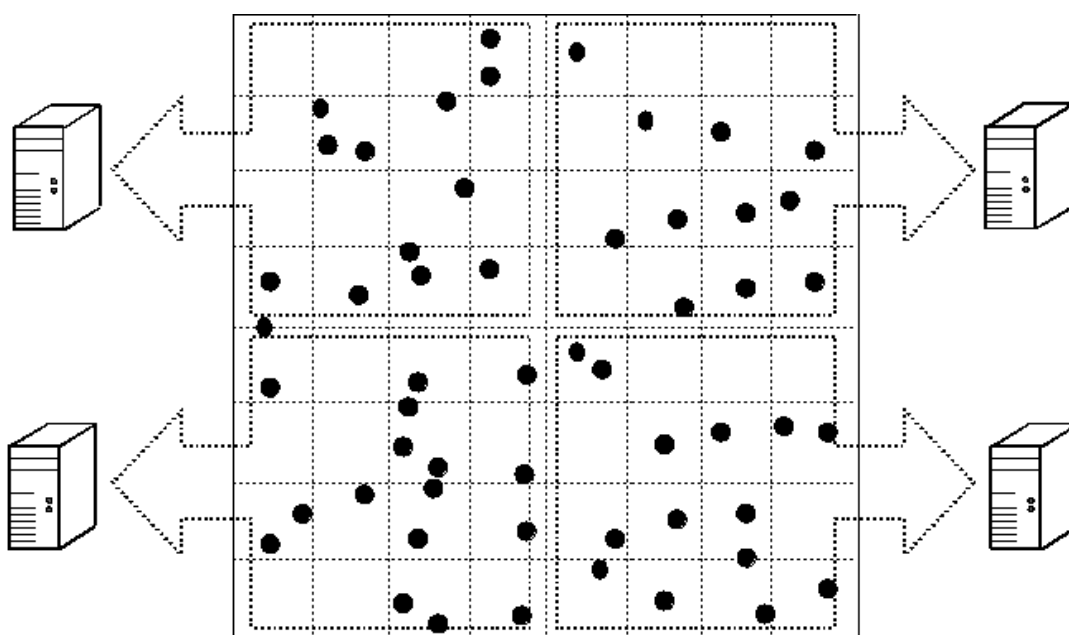
,gdzie: $\mathbf{z}_{i,k,l} = z_x + z_y \mathbf{i}$ jest odległością wiru o numerze i od środka subdomeny (k,l) , zaś \mathbf{i} nie użyte jako indeks oznacza jednostkę urojoną.

Mając charakterystyki subdomeny możliwe jest wyznaczenie prędkości wywołanej działaniem wszystkich wirów w tej subdomenie w dowolnym punkcie poza tą subdomeną (6.11).

$$u_{j,(k,l)} = u_x + u_y i = \frac{\Gamma_0}{2\pi z_{j,(k,l)}} \cdot \left[1 + \frac{Z_1}{z_{j,(k,l)}} + \frac{Z_2}{z_{j,(k,l)}^2} + \frac{Z_3}{z_{j,(k,l)}^3} + \dots \right] \quad (6.11)$$

Dokładność powyższej metody zależy od maksymalnego rzędu momentu wirowego użytego w obliczeniach. Oczywisty jest fakt, że im dalej od subdomeny obliczamy prędkość tym lepsza dokładność wyniku.

W przypadku zastosowania klastra każdy węzeł obliczeniowy obsługuje wybrany fragment domeny obliczeniowej. Każdy taki fragment jest dalej podzielony na subdomeny (rys. 7.12)



Rys 7.12: Dekompozycja geometryczna zadania. Symbol komputera oznacza oddzielny proces obliczeniowy. (Opracowanie własne)

Zachodzi tutaj przypadek dekompozycji geometrycznej. Procesy w trakcie działania programu wymieniają się informacją na temat charakterystyk opisujących subdomeny oraz przekazują sobie elementy wirowe. Zastosowanie subdomen w sposób pośredni realizuje sortowanie elementów użytych w symulacji.

W dalszej części tego paragrafu zostanie przedstawiony sposób doboru optymalnej ilości subdomen.

Zakładając, że w każdej z subdomen znajduje się NV elementów wirowych, ilość interakcji bezpośrednich (cząstka – cząstka) dla N_{sub} subdomen wynosi:

$$I_B = N_{sub} \cdot \left(\frac{NV}{N_{sub}} \right)^2 = \frac{NV^2}{N_{sub}} \quad (6.12)$$

Zaś ilość interakcji pośrednich (cząstka – siatka) jest równa:

$$I_P = N_{sub} \cdot \frac{NV}{N_{sub}} = \frac{NV}{N_{sub}} \quad (6.13)$$

Optymalny przypadek zachodzi, gdy ilość interakcji bezpośrednich jest 2 razy większa niż ilość interakcji pośrednich:

$$I_B = 2 \cdot I_P \quad (6.14)$$

Stąd obliczamy optymalną ilość subdomen:

$$N_{sub} = \sqrt{\frac{NV}{2}} \quad (6.15)$$

W przypadku równomiernego podziału domeny kwadratowej na $N_{sub} = N_{sd} \cdot N_{sd}$:

$$N_{sd} = \sqrt[4]{\frac{NV}{2}} \quad (6.16)$$

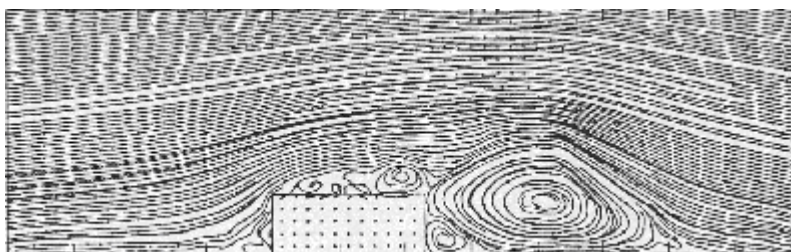
Wyprowadzone zależności zostały potwierdzone eksperymentem opisanym w dalszej części pracy.

7.3. Zastosowanie DVM

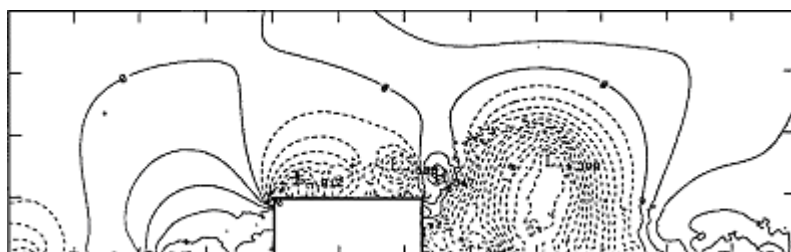
Znanych jest wiele udanych zastosowań metody wirów dyskretnych w inżynierii lądowej. Metoda ta służyła do badania opływów statycznych obiektów jak i do badania zjawisk interferencji aeroelastycznych. W praktycznych zastosowaniach ważnym zagadnieniem jest interakcja elementu wirowego z ciałem stałym. Jednak zagadnienie to nie jest tematem prezentowanej pracy i dlatego zostanie pominięte.

7.3.1. Opływ budynku

W przytoczonym przykładzie analizie został poddany rzeczywisty budynek wchodzący w skład kampusu *Texas Tech University* [7]. Obiekt ma kształt prostopadłościanu o wymiarach w rzucie poziomym $9.1\text{m} \times 13.7\text{m}$ i o wysokości 4m. Budynek znajduje się na terenie płaskim z dala od innych zabudowań. Założono średnią prędkość wiatru równą 8.6 m/s. Założono kąt natarcia wiatru 90° względem dłuższego boku obiektu. W trakcie dwuwymiarowej symulacji DVM liczba cząstek wirowych wahała się od 12 000 do 24 000. Wyniki obliczeń skonfrontowano z pomiarami. Wybrane wyniki analizy przedstawiają rysunki 7.13 i 7.14. Oba rysunki dotyczą tej samej chwili czasowej.



Rys 7.13: Linie prądu w przepływie wokół budynku Texas Tech. [7]

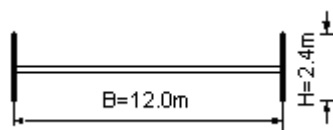


Rys 7.14: Izobary przepływu wokół budynku Texas Tech. [7]

7.3.2. Przewidywanie dynamicznej odpowiedzi konstrukcji z uwzględnieniem efektów aeroelastycznych

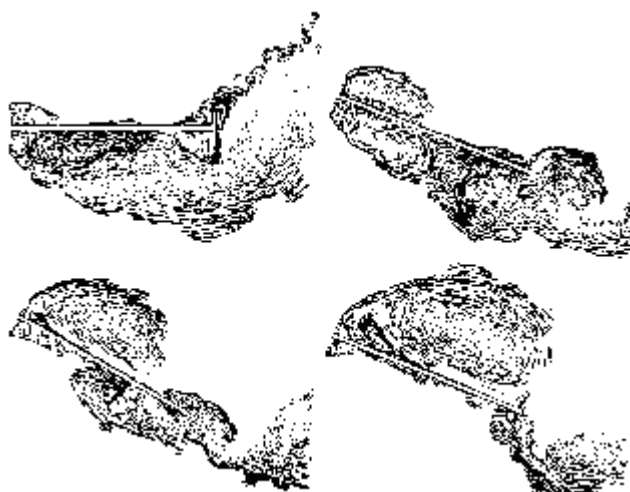
Pełne wykorzystanie możliwości DVM odnaleźć można w pracy [8], gdzie są przedstawione wyniki symulacji zachowania się pomostów znajdujących się w przepływie turbulentnym. Jednym z analizowanych mostów był *Tacoma Narrows Bridge*, który runął 7 listopada 1940 wskutek wystąpienia silnego flatteru przy prędkości wiatru wynoszącej ok. 19m/s. Oszacowano wówczas, że pomost tuż przed

katastrofą osiągnął kąt skreńcenia ok. 30° . Wymiary pomostu przedstawione są na rysunku 7.15.



