AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE



Wydział Inżynierii Metali i Informatyki Przemysłowej



Praca Magisterska

pt.

„Implementacja metody dynamiki molekularnej dla heterogenicznych platform sprzętowych”

Imię i nazwisko dyplomanta: **Tomasz Nowak**

Kierunek studiów: **Informatyka Stosowana**

Profil dyplomowania:  **Modelowanie i Technologie Informacyjne**

Nr albumu: **232187**

Promotor: **drinż. Łukasz Rauch**

Podpis dyplomanta: Podpis promotora:

Kraków 2014

***Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszy projekt inżynierski wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.***

Kraków, dnia ………….… Podpisdyplomanta…………….

**Spis treści**

[1. Wstęp 4](#_Toc398667623)

[2. Dynamika molekularna 5](#_Toc398667624)

[2.1. Przegląd artykułów 6](#_Toc398667625)

[2.2. Przegląd algorytmów 6](#_Toc398667626)

[3. Realizacja oprogramowania 8](#_Toc398667627)

[3.1. GPU 8](#_Toc398667628)

[3.2. Nvidia CUDA 8](#_Toc398667629)

[3.3. Pthreads 9](#_Toc398667630)

[3.4. OpenGL 10](#_Toc398667631)

[3.5. Projekt aplikacji 10](#_Toc398667632)

[3.6. Implementacja 13](#_Toc398667633)

[3.7. Optymalizacja 13](#_Toc398667634)

[4. Testy wydajności 14](#_Toc398667635)

[4.1. Testy sekwencyjne 14](#_Toc398667636)

[4.2. Testy zrównoleglenia 14](#_Toc398667637)

[4.3. Testy optymalizacji 14](#_Toc398667638)

[4.4. Test wielu GPU 14](#_Toc398667639)

[5. Model algorytm - urządzenie 15](#_Toc398667640)

[5.1. XXX 15](#_Toc398667641)

[6. Podsumowanie 16](#_Toc398667642)

[7. Bibliografia 17](#_Toc398667643)

# Wstęp

Rosnąca moc obliczeniowa współczesnego sprzętu dostarcza nowe możliwości rozwiązywania coraz bardziej skomplikowanych zadań. Zagadnieniem bardzo wymagającym obliczeniowo jest dynamika molekularna. Jest jednocześnie ciekawym obszarem badań, który łączy w sobie wiele dziedzin nauki. Zastosowanie dynamiki molekularnej można znaleźć w badaniach nad własnościami fizycznymi materiałów. Symulacje z wykorzystaniem nanostruktur potrafią usprawnić badania, ograniczając czasochłonne wykonywanie doświadczeń z wykorzystaniem rzeczywistych próbek. Skrajnie różnym, ale bazującym na zbliżonej symulacji, doświadczeniem jest analiza wytrzymałości wiązań chemicznych. Domena tego typu symulacji jest dużo mniejsza, ale opiera się wciąż na działaniu sił zewnętrznych na nanostrukturę. Można szukać podobnych zagadnień również w astronomii, biologii, fizyce lub najprościej rzecz ujmując wszędzie, ponieważ wszystko jest zbudowane z atomów. Dlatego właśnie dynamika molekularna opisująca interakcję sił zewnętrznych z atomami, może zostać wykorzystana do dowolnych badań w skali nano. Aby w pełni korzystać z możliwości jakie daje ta metoda niezbędny jest odpowiedni sprzęt umożliwiający przeprowadzenie symulacji. Wykorzystanie akceleratorów graficznych GPU jest odpowiednim rozwiązaniem, ze względu na łatwą dekompozycję danych oraz możliwość efektywnego wykorzystania oferowanej przez sprzęt masowej wielowątkowości. Odpowiednia implementacja zagadnienia umożliwi wielokrotne wykorzystanie powstałej aplikacji do rozwiązywania problemów z różnych dziedzin przy niewielkim nakładzie pracy włożonym w modyfikację i przystosowanie do aktualnie analizowanej sytuacji.

Celem pracy jest stworzenie jak najbardziej uniwersalnej i otwartej implementacji algorytmów dynamiki molekularnej przy wykorzystaniu akceleratorów GPU. Implementacja zostanie wykorzystana do analizy problemów odkształceń materiałów przy różnorodnej definicji działających sił zewnętrznych np. temperatury, prasy, symulacji defektu struktury.