Rys 7.15: Wymiary pomostu Tacoma Narrows Bridge. (Opracowanie własne)

Dwuwymiarowa symulacja DVM przeprowadzona dla pomostu pozwoliła odtworzyć jego zachowanie się w przepływie turbulentnym (rys. 7.16) i wyraźnie wykazała niebezpieczeństwo wystąpienia flatteru. Badania wykonano dla różnych prędkości wiatru mieszczących się w przedziale od 5m/s do 20m/s. Symulacja potwierdziła wystąpienie silnego flatteru przy prędkości ok. 19 m/s. Stwierdzono, że amplituda kąta obrotu osiąga wartość ok. 30° .



Rys 7.16: Flatter skretny pomostu Tacoma Narrows Bridge (wynik symulacji komputerowej) [8].

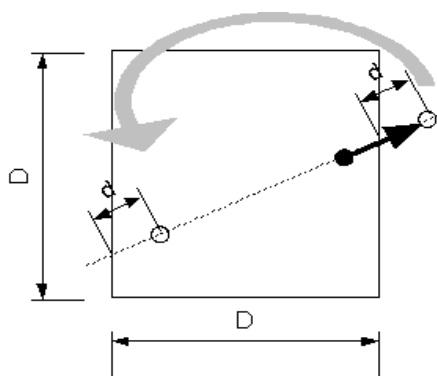
8. Implementacja algorytmu

Celem pracy było określenie możliwości zrealizowanego klastra komputerowego zastosowanego do obliczeń metodą wirów dyskretnych. Dlatego do badań został wybrany problem teoretyczny, który umożliwiał równomierne i maksymalne obciążenie obliczeniami każdego z węzłów. Przypadek ten również powodował dużą potrzebę komunikacji pomiędzy węzłami.

8.1. Założenia

W celu zapewnienia stałej liczby elementów wirowych w domenie obliczeniowej zostało przyjęte założenie domeny „sferycznej”. Każdy wir który opuszcza domenę obliczeniową jest przywracany w taki sposób, aby przeciwległy bok realizował kontynuację domeny (Rys 8.1).

W praktycznych problemach rozwiązywanych metodą wirów dyskretnych, w których występują ciała w przepływie, następuje wytwarzanie elementów wirowych na styku ciało – płyn. Najwięcej wirów pojawia się w miejscach występowania dużych gradientów ciśnienia. Zatem wiry, które opuszczają domenę obliczeniową (i mogą być pominięte w obliczeniach) są zastępowane nowymi elementami wirowymi. W analizowanym problemie powrót wiru może być również traktowany jako produkcja wiru. W związku z tym zrealizowane zadanie nie odbiega zbytnio od przypadku rzeczywistego.



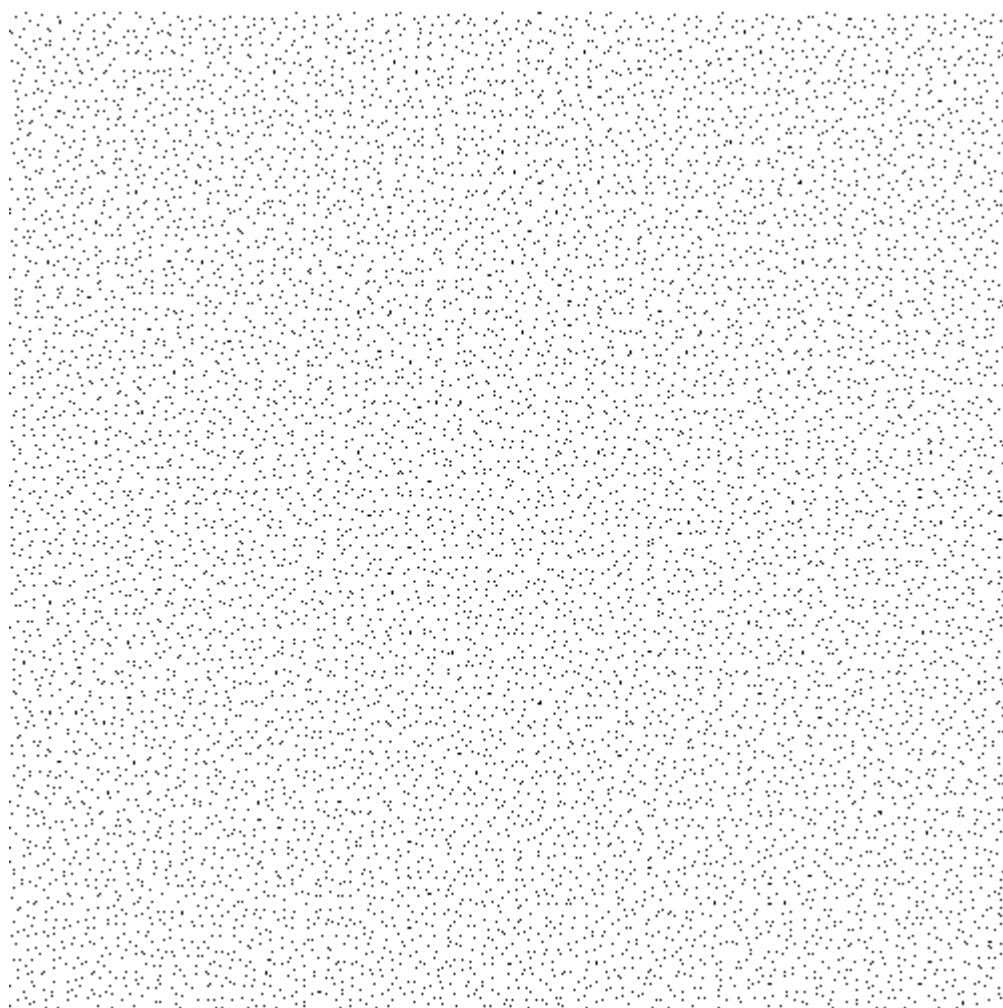
Rys 8.1: Domena „sferyczna”. Sposób powrotu elementu wirowego do domeny obliczeniowej.
(Opracowanie własne)

Wszystkie symulacje przedstawione w pracy dotyczą kwadratowej domeny obliczeniowej o wymiarze 100×100 . W domenie równomiernie rozmieszczone są wiry o mocach od -1 do 1. Został użyty równomierny generator liczb losowych Marsaglia (1995) [9] opisany formułą:

$$X_n = (1176 \cdot X_{n-1} + 1476 \cdot X_{n-2} + 1776 \cdot X_{n-3}) \bmod (2^{32} - 5) \quad (8.1)$$

Generator jest inicjalizowany czasem systemowym aktualnym w chwili wykonywania programu. Wartości uzyskiwane ze wzoru (8.1) są równomiernie przeliczane na przedział użytkowy, tzn. rozmiar domeny lub przedział wartości wirowości.

Jedna z analizowanych próbek została zwizualizowana na rysunku 8.2.



Rys 8.2: Losowo rozłożone wiry. Obraz pliku o nazwie 10k uzyskany za pomocą autorskiego programu vrb2txt. (Opracowanie własne)

8.2. Zrealizowane w ramach pracy programy komputerowe

W ramach realizacji pracy zostały napisane następujące programy komputerowe:

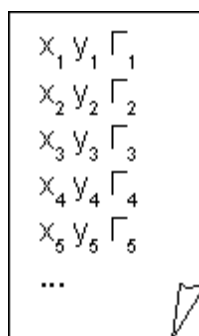
- *vorsym_s*
- *vorsym_q*
- *vorsym_p*
- *vspread*
- *vr2txt*
- *vr2bmp*

Wszystkie programy zostały napisane i przetestowane w systemie operacyjnym *Linux*. Programy znajdują się na dołączonej płycie CD. W celu skompilowania i zbudowania programów należy wydać polecenie *make* w katalogu głównym każdego z programów.

Programy *vorsym_s*, *vorsym_q*, *vorsym_p* zostały przetestowane dla uprzednio opisanych szczególnych konstelacji elementów wirowych.

8.2.1. Formaty plików używanych przez programy

Programy *vorsym_s*, *vorsym_q*, *vorsym_p* jako plik danych przyjmują plik tekstowy z rozszerzeniem *.vrt* z opisem położenia wirów i ich mocy. Format tego pliku jest przedstawiony na rysunku 8.3.



Rys 8.3: Format pliku *.vrt*. – x_i, y_i położenie elementu wirowego, Γ_i - moc elementu.
(Opracowanie własne)

Efekt działania programów jest plik binarny z rozszerzeniem *.vrb* z kolejnymi położeniami wirów pokazany na rysunku 8.4.

NAGŁÓWEK	MOCE WIRÓW	KOLEJNE POŁOŻENIA WIRÓW
----------	------------	-------------------------

Rys 8.4: Format pliku binarnego *.vrb*. (Opracowanie własne)

Szczegóły implementacyjne można odnaleźć w pliku nagłówkowym *vortex.h*.

8.2.2. Program *vorsym_s*

Program *vorsym_s* (*Vortex Simulator – Slow*) realizuje algorytm cząstka – cząstka. Program *vorsym_s* jest programem sekwencyjnym (jednowątkowym) Program wywołuje się z konsoli wpisując:

```
vorsym_s wejscie wynik NS D v dT
```

gdzie:

wejscie – nazwa pliku wejściowego *.vrt*

wynik – nazwa pliku wyjściowego *.vrb*

NS – liczba kroków symulacji do wykonania

D – rozmiar domeny obliczeniowej

v - lepkość płynu

dT – krok czasowy

Wynikiem działania programu jest plik binarny z rozszerzeniem *.vrb* zawierający opis kolejnych położenia wszystkich wirów.

8.2.3. Program *vorsym_q*

Program *vorsym_q* (*Vortex Simulator – Quick*) realizuje algorytm cząstka – cząstka – cząstka – siatka. Program *vorsym_q* jest programem sekwencyjnym (jednowątkowym). Program wywołuje się z konsoli poleceń wpisując:

```
vorsym_q wejscie wyjscie NS Nsd Nvs D v dT
```

gdzie:

wejscie – nazwa pliku wejściowego *.vrt*

wynik – nazwa pliku wyjściowego .vrb

NS – liczba kroków symulacji do wykonania

NSd – podział boku domeny; ilość subdomen to: $N_{sub} = N_{sd} \cdot N_{sd}$

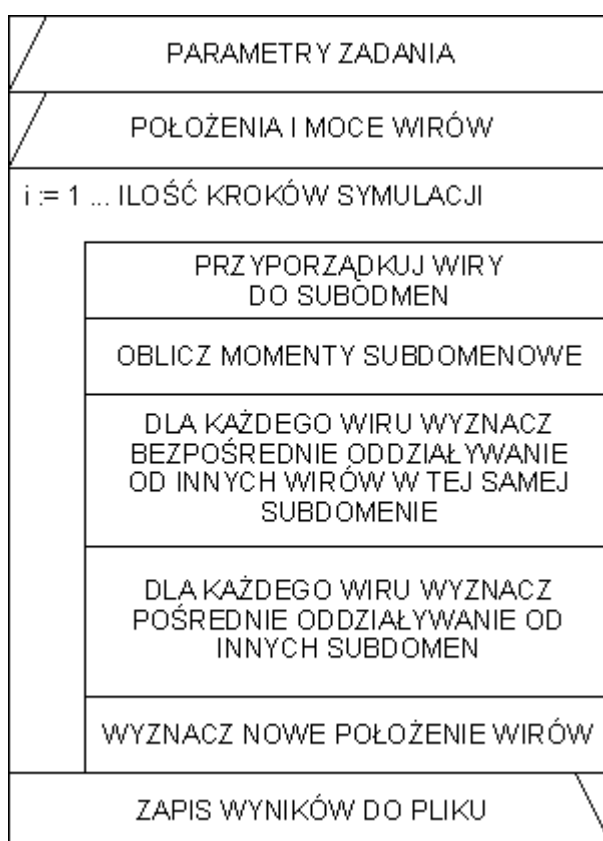
NVs – początkowa pojemność subdomeny; jeżeli 0 to program ustali automatycznie

D – rozmiar domeny obliczeniowej

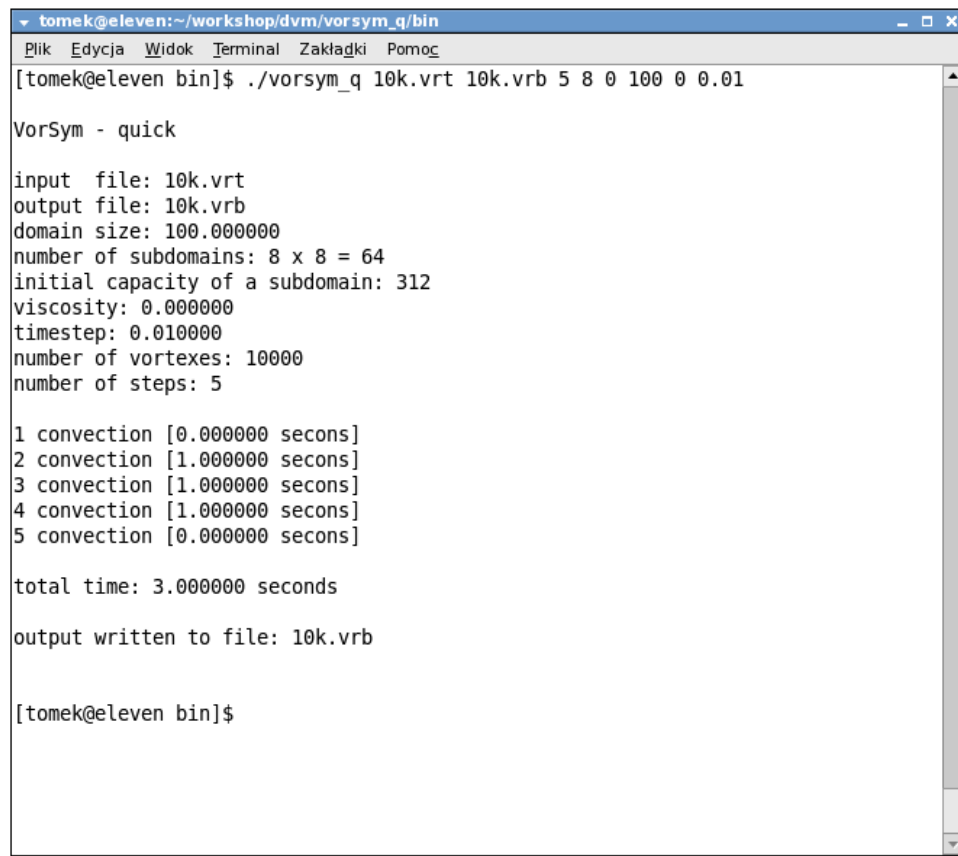
ν - lepkość płynu

dT – krok czasowy

Wynikiem działania programu jest plik binarny z rozszerzeniem .vrb zawierający opis kolejnych położenia wszystkich wirów. Rysunek 8.5 przedstawia uproszczony algorytm programu, zaś rysunek 8.6. pokazuje widok ekranu komputera wykonującego program.



Rys 8.5: Uproszczony algorytm programu vorsym_q. (Opracowanie własne)



```
tomek@eleven:~/workshop/dvm/vorsym_q/bin
Plik  Edycja  Widok  Terminal  Zakładki  Pomoc
[tomek@eleven bin]$ ./vorsym_q 10k.vrt 10k.vrb 5 8 0 100 0 0.01

VorSym - quick

input file: 10k.vrt
output file: 10k.vrb
domain size: 100.000000
number of subdomains: 8 x 8 = 64
initial capacity of a subdomain: 312
viscosity: 0.000000
timestep: 0.010000
number of vortexes: 10000
number of steps: 5

1 convection [0.000000 secons]
2 convection [1.000000 secons]
3 convection [1.000000 secons]
4 convection [1.000000 secons]
5 convection [0.000000 secons]

total time: 3.000000 seconds

output written to file: 10k.vrb

[tomek@eleven bin]$
```

Rys 8.6: Widok ekranu komputera wykonującego program *vorsym_q*. (Opracowanie własne)

8.2.4. Program *vorsym_p*

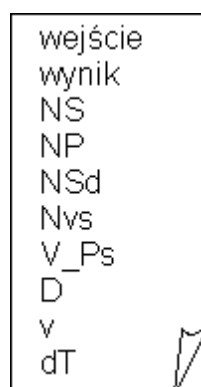
Program *vorsym_p* (*Vortex Simulator – Parallel*) realizuje algorytm cząstka – cząstka – cząstka – siatka. Program *vorsym_p* jest programem równoległym. Po uruchomieniu demona *mpd*, program wywołuje się z konsoli poleceń wpisując:

```
mpiexec -n P vorsym_p parametry
```

gdzie:

P – ilość procesów na których będzie wykonywał się program

parametry – plik tekstowy z parametrami wywołania programu przedstawiony na rysunku 8.7



Rys 8.7: Zawartość pliku z parametrami zadania.. (Opracowanie własne)

Parametry zadania to:

wejście – nazwa pliku wejściowego *.vrt* *wynik* – nazwa pliku wyjściowego *.vrb*

NS – liczba kroków symulacji do wykonania

NP – parametr dekompozycji geometrycznej; ilość subdomen procesowych wynosi:

$$P = NP \cdot NP$$

NSd – podział boku domeny dla procesu; ilość subdomen procesu to:

$$N_{\text{sub}} = N_{\text{sd}} \cdot N_{\text{sd}}$$

NVs – początkowa pojemność subdomeny; jeżeli 0 to program ustali automatycznie

V_Ps – należy podać 0

D – rozmiar domeny obliczeniowej

v- lepkość płynu

dT – krok czasowy

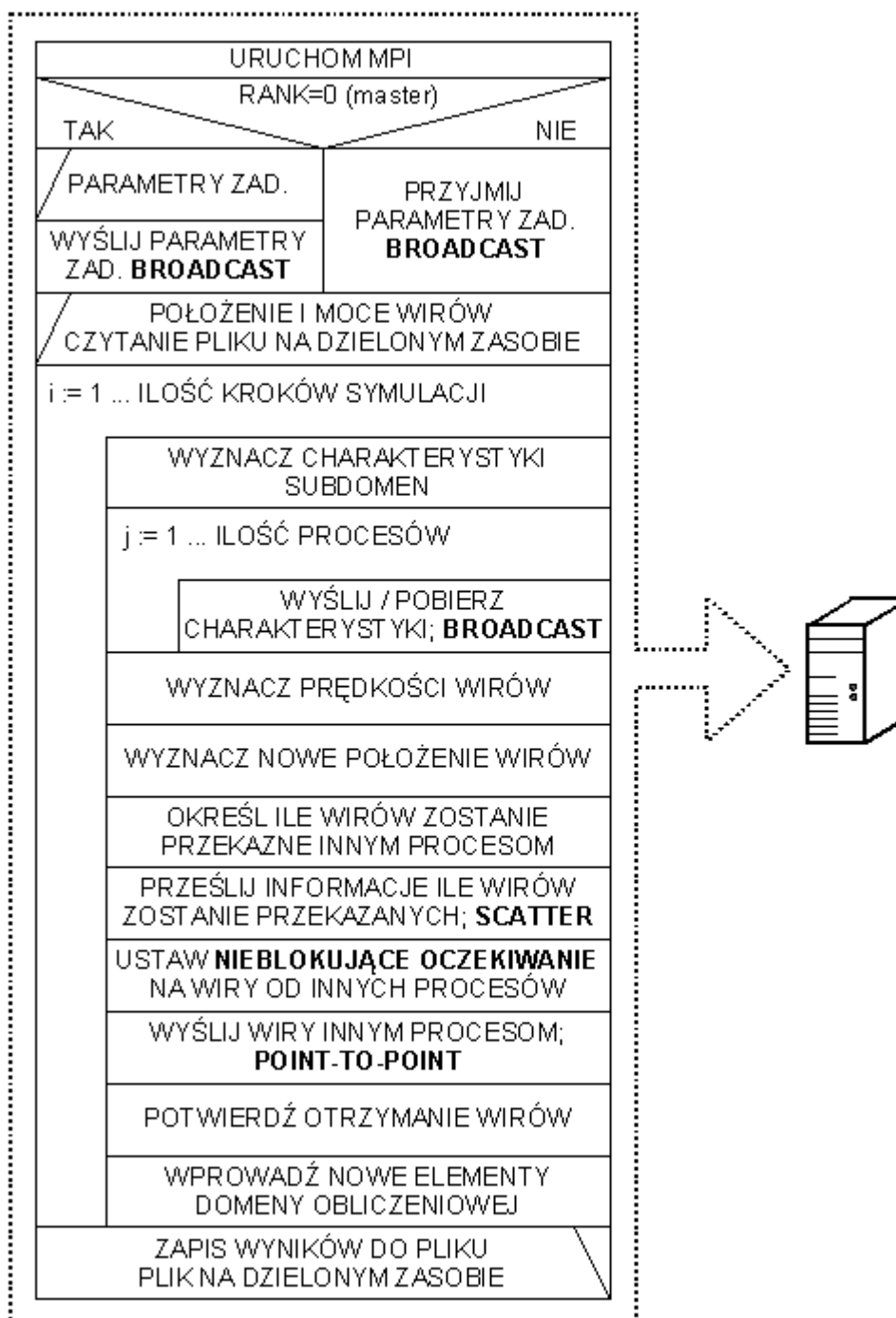
Program nie rozpocznie działania, jeżeli zostanie wywołany dla innej liczby procesów niż to wynika z dekompozycji geometrycznej. Zatem musi zachodzić warunek: $P = NP \cdot NP$