# Dynamika molekularna

Dynamika molekularna jest komputerową symulacją fizycznych ruchów atomów wewnątrz struktury. Atomy mogą wchodzić w interakcję między sobą w kolejnych krokach czasowych, dając wrażenie poruszania się. Siły pomiędzy atomami oraz potencjał energetyczny definiowane są przez mechanikę molekularną pól siłowych.[9]

//ROWNANIE F=m\*a i z gradientem

N – ilość atomów w strukturze

mi – masa atomu

a – przyspieszenie

Fi – działająca siła

V(r1 …. rN) – funkcja pozycji atomów; reprezentowana jest przez potencjał energetyczny

W praktyce potencjał jest zapisywany jako suma interakcji pomiędzy parami atomów [8]:

//ROWNANIE V(r1….rN) = SUMu1…………u2….

Najczęściej używanym praktycznie potencjałem jest potencjał Lenarda – Jonesa (LJ). Jest najczęściej wykorzystywany w modelowaniu interakcji pomiędzy atomami. Definiuje go poniższe równanie [8]:

//ROWNANIE LJ PDF Ti.odio.twiki

r – dystans pomiędzy atomami będącymi w interakcji

DELTA – średnica

EURO – minimum lokalne potencjału

EURO i DELTA są stałymi wyznaczanymi tak, aby definiowały właściwości fizyczne materiału [8].

Najbardziej czasochłonną częścią algorytmu jest obliczanie sił interakcji pomiędzy atomami. Zajmuje to ok. 90% czasu całej symulacji. Analizując przebieg symulacji widać, że obliczenie siły występuje pomiędzy każdym atomem struktury z każdym z pozostałych. Daje to złożoność obliczeniową rzędu O(N2). [8]

## Przegląd artykułów

W trakcie analizowania problemu dynamiki molekularnej, można spotkać się z wieloma sposobami optymalizacji algorytmu tak, aby znacząco zmniejszyć złożoność obliczeniową przy jednoczesnym zachowaniu jak największej dokładności.

Jednym ze sposobów jest zastosowanie tzw. promienia odcięcia. Metoda ta redukuje ilość obliczeń tylko do tych niezbędnych, ponieważ interakcja z bardziej oddalonymi atomami jest na tyle znikoma, że może zostać pominięta.[8]

Problem zdefiniowany globalnie tj. przechowywanie całej struktury w pamięci urządzenia nie jest optymalny i ciężki do równoległego wykonywania na więcej niż i układzie GPU jednocześnie. Aby usprawnić obliczenia i przygotować dane wejściowe na platformę równoległą można zastosować listę sąsiadów. Dla każdego atomu, na podstawie promienia odcięcia definiuje się listę atomów, które spełniają dany warunek i tylko one trzymane są w pamięci podczas obliczeń interakcji z wybranym atomem. Daje to możliwość podzielenia danych niezależnie od wielkości całej struktury i rozesłania mniejszych list na kilka niezależnych urządzeń. Rozwiązuje również problem z pamięcią, gdy struktura jest na tyle duża, że nie zmieści się w pamięci pojedynczego urządzenia. Lista sąsiadów jest aktualizowana co pewien określony przedział czasowy. [8]

Tworzenie listy sąsiadów jest skuteczne podczas symulacji bazujących na gęsto upakowanych atomach (ciała stałe, gęste ciecze) ponieważ atomy są ułożone bardzo blisko siebie i nie mają możliwości szybkiego przemieszczania się. Atomy gazów, mają dużo większe odległości i są niejednorodnie rozłożone w strukturze. Wymagałoby to zastosowanie dużo większego promienia odcięcia i znacznie częstszych aktualizacji list sąsiadów, aby wynik symulacji był wciąż miarodajny i poprawny.

## Przegląd algorytmów

Na przestrzeni lat powstało wiele implementacji dynamiki molekularnej. Algorytmy różniły się od siebie ze względu na zależności związane z architekturą sprzętu na którym powstawały lub zależnościami samej aplikacji. Z punktu widzenia dekompozycji danych można podzielić je następująco [8]:

* Dekompozycja atomów

Każdy z procesorów ma przydzieloną podzbiór danych wielkości N/P (N jest liczbą atomów; P jest liczbą procesorów) na początku symulacji. Każdy z procesorów musi przechowywać identyczną kopię informacji o atomach (metoda replikacji danych). Metoda ta znalazła zastosowanie wśród architektur z pamięcią współdzieloną. [8]