Rysunek 8.8 przedstawia uproszczony algorytm działania programu *vorsym_p*, tzn. algorytm wykonywany przez każdy proces. Zaś rysunek 8.9 pokazuje widok ekranu komputera wykonującego ten program.

W celu ograniczenia komunikacji między procesorami wymiana elementów wirowych odbywa się w 4 etapach:

1. Procesy przygotowują wiry do wysłania poprzez umieszczenie ich w ciągłych obszarach pamięci

2. Procesy wysyłają i odbierają od innych procesów informacje o ilości wirów, które będą przekazane
3. Procesy przygotowują ciągle obszary pamięci na wiry które zostaną im przesłane
4. Procesy asynchronicznie wysyłają i odbierają wiry



Rys 8.8: Uproszczony algorytm każdego z procesów programu vorsym_p. Symbol komputera oznacza, że taki sam algorytm wykonuje każdy proces. Pogrubionym drukiem zostały wskazane miejsca, w których zachodzi komunikacja między procesorami. (Opracowanie własne)

```

tomek@eleven:~/workshop/dvm/vorsym_p/bin
Plik  Edycja  Widok  Terminal  Zakładki  Pomoc
[tomek@eleven bin]$ mpirun -n 9 ./vorsym_p param.txt

VorSym - parallel

param file: param.txt
input file: 10k.vrt
output file: 10k.vrb
domain size: 100.000000
number of processes: 3 x 3 = 9
number of subdomains in one process: 9 x 9 = 81 total:729
viscosity: 0.000000
timestep: 0.010000
total number of vortexes: 10000
number of steps: 5
process NVs      V_Ps    number of vortexes
[2, 0] 14      2
[0, 1] 14      2
[0, 0] 14      2
[1, 1] 14      2
[1, 2] 14      2
[1, 0] 14      2
[0, 2] 14      2
[2, 1] 14      2
[2, 2] 14      2
step: 0
step: 1
step: 2
step: 3
step: 4
Creating the output file: 10k.vrb
Writing vortexes
6) eleven [2, 0]: done
1) eleven [0, 1]: done
0) eleven [0, 0]: done
5) eleven [1, 2]: done
4) eleven [1, 1]: done
3) eleven [1, 0]: done
2) eleven [0, 2]: done
7) eleven [2, 1]: done
8) eleven [2, 2]: done

```

Rys 8.9: Widok ekranu komputera wykonującego program *vorsym_p*. (Opracowanie własne)

8.2.5. Program *vspread*

Program *vspread* tworzy pliki danych użyte w przytoczonych analizach. Program wywołuje się wpisując w konsoli polecenie:

$$\text{vspread } D \ N \ \Gamma_{\min} \ \Gamma_{\max} \ \text{wynik}$$

gdzie:

D – rozmiar domeny obliczeniowej

N – podział boku domeny na przedziały; w związku z tym ilość wygenerowanych wirów wynosi $N \cdot N$

$(\Gamma_{\min}, \Gamma_{\max})$ - przedział zmienności natężenia generowanych wirów

wynik – nazwa pliku wynikowego (plik *.vrt*)

8.2.6. Program *vrb2txt*

Program *vrb2txt* służy do zamiany binarnego pliku *.vrb* na plik tekstowy. Program wywołuje się wpisując w konsoli polecenie:

```
vrb2txt wejście [wynik]
```

gdzie:

wejście – binarny plik *.vrb*

wynik – nazwa wynikowego pliku tekstowego; jeżeli nazwa nie zostanie podana, to jest tworzona na podstawie nazwy pliku wejściowego poprzez dodanie rozszerzenia *.txt*.

8.2.7. Program *vrb2bmp*

Program *vrb2bmp* służy do zamiany binarnego pliku *.vrb* na cykl obrazów zapisanych w plikach graficzny *.bmp*. W programie została wykorzystana biblioteka *EasyBMP* dostępna na licencji BSD. Program wywołuje się wpisując w konsoli polecenie:

```
vrb2bmp wejście rozmiar wynik
```

gdzie:

wejście – binarny plik *.vrb*

rozmiar – rozmiar bitmapy w pikselach

wyniki – cykl bitmap; ilość bitmap odpowiada ilości kroków zapisanych w pliku *.vrb*.

Nazwy kolejnych bitmap są tworzone automatycznie na podstawie podanej poprzez dołączanie sekwencyjnych numerów.

8.3. Przeprowadzone symulacje

Wszystkie symulacje przeprowadzone zostały dla domeny kwadratowej o wymiarach 100×100 . Moce wirów zmieniały się od $\Gamma_{min} = -1$ do $\Gamma_{max} = 1$. Przyjęty został krok czasowy wynoszący $dT = 0.01$. Maksymalny rząd wirowy uwzględniony w obliczeniach wynosił 5. Pliki z danymi wejściowymi zostały wygenerowane za pomocą programu *vspread*. Łącznie przeprowadzono 34 symulacje. Liczba elementów

wirowych zmieniała się od 9 do 10^6 . Pliki zestawia tabela 8.1. Wszystkie pliki znajdują się na dołączonej do pracy płycie CD.

Tabela 8.1: Pliki użyte do przeprowadzenia symulacji.

	<i>Nazwa pliku</i>	<i>Liczba wirów</i>
1	9.vrt	9
2	25.vrt	25
3	36.vrt	36
4	49.vrt	49
5	81.vrt	81
6	100.vrt	100
7	200.vrt	196
8	300.vrt	289
9	400.vrt	400
10	500.vrt	484
11	600.vrt	576
12	700.vrt	676
13	800.vrt	784
14	900.vrt	900
15	1k.vrt	1024
16	10k.vrt	10000
17	20k.vrt	19881
18	30k.vrt	29929
19	40k.vrt	40000
20	50k.vrt	49729
21	60k.vrt	59536
22	70k.vrt	69696
23	80k.vrt	69696
24	90k.vrt	90000
25	100k.vrt	100489
26	200k.vrt	200704
27	300k.vrt	299209
28	400k.vrt	399424
29	500k.vrt	499849
30	600k.vrt	600625
31	700k.vrt	700569

32	800k.vrt	801025
33	900k.vrt	900601
34	1M.vrt	1000000

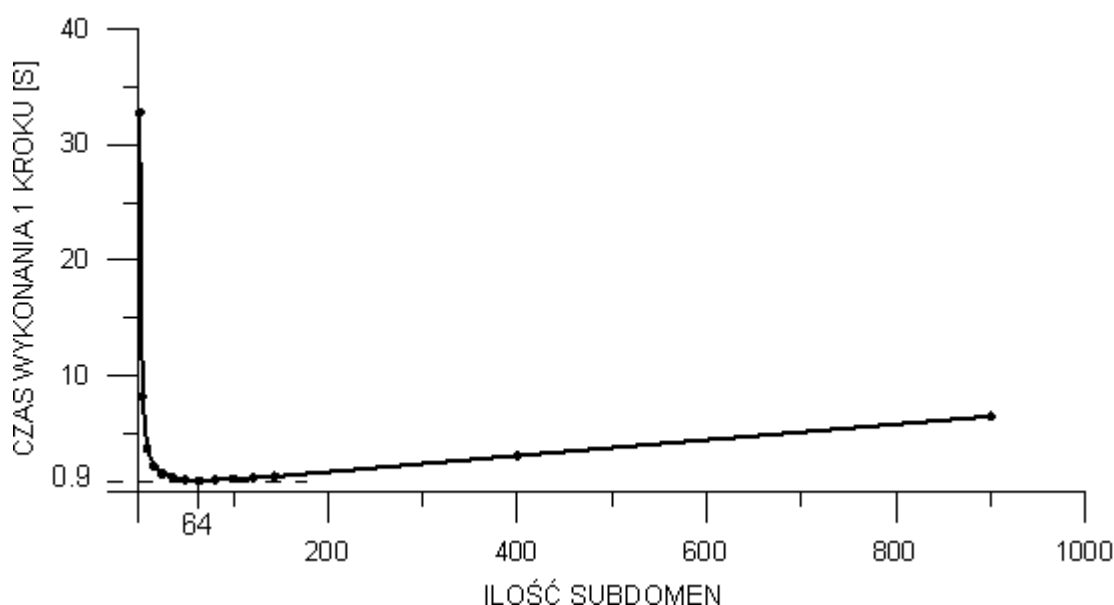
8.4. Wyniki symulacji

Celami przeprowadzonych symulacji było:

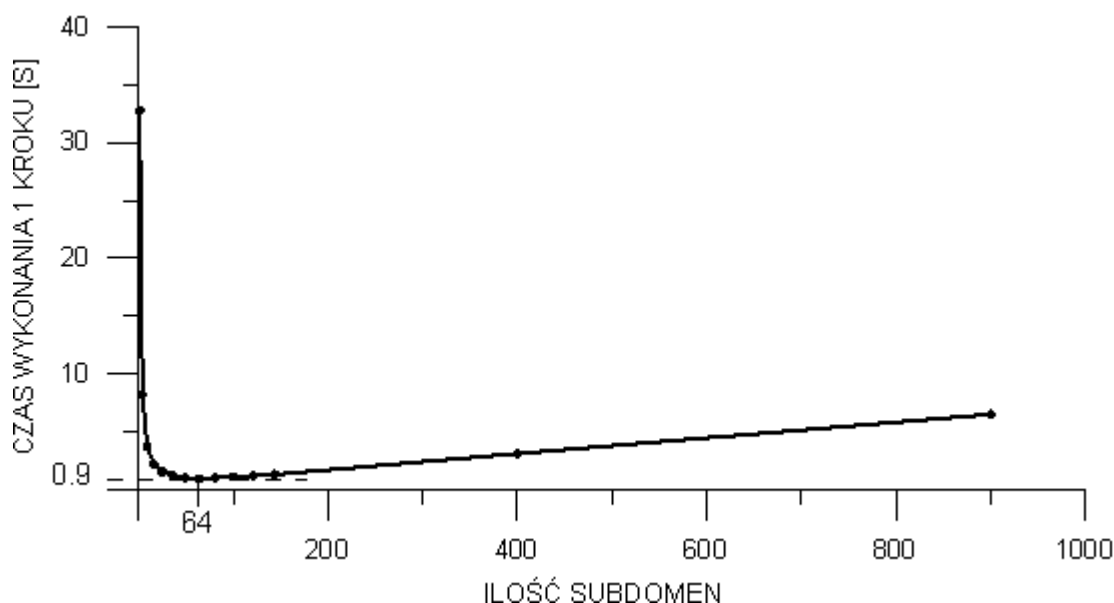
- potwierdzenie poprawności oszacowania optymalnej ilości subdomen
- praktyczne sprawdzenie wydajności skonstruowanego klastra komputerowego

8.4.1. Wpływ ilości subdomen na czas obliczeń

Sprawdzenie wpływu ilości subdomen na czas obliczeń został wykonany dla danych wejściowych z plików: 10k.vrt (10000 elementów wirowych) oraz 100k.vrt (100000 elementów wirowych). Dla tych danych optymalna ilość subdomen obliczona wg wzoru (6.15) wynosi 64 dla 10000 wirów oraz 225 dla 100000 wirów. Symulacje dla różnych ilości subdomen przeprowadzone zostały przy użyciu programu *vorsym_q*. Uzyskane wyniki przedstawione są na rysunku 8.10 oraz 8.11.



Rys 8.10: Wpływ ilości subdomen na czas obliczeń dla zadania złożonego z 10000 elementów wirowych (plik 10k.vrt). Najkrótszy czas jednego kroku zanotowany został dla 64 subdomen - 0.9 sekundy.
(Opracowanie własne)



Rys 8.11: Wpływ ilości subdomen na czas obliczeń dla zadania złożonego ze 100000 elementów wirowych (plik 100k.vrt). Najkrótszy czas jednego kroku zanotowany został dla 225 subdomen – 30.7 sekundy.
(Opracowanie własne)

Uzyskane wyniki potwierdzają poprawność oszacowania optymalnej ilości subdomen. Przedstawione wykresy wyraźnie demonstrują potrzebę odpowiedniego dobrania ilości subdomen. Nieodpowiedni podział domeny obliczeniowej może prowadzić do 2 a nawet 3-krotnie dłuższych czasów obliczeń. Z kształtów wykresów możemy wnioskować, że mniej niekorzystne jest przyjęcie większej ilości subdomen niż ilości mniejszej. Ten ostatni wniosek jest bardzo pomocny przy zaokrąglaniu do wartości całkowitych wyniku wzoru (6.15).

8.4.2. Czasy trwania symulacji

Symulacje zostały przeprowadzone dla wszystkich przygotowanych zadań (tabela 8.1). W zależności od wielkości zadania symulacja obejmowała od 3 do 10000 kroków. Ze względu na długi czas działania programu *vorsym_s* nie przeprowadzano tym programem symulacji dla większej ilości wirów niż 60000. Program równoległy *vorsym_p* został uruchamiany na 4 i na 9 węzłach. Program *vorsym_p* uruchamiany na n węzłach jest oznaczany jako *vorsym_p(n)*. Łącznie przeprowadzonych zostało 123 symulacje.

Uzyskane czasy obliczeń zestawiają tabela 8.2 oraz tabela 8.3. Wyniki zostały

także przedstawione na wykresach – rysunki od 8.12 do 8.15.

Tabela 8.2: Czas wykonania 1000 kroków symulacji [s].

Plik	Ilość wirów	vorsym_s	vorsym_q		vorsym_p(4)		vorsym_p(9)	
		Czas [s]	N_{sub}	Czas [s]	N_{sub}	Czas [s]	N_{sub}	Czas [s]
9.vrt	9	0.15	1	0.29	1	6.60	1	48.60
25.vrt	25	0.42	4	0.72	1	6.60	1	80.00
36.vrt	36	0.66	4	0.91	1	6.50	1	50.00
49.vrt	49	1.08	4	1.11	4	8.00	1	49.00
81.vrt	81	1.90	9	1.46	4	8.00	1	48.00
100.vrt	100	2.76	9	1.58	4	8.00	1	46.00
200.vrt	196	10.60	9	2.80	4	10.00	4	66.67
300.vrt	289	22.60	9	4.90	4	10.00	4	82.30
400.vrt	400	43.40	16	8.00	9	10.00	4	100.00
500.vrt	484	64.00	16	11.00	9	15.00	4	83.33
600.vrt	576	91.00	16	14.00	9	15.00	4	60.00
700.vrt	676	145.00	16	19.00	9	15.00	4	100.00
800.vrt	784	167.00	16	22.00	9	15.00	9	100.00
900.vrt	900	218.00	25	27.00	9	15.00	9	100.00
1k.vrt	1024	300.00	25	30.00	9	17.50	9	100.00

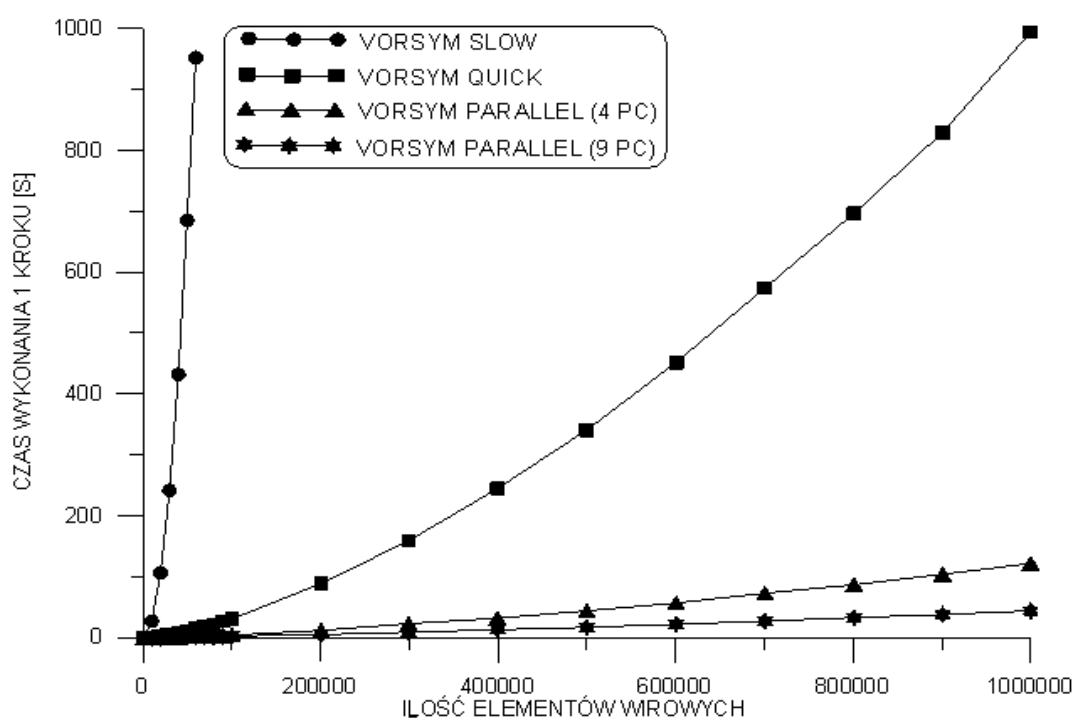
Tabela 8.3: Czas wykonania 1 kroku symulacji.