* Dekompozycja siły

Podzbiór danych pętli sił jest przydzielany każdemu procesorowi. Redukuje to kosztowną komunikację oraz koszt dostępu do pamięci do (PIERWIASTEK Z P) w porównaniu z dekompozycją atomów. Problemem jest odpowiednie zbalansowanie rozkładu danych pomiędzy procesory. [8]

* Dekompozycja przestrzeni

Polega na dekompozycji geometrycznej fizycznej struktury atomów. Każdy procesor oblicza siły działające na atomy tylko na fragmencie całej struktury. W trakcie symulacji procesory wymieniają ze sobą tylko atomy które przekroczyły granice swojej części struktury i przemieściły się do sąsiedniej. Metoda ta wspiera wielkoskalowe symulacje. Osiąga optymalną skalowalność O(N/P). [8]

Biorąc pod uwagę architekturę jaka będzie wykorzystywana do wykonywania symulacji, najlepszą metodą dekompozycji danych jest dekompozycja atomów. Główną zaletą tego podejścia jest bardzo dobre zrównoważenie obciążenia pomiędzy urządzeniami oraz łatwa do osiągnięcia skalowalność. Model współdzielonej pamięci GPU pomiędzy procesorami umożliwia osiągnięcie wysokiej wydajności w trakcie obliczeń.

# Realizacja oprogramowania

## GPU

Karty graficzne stają się coraz bardziej zaawansowane, przez co zaczynają być wykorzystywane również do ogólnych zastosowań. Superkomputery rozwiązują wiele skomplikowanych zadań obliczeniowych na liczbach zmiennoprzecinkowych. Procesory GPU dzięki swojej równoległej architekturze, są idealnym wsparciem w obliczeniach. Aby mogły zostać zastosowane w tym celu, potrzebny jest zestaw instrukcji który umożliwi pisanie aplikacji w sposób zrozumiały zarówno dla urządzenia jak i programisty. [7]

Nvidia jest wiodącym producentem procesorów GPU. Jest również twórcą CUDA - biblioteki i środowiska umożliwiającego wykorzystanie kart graficznych do ogólnych zastosowań.

## Nvidia CUDA

Nvidia CUDA jest platformą służącą do obliczeń równoległych oraz modelem programistycznym umożliwiającym uzyskać znacząco lepszą wydajność obliczeniowa wykorzystując moc układu graficznego GPU.

Środowisko daje programiście bezpośredni dostęp do zestawu wirtualnych instrukcji i pamięci na równoległych jednostkach procesorów. Można dzięki temu wykorzystywać GPU do ogólnych zastosowań nie tylko do przetwarzania grafiki. Biblioteki CUDA wspierają popularne języki programowania: C, C++, Fortran.

Aplikacje wspierające akceleratory graficzne GPU oraz biblioteki CUDA wykorzystywane są między innymi w:

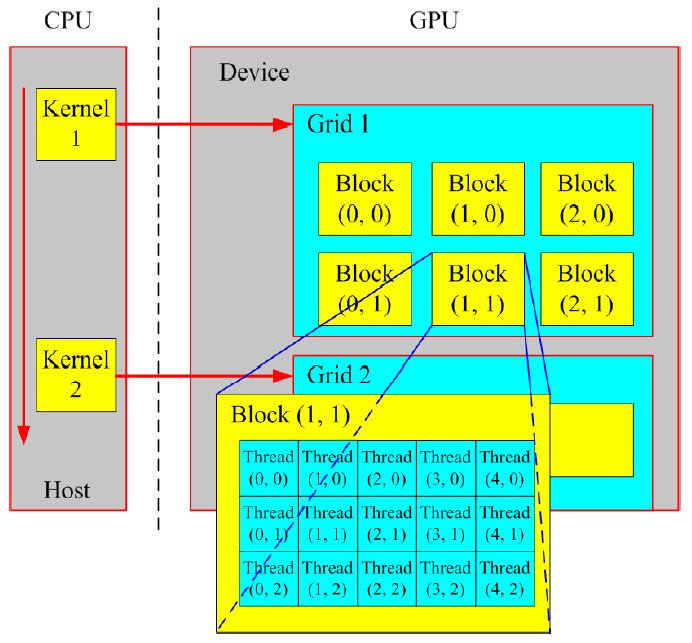
* astronomii,
* biologii,
* chemii,
* fizyce,
* finansach,
* procesach przemysłowych.