Plik	Ilość wirów	vorsym_s	vorsym_q		vorsym_p(4)		vorsym_p(9)	
		Czas [s]	N_{sub}	Czas [s]	N_{sub}	Czas [s]	N_{sub}	Czas [s]
1k.vrt	1024	0.3	25	0.03	9	0.02	9	0.1
10k.vrt	10000	27.1	64	0.9	36	0.3	25	0.1
20k.vrt	19881	106.3	100	2.7	49	0.8	36	0.5
30k.vrt	29929	241.7	121	5.1	64	1.0	36	0.7
40k.vrt	40000	432.0	144	7.8	64	1.3	49	1.0
50k.vrt	49729	685.0	144	10.7	81	1.7	49	1.3
60k.vrt	59536	952.0	169	14.3	81	2.0	64	1.3
70k.vrt	69696	-	169	18.1	100	2.7	64	1.7
80k.vrt	79524	-	196	21.7	100	3.0	64	2.0

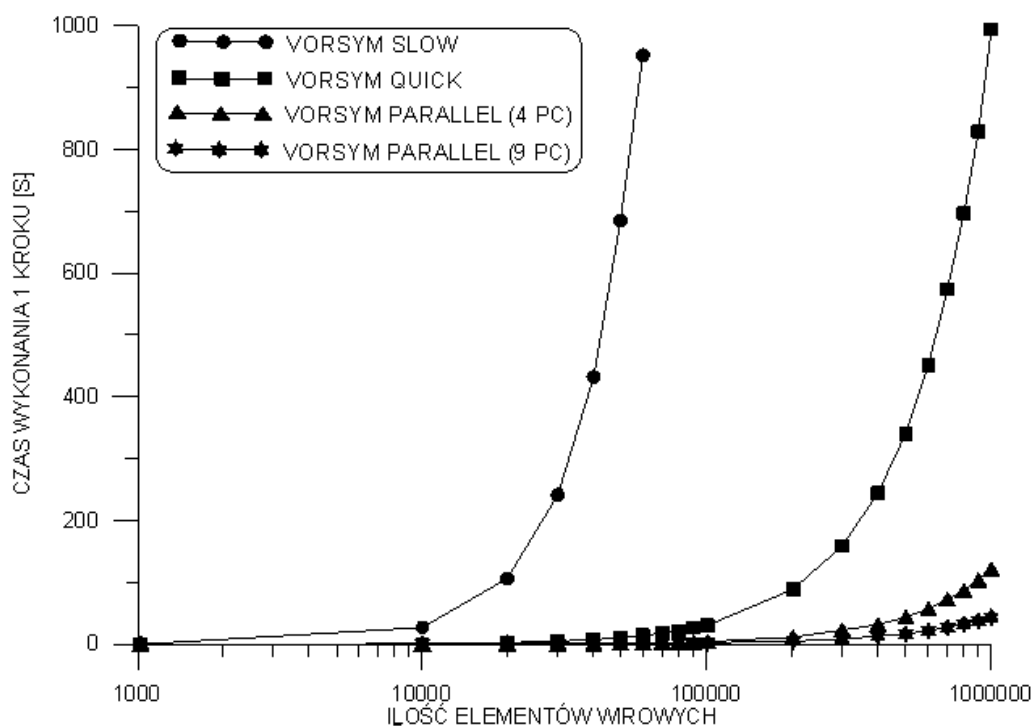
90k.vrt	90000	-	225	27.0	100	3.5	64	2.3
100k.vrt	100489	-	225	30.7	121	4.3	81	2.7

Tabela 8.3: Czas wykonania 1 kroku symulacji - ciąg dalszy.

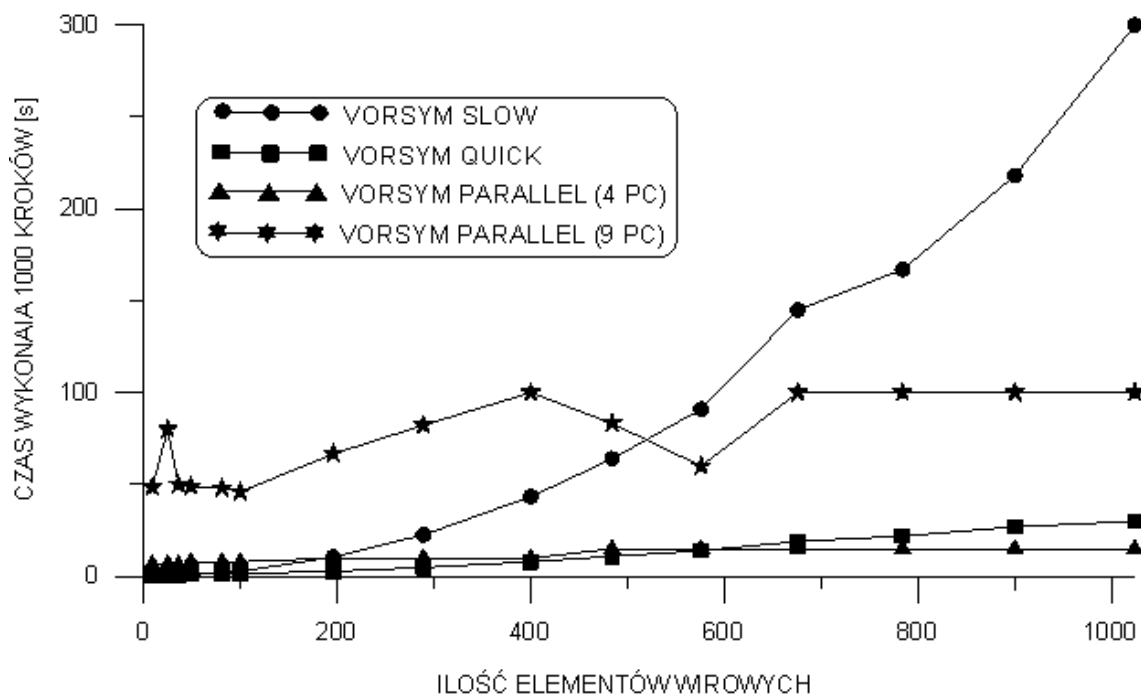
Plik	Ilość wirów	vorsym_s	vorsym_q		vorsym_p(4)		vorsym_p(9)	
		Czas [s]	N_{sub}	Czas [s]	N_{sub}	Czas [s]	N_{sub}	Czas [s]
200k.vrt	200704	-	324	89.0	169	12.0	100	5.0
300k.vrt	299849	-	400	159.0	196	23.0	121	9.0
400k.vrt	399424	-	441	245.0	225	32.0	144	13.0
500k.vrt	499849	-	484	341.0	256	44.0	169	17.0
600k.vrt	600625	-	529	452.0	289	57.0	196	22.0
700k.vrt	700569	-	576	574.0	289	73.0	196	27.0
800k.vrt	801025	-	625	697.0	324	87.0	225	33.0
900k.vrt	900601	-	625	829.0	324	104	225	38.0
1M.vrt	10^6	-	279	994.0	361	122	225	44.0



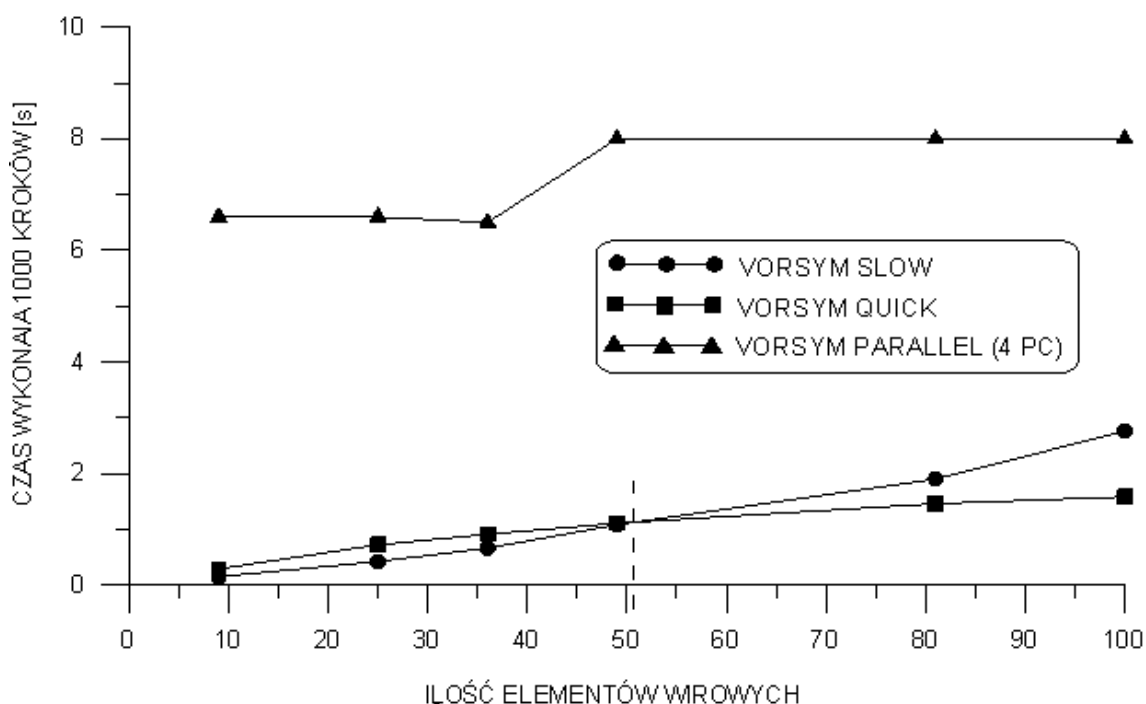
Rys 8.12: Uśredniony czas wykonania jednego kroku symulacji w zależności od wielkości zadania, tj. ilości elementów wirowych. Wyniki przedstawione w podziałce liniowej. (Opracowanie własne)



Rys 8.13: Uśredniony czas wykonania jednego kroku symulacji w zależności od wielkości zadania, tj. ilości elementów wirowych. Wyniki przedstawione w podziałce logarytmicznej. (Opracowanie własne)



Rys 8.14: Uśredniony czas wykonania 1000 kroków symulacji w zależności od wielkości zadania. Wyniki przedstawione w podziałce liniowej. (Opracowanie własne)



Rys 8.15: Uśredniony czas wykonania 1000 kroków symulacji w zależności od wielkości zadania – zbliżenie. Wyniki przedstawione w podziałce liniowej. (Opracowanie własne)

Na podstawie uzyskanych wyników można stwierdzić, że do 50 elementów wirowych najwydajniejszy jest algorytm bezpośredni (program *vorsym_s*). Jednowątkowy program wykonujący szybki algorytm (*vorsym_q*) jest najszybszy dla zadania od 50 do 600 wirów. W przedziale od 600 do 1000 elementów program równoległy (*vorsym_p(4)*) działający na 4 węzłach wykonuje zadane najszybciej. Powyżej 1000 elementów wirowych program równoległy (*vorsym_p(9)*) działający na 9 węzłach jest najszybszy.

Typowe zadanie inżynierskie rozwiązywane metodą wirów dyskretnych wymaga aby pole przepływu było modelowane przy użyciu kilkudziesięciu tysięcy elementów wirowych. Natomiast symulacja musi być wystarczająco długa, aby było możliwe wyznaczenie statystycznych cech przepływu. Zazwyczaj wystarczające jest przeprowadzenie 100 000 kroków. Orientacyjne czasy symulacji komputerowych wykonywanych przy użyciu analizowanych programów zestawia tabela 8.4. Wyniki zestawione w tabeli wyraźnie wskazują na celowość zastosowania klastra.

Tabela 8.4: Orientacyjne czasy wykonania symulacji składającej się ze 100 000 kroków obliczeniowych w zależności od wielkości zadania.

Program	20 000 wirów	50 000 wirów	100 000 wirów
vorsym_s	123 dni (4 miesiące)	793 dni (2 lata)	ok. 3140 dni (9 lat)*
vorsym_q	75 godzin (3 dni)	297 godzin (12 dni)	853 godzin (ok 36 dni)
vorsym_p(4)	22 godziny (< 1 dzień)	47 godzin (2 dni)	119 godzin (ok 5 dni)
vorsym_p(9)	14 godzin (< 1 dzień)	36 godzin (1.5 dnia)	75 godzin (ok 3 dni)

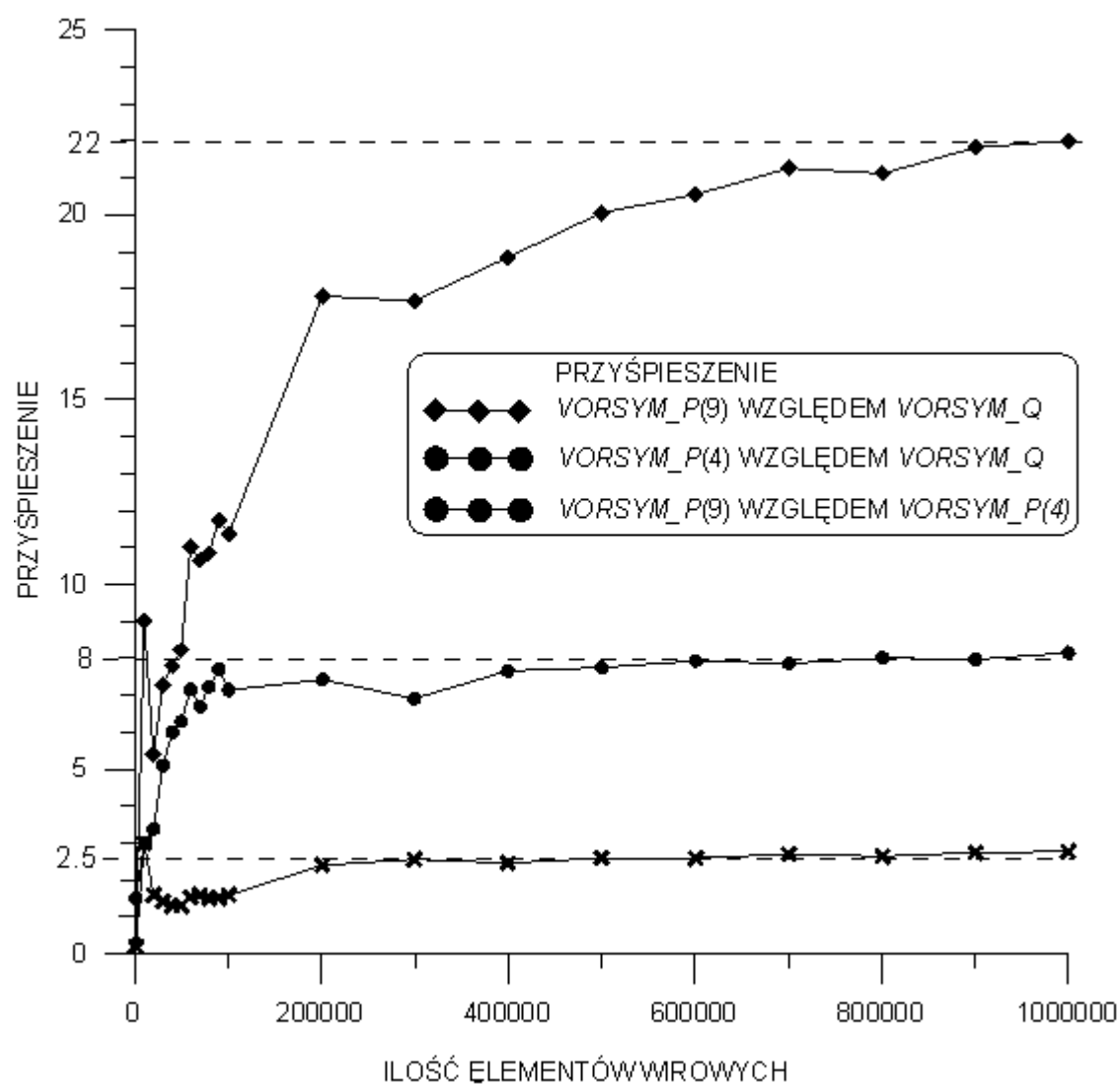
*oszacowanie przez ekstrapolację

8.4.3. Przyspieszenie i efektywność wykorzystania procesorów

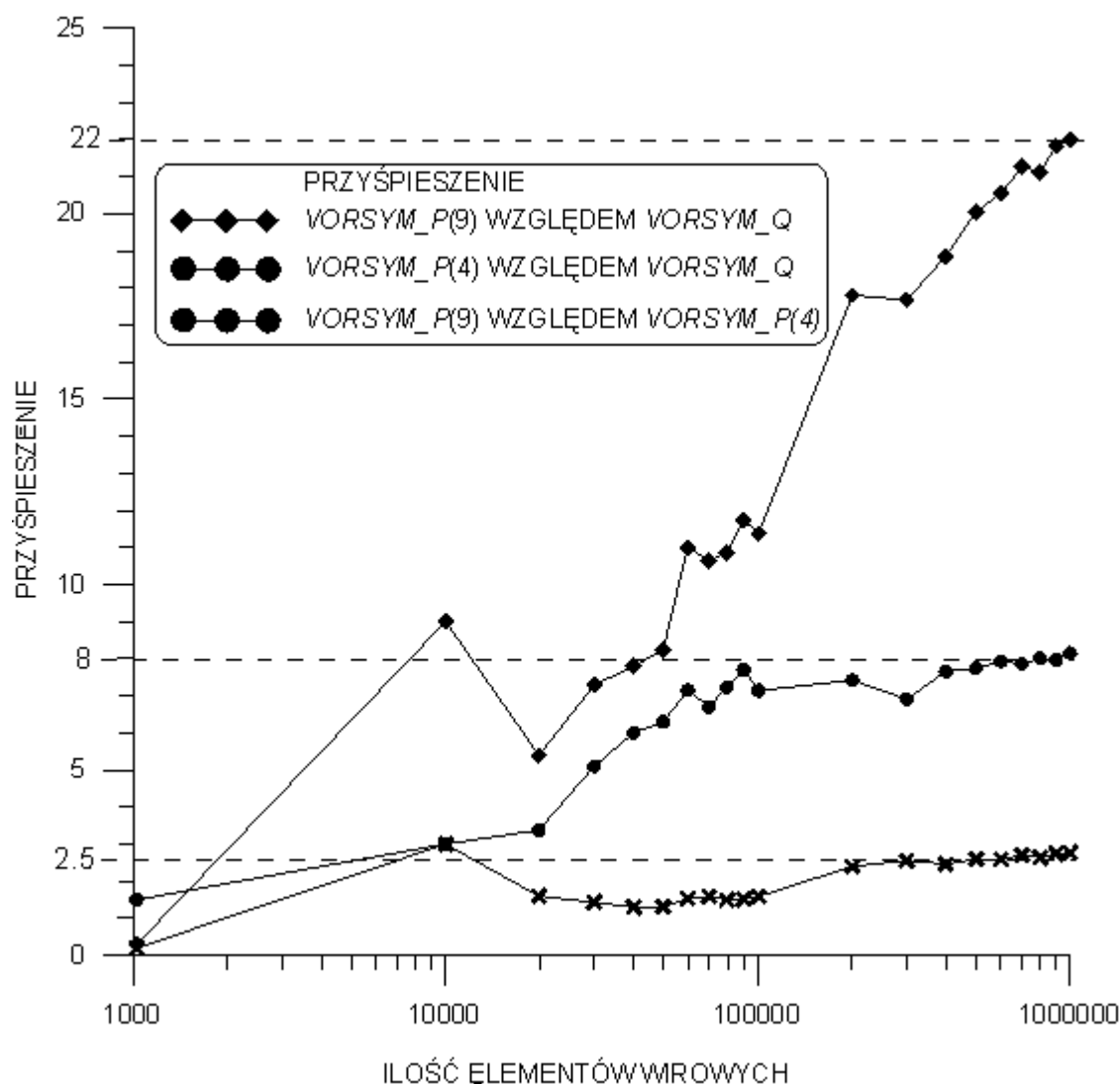
Na podstawie uzyskanych czasów symulacji obliczone zostało przyspieszenie względne analizowanych programów. Wyznaczone zostało:

- przyspieszenie programu równoległego działającego na 4 węzłach *vorsym_p(4)* względem programu jednowątkowego wykonującego szybki algorytm *vorsym_q*
- przyspieszenie programu równoległego działającego na 9 węzłach *vorsym_p(9)* względem programu jednowątkowego wykonującego szybki algorytm *vorsym_q*
- przyspieszenie programu równoległego działającego na 9 węzłach *vorsym_p(9)* względem programu równoległego działającego na 4 węzłach *vorsym_p(4)*

Wyniki zostały przedstawione w formie wykresów na rysunkach 8.16 oraz 8.17.