Model programowania CUDA definiowany jest przez kilka podstawowych założeń prawdziwych dla programów pisanych pod karty Nvidii:

* kod równoległy tzw. *kernel* jest uruchamiany na urządzeniu (karcie graficznej)   
  i wykonywany przez wiele wątków,
* wątki są grupowane w bloki wątków,
* kod równoległy jest pisany dla pojedynczego wątku
  + każdy wątek wykonuje unikalną część kodu
* bloki są grupowane w siatki

Na potrzeby implementacji zagadnienia będącego tematem pracy wykorzystane zostały [4]:

* CUDA Toolkit 5.5
* CUDA Samples – biblioteki wspomagające oraz funkcje pomocnicze
* CUDA Tools – narzędzia do profilowania i badania pamięci



Rys 3.1. Model programowania CUDA

## Pthreads

POSIX Threads często nazywany również Pthreads jest standardem POSIX dla wątków. Standard ten definiuje API[[1]](#footnote-2) do tworzenia i manipulowania wątkami. Implementacja ta jest dostępna dla wielu systemów operacyjnych opartych na systemie UNIX takich jak: FreeBSD, NetBSD, OpenBSD, GNU/Linux, Mac OS X, Solaris. Istnieje również implementacja przeznaczona dla systemu Microsoft Windows zawarta w WinAPI[[2]](#footnote-3). [6]

Pthreads definiuje zestaw typy, funkcje, stałe dla języka C. Implementacja znajduje się w pliku *pthreads.h* biblioteki zarządzającej wątkami. Zawiera ponad 100 funkcji, które można podzielić na grupy: [6]

* zarządzanie wątkami,
* muteksy,
* zmienne warunkowe,
* synchronizacja

Podstawowym celem Pthreads jest przyspieszenie realizacji zadań. Porównując koszty tworzenia i zarządzania procesem, wątek może być utworzony przy minimalnym udziale zasobów systemu.

W powyższym projekcie technologia Pthreads została wykorzystana w celu zarządzania akceleratorami obliczeń, jakimi są karty graficzne. Bierze udział w procesie podziału danych i przygotowaniu ich do przesłania na zewnętrzne urządzenia. Pomaga również w końcowym procesie zarządzania wynikami obliczeń dostarczonymi do maszyny gospodarza (eng. host).

## OpenGL

OpenGL jest wieloplatformową technologią używaną przez różnorodne języki programowania umożliwiającą przetwarzanie obrazu dwu- i trójwymiarowego. Wykorzystywany jest do interakcji z procesorami graficznymi GPU aby uzyskać sprzętową akcelerację przetwarzanego obrazu. OpenGL jest stale rozwijaną technologią, która wspiera pojawiające się na rynku najnowsze karty graficzne wiodących producentów.

## Projekt aplikacji

Jednym z celów projektu było stworzenie aplikacji, która będzie nie tylko implementacją mechaniki dynamiki molekularnej, ale również otwartym oprogramowaniem łatwym w użytkowaniu i rozbudowie. Istotną cechą programu jest możliwość wizualizacji wyników, jak również śledzenia zmian zachodzących w strukturze w czasie rzeczywistym. W przypadku uruchamiania testów na klastrach obliczeniowych jest możliwość uruchomienia aplikacji wyłącznie w trybie tekstowym.

Aplikacja wykorzystuje funkcje pomocnicze oraz obsługę błędów pochodzące z biblioteki CUDA. Każdy z plików będący częścią tej biblioteki jest oznaczony na początku specjalnym tekstem. Wspomnienie źródła pochodzenia takiego kodu umożliwia jego wykorzystanie w dowolnym celu edukacyjnym nie łamiąc jednocześnie praw jego stosowania. Przykładowe oznaczenie:

/\*\*

\* Copyright 1993-2013 NVIDIA Corporation. All rights reserved.

\*

\* Please refer to the NVIDIA end user license agreement (EULA) associated

\* with this source code for terms and conditions that govern your use of

\* this software. Any use, reproduction, disclosure, or distribution of

\* this software and related documentation outside the terms of the EULA

\* is strictly prohibited.