Rys 8.16: Przyspieszenie względne programów w zależności od rozmiaru zadana. Podziałka linowa.
(Opracowanie własne)



Rys 8.17: Przyspieszenie względne programów w zależności od rozmiaru zadania. Podziałka logarytmiczna o podstawie 10. (Opracowanie własne)

Z przedstawionych wykresów wynika, że przyspieszenia względne wraz ze wzrostem rozmiaru zadania stabilizują się na określonym poziomie. Program równoległy wykonywany na 4 węzłach jest ok. 8 razy szybszy niż program jednowątkowy. Program równoległy wykonywany na 9 węzłach jest ok 22 razy szybszy niż program jednowątkowy. Program wykonywany równoległy wykonywany na 9 węzłach jest ok 2.5 raza szybszy niż wtedy, kiedy jest wykonywany na 4 węzłach. Świadczy to o dużej lokalności obliczeń uzyskanej za pomocą dużej granuli obliczeniowej. Algorytm wykazuje w związku z tym dużą homogeniczność i wysoki współczynnik obliczenia – komunikacja.

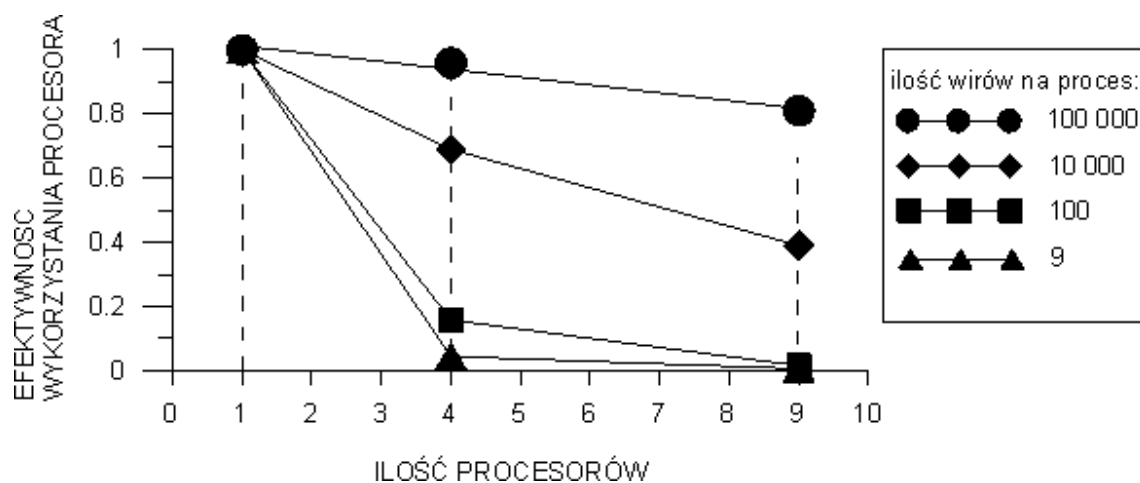
Uzyskane wysokie przyspieszenia, tzn. przekraczające proporcję zastosowanych

procesorów, wytłumaczyć należy silną zależnością czasu wykonania zadania od jego wielkości. Zatem w analizowanym przypadku założenie Amdahla o niezależności części szeregowej programu do jego części równoległej nie jest prawdziwe. W przypadku mechaniki wirowej wraz ze wzrostem ilości wirów dyskretnych bardzo szybko rośnie liczba obliczeń wykonywanych w fazie równoległej pracy programu.

Jednak uzyskane wyniki ciągle wskazują na spadek wydajności systemu wieloprocessorowego. Należy bowiem porównywać ze sobą zadania, przy których na jeden proces przypada taka sama ilość elementów wirowych tabela 8.5. Wówczas wyraźnie widać, że spadek wydajności pojedynczego mikroprocesora następuje i jest również zależny od wielkości zadania. Przy czym spadek wydajności w mniejszych zadaniach jest szybszy (rys 8.18).

Tabela 8.5: Spadek efektywności wykorzystania procesora.

<i>Program</i>	<i>Nazwa zadania</i>	<i>Ilość wirów na proces</i>	<i>Czas [s]</i>	<i>Efektywność [-]</i>
<i>vorsym_q</i>	9.vrt	9	0.29 (1000 kroków)	1
<i>vorsym_p(4)</i>	36.vrt	9	6.50 (1000 kroków)	$0.29/6.50=\mathbf{0.045}$
<i>vorysm_p(9)</i>	81.vrt	9	48.00 (1000 kroków)	$0.29/48.00=\mathbf{0.006}$
<i>vorsym_q</i>	100.vrt	100	1.58 (1000 kroków)	1
<i>vorsym_p(4)</i>	400.vrt	100	10.00 (1000 kroków)	$1.58/10.00=\mathbf{0.158}$
<i>vorysm_p(9)</i>	900.vrt	100	100.00 (1000 kroków)	$1.58/100.00=\mathbf{0.016}$
<i>vorsym_q</i>	10k.vrt	10 000	0.9 (1 krok)	1
<i>vorsym_p(4)</i>	40k.vrt	10 000	1.3 (1 krok)	$0.9/1.3=\mathbf{0.69}$
<i>vorysm_p(9)</i>	90k.vrt	10 000	2.3 (1 krok)	$0.9/2.3=\mathbf{0.39}$
<i>vorsym_q</i>	100k.vrt	100 489	30.7 (1 krok)	1.00
<i>vorsym_p(4)</i>	400k.vrt	999 962	32.0 (1 krok)	$30.7/32.0=\mathbf{0.96}$
<i>vorysm_p(9)</i>	900k.vrt	100 000	38.0 (1 krok)	$30.7/38.0=\mathbf{0.81}$



Rys 8.18: Spadek efektywności pojedynczego procesora w klastrze w zależności od ilości węzłów dla zadań o różnym rozmiarze. (Opracowanie własne)

9. Podsumowanie i wnioski

Niskobudżetowe klastery komputerowe budowane w oparciu o istniejącą infrastrukturę komputerową i darmowe oprogramowanie dostępne w zasobach światowego Internetu są przydatnym narzędziem dla inżyniera teoretyka w obliczu braku profesjonalnej infrastruktury obliczeniowej. Pozwalają znacznie zredukować czas symulacji komputerowych lub przeprowadzać większe symulacje niż przy użyciu pojedynczego komputera.

W prezentowanej pracy przedstawiono realizację klastra komputerowego i zbadano jego możliwości w przypadku zastosowania do realizacji algorytmów obliczeniowych metody wirów dyskretnych. W tym celu napisanych zostało 5 programów komputerowych.

Pracę można zakończyć następującymi wnioskami:

- Za sprawą Otwartego Oprogramowania implementacja klastra komputerowego na istniejącym sprzęcie nie wiąże się z żadnymi kosztami
- Na obecnym stopniu rozwoju techniki komputerowej budowa i konfiguracja klastra nie wiąże się z większymi komplikacjami.
- Ogólna dostępność bibliotek zgodnych ze standardem MPI pozwala szybko implementować algorytmy równoległe.
- Klaster komputerowy tworzony na bazie istniejącego laboratorium jest zaawansowanym systemem obliczeniowym i może być z powodzeniem stosowany do rozwiązywania problemów inżynierskich.
- Algorytmy obliczeniowe metody wirów dyskretnych ze względu na dekompozycję geometryczną osiągają duże przyspieszenia w postaci równoległej.

10. Literatura

Materiały drukowane:

- [1] Bogdan Edward Borowik, *Programowanie równoległe w zastosowaniach*, MIKOM, Warszawa wrzesień 2001, ISBN 83-7279-176-7
- [2] Thomas H.Cormen, Charles E. Leiserson, Ronald L. Rivest, *Wprowadzenie do algorytmów*, Wydawnictwa Naukowo-Techniczne, Warszawa 2001, wydanie piąte, ISBN-83-204-2800-9
- [3] Joseph Sloan, *High Performance Linux Clusters with OSCAR, Rocks, openMosix, and MPI*, O'Reilly 2005, ISBN 0-596-00570-9
- [4] R. I. Lewis, *Vortex Element Methods for Fluid Dynamics of Engineering Systems*, Cambridge University Press, 1991, ISBN-13 978-0-521-01754-1
- [5] Georges-Henri Cottet, Petros D. Koumoutsakos, *Vortex Methods, Theory and Practice*, Cambridge University Press 200, ISBN 0-521-62186-0
- [6] Roger W. Hockney, James W. Eastwood, *Computer Simulation Using Particles*, Advanced Book Program, NASA
- [7] G. Turkiyyah, D. Reed, J. Yang, *Fast vortex methods for predicting wind – induced pressures on buildings*, J. Wind Eng. Ind. Aerodyn. 58 (1995)
- [8] A. Larsen, J. H. Walther, *Aeroelastic analysis of bridge girder sections based on discrete vortex simulation*, J. Wind Eng, Ind. Aerodyn. 67&68 (1997).
- [9] R. Wieczorkowski, R. Zieliński *Komputerowe generatory liczb losowych*, Wydawnictwo Naukowo Techniczne, Warszawa 1997, ISBN 83-204-2160-8

Materiały elektroniczne:

- [10] www.mpi-forum.org
- [11] www-unix.mcs.ani.gov/mpi
- [12] www-unix.mcs.ani.gov/mpi/mpich2/
- [13] www.openclustergroup.org
- [14] www.beowulf.org

-
- [15] B. Reitinger, *On-line Program and Data Visualization of Parallel Systems in Monitoring and Steering Environment*, Diploma Thesis, Institute of Technical ComputerScience and Telematics, Department for Graphics and Parallel Processing Johannes Kepler University, Linz

11. Dodatek

Dołączona do pracy płyta CD zawiera:

- Omawianie w pracy autorskie programy komputerowe - katalog **dvm**:
 - *vorsym_s*
 - *vorsym_q*
 - *vorsym_p*
 - *vrb2txt*
 - *vrb2bmp*
 - Zadania analizowane w pracy - katalog **symulacje**:
 - treść pracy – katalog **treść_pracy**:
 - tomasz_nowicki_praca_magisterska.odt* – format OpenOffice
 - tomasz_nowicki_praca_magisterska.pdf*
- Plikiem źródłowym jest plik z rozszerzeniem *.odt*.