\*

\*/

////////////////////////////////////////////////////////////////////////////////

// These are CUDA Helper functions for initialization and error checking

Projekt zakładał stworzenie spójnej aplikacji składającej się z jak najbardziej niezależnych od siebie elementów współpracujących ze sobą. Takie założenie umożliwiło realizację oprogramowania, składa się z poniższych komponentów:

* MolecularDynamics  
  Jest to główny komponent programu. Struktura wejściowe jest tutaj tworzona i inicjalizowana. Odpowiada również za wczytanie parametrów i uruchomienie symulacji.
* Simulation  
  Jest komponentem zajmujący się przetwarzaniem danych wejściowych oraz sterowaniem przebiegu symulacji na GPU. Przetwarzane są tutaj wszelkie parametry dodatkowe podawane przez użytkownika w trakcie uruchamiania programu. Mają one wpływ na typ symulacji oraz urządzenia, które zostaną wykorzystane do jej przeprowadzenia.
* Atom  
  Jest podstawowym elementem wykorzystywanym w dynamice molekularnej. Przechowuje szczegółowe informacje o położeniu aktualnym oraz początkowym, typie czy sile działającej na pojedynczy atom w strukturze molekularnej.
* Structure  
  Jest kombinacją setek, tysięcy lub nawet milionów atomów, tworzących całość jaką jest struktura molekularna. Przechowuje informacje o typie zastosowanego potencjału, sposobie przyłożenia siły zewnętrznej oraz wartości tej siły. Jest głównym parametrem wejściowym do symulacji.
* GpuHandler  
  Jest komponentem ułatwiającym zarządzanie urządzeniami GPU. Przechowuje identyfikatory dostępnych kart graficznych oraz ich ilość. Umożliwia ich inicjalizowanie i przygotowanie do obliczeń.
* GpuDisplay  
  Jest częścią aplikacji zarządzającą wizualizacją danych. Zajmuje się inicjalizacją środowiska OpenGL, umożliwia konwersję danych wykorzystywanych w obliczeniach do postaci umożliwiającej ich wyświetlenie oraz obsługą myszki i klawiatury. Zapewnia możliwość interakcji ze strukturą w czasie rzeczywistym oraz podgląd każdej części struktury w dowolnym powiększeniu.
* GpuKernel  
  Jest to komponent zajmujący się wykonywaniem obliczeń przy pomocy GPU. Zarządza pamięcią oraz danymi przesyłanymi na urządzenie. Uruchamia funkcje przeznaczone na urządzenie (eng. kernel). Umożliwia wykonywanie obliczeń w trybie wizualizacji lub trybie tekstowym.
* CudaHelpers  
  Jest miejscem do przechowywania elementów aplikacji wykorzystywanych przez wszystkie komponenty wchodzące w interakcję z GPU. Przechowuje dostępne kernele oraz funkcje zajmujące się wielowątkowym przetwarzaniem na kilku urządzeniach GPU.
* Log  
  Jest użytecznym komponentem, umożliwiającym tworzenie logów na standardowym wyjściu jak i w pliku tekstowym. Ułatwia śledzenie błędów gdy aplikacja niespodziewanie wyrzuci błąd lub przestanie działać.

## Implementacja

**Symulacja**

**GpuHandler**

**Wyniki**

**Wizualizacja**

**Brak wizualizacji**

**Kernel**

**Kernel**

Rys 3.2. Schemat przepływu danych w programie

## Optymalizacja

TODO

# Testy wydajności

## Testy sekwencyjne

TODO

## Testy zrównoleglenia

TODO

## Testy optymalizacji

TODO

## Test wielu GPU

TODO

# Model algorytm - urządzenie

## XXX

# Podsumowanie

# Bibliografia

[1] Tomasz Nowak, *Modelowanie defektów strukturalnych w skali nano z wykorzystaniem akceleratorów obliczeń GPGPU*

[2] Daniel Bachniak, *Modelowanie defektów strukturalnych w skali nano z wykorzystaniem metody statyki molekularnej w heterogenicznych architekturach sprzętowych*

[3] https://developer.nvidia.com (05.04.2014)

[4] http://www.nvidia.com/ (05.04.2014)

[5]https://computing.llnl.gov/ (05.04.2014)

[6] http://en.wikipedia.org/wiki/POSIX\_Threads (05.04.2014)

[7]http://wakespace.lib.wfu.edu/bitstream/handle/10339/38561/Proctor\_wfu\_0248M\_10427.pdf (10.08.2014)

[8] http://ti.odio.twiki.di.uniroma1.it/pub/CI/WebHome/ParMolDyn.pdf (28.08.2014

[9] http://en.wikipedia.org/wiki/Molecular\_dynamics

1. API – eng. application programming interface [↑](#footnote-ref-2)
2. Windows API – zestaw wbudowanych interfejsów programistycznych dostępnych dla systemów Microsoft Windows [↑](#footnote-ref-3)