AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE



Wydział Inżynierii Metali i Informatyki Przemysłowej



PROJEKT INŻYNIERSKI

pt.

„Modelowanie defektów strukturalnych w skali nano z wykorzystaniem akceleratorów obliczeń GPGPU”

Imię i nazwisko dyplomanta: **Tomasz Nowak**

Kierunek studiów: **Informatyka Stosowana**

Profil dyplomowania:  **Modelowanie i Technologie Informacyjne**

Nr albumu: **232187**

Promotor: **dr inż. Łukasz Rauch**

Podpis dyplomanta: Podpis promotora:

Kraków 2012

***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 ………….… Podpis dyplomanta…………….

**Spis treści**

[1. Wstęp 4](#_Toc345249444)

[2. Wprowadzenie 6](#_Toc345249445)

[2.1. Struktury krystalograficzne 6](#_Toc345249446)

[2.1.1. A1 – regularna powierzchniowo centrowana 6](#_Toc345249447)

[2.1.2. A2 – regularna przestrzennie centrowana 7](#_Toc345249448)

[2.2. Potencjały 8](#_Toc345249449)

[2.2.1. Lennard – Jones 9](#_Toc345249450)

[2.2.2. Sutton – Chen 10](#_Toc345249451)

[3. OpenCL jako narzędzie implementacyjne 12](#_Toc345249452)

[3.1. Architektura OpenCl 12](#_Toc345249453)

[3.2. Przenośność kodu źródłowego 14](#_Toc345249454)

[4. Oprogramowanie 15](#_Toc345249455)

[4.1. Solwer metody statyki molekularnej 15](#_Toc345249456)

[4.2. Implementacja struktur A1, A2 17](#_Toc345249457)

[4.3. Wizualizacja struktur 20](#_Toc345249458)

[4.4. Oprogramowanie dla GPU 22](#_Toc345249459)

[5. Analiza wyników 23](#_Toc345249460)

[5.1. Defekty strukturalne 23](#_Toc345249461)

[5.1.1. Struktury bez defektów 23](#_Toc345249462)

[5.1.2. Defekty struktury A2 26](#_Toc345249463)

[5.1.3. Defekty struktury A1 27](#_Toc345249464)

[5.2. Porównanie wydajności na różnych platformach testowych 27](#_Toc345249465)

[5.2.1. Platforma testowa 1 – GPGPU 29](#_Toc345249466)

[5.2.2. Platforma testowa 2 – PC 32](#_Toc345249467)

[5.2.3. Platforma testowa 3 – laptop 36](#_Toc345249468)

[5.3. Skalowalność rozwiązania 38](#_Toc345249469)

[6. Podsumowanie 42](#_Toc345249470)

[7. Bibliografia 43](#_Toc345249471)

# 1. Wstęp

Postęp technologiczny idzie nieustannie do przodu. Osiągnięcia dokonywane w różnych dziedzinach naszego życia wymagają od ludzi coraz większej wiedzy i umiejętności. Kiedyś komputery pomagały w rozwoju, dziś rozwój byłby bez nich niemal niemożliwy. Obliczenia wykonywane w inżynierii czy w medycynie, stają się na tyle skomplikowane i pracochłonne, że człowiek już nie jest w stanie poradzić sobie z nimi tylko przy pomocy kartki i ołówka. Wymusiło to zmianę kierunku rozwoju mikroprocesorów (CPU) w komputerach osobistych. Ewolucja w kierunku zwiększania ilości rdzeni stała się po pewnym czasie nieefektywna i niewydajna [1]. Zaczęto zmieniać architekturę, prześcigając się w „nanotechnologii”. Spowodowało to zapotrzebowanie na nowych specjalistów, którzy będą potrafili wykorzystać moc obliczeniową nowych urządzeń. Specjalistów potrafiących wykonywać obliczenia nie tylko sekwencyjnie, ale też równolegle wykorzystując nowe możliwości mikroprocesorów. Zaczęto łączyć wiele urządzeń tego samego typu w klastry obliczeniowe, które otworzyły przed ludźmi nowe możliwości. Obliczenia trwające kiedyś miesiąc, można było przeprowadzić w ciągu kilku godzin. Jednak rozmiar problemów wciąż się powiększał. Nie tak dawno zaczęto wykorzystywać technologie, które już od dawna były dostępne, do zupełnie innych celów niż pierwotnie zakładano. Były to karty graficzne (GPU), których nowym zadaniem, oprócz wyświetlania obrazu, stało się wykonywanie obliczeń. Ze względu na konieczność wyświetlania dużych ilości danych w jak najkrótszym czasie stały się idealnym narzędziem do wykonywania skomplikowanych i czasochłonnych obliczeń, pomijając już wyświetlanie obrazu. Zaczęto przystosowywać te urządzenia do nowych celów tak, aby uprościć korzystanie z nich i zwiększyć wydajność pracy. Powstały również specjalne serie GPU przeznaczone tylko i wyłącznie do obliczeń, a ich podstawowa funkcja wyświetlania obrazu została usunięta. Podobnie jak CPU, GPU zaczęto łączyć w klastry obliczeniowe tworzące tzw. GPGPU (ang. General – Purpose Graphics Processing Unit). Jest to technologia stosunkowo nowa i wielu programistów dopiero odkrywa sekrety i możliwości tych urządzeń. Nie można całkowicie zamienić CPU na mikroprocesory GPU, ale łącząc obydwie technologie można osiągnąć znacznie więcej. GPGPU już jest, a na horyzoncie pojawiają się komputery kwantowe, które całkowicie mogą zmienić nasze spojrzenie na komputery i obliczenia. A może to być już niedaleka przyszłość.

Celem niniejszej pracy jest wykonanie analizy wydajności procesorów GPU z wykorzystaniem metod statyki molekularnej. Zaimplementowane zostaną struktury krystalograficzne występujące w metalach, które stanowić będą dane wejściowe do algorytmów stabilizacji nanostruktur w oparciu o różne potencjały. W tym celu wykorzystane zostaną dwa potencjały tj. Lennarda-Jonesa oraz Suttona-Chena. Praca zakończy się interpretacją uzyskanych wyników oraz podsumowaniem.

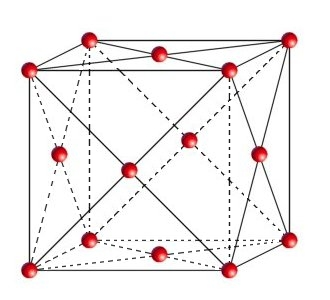
# 2. Wprowadzenie

## 2.1. Struktury krystalograficzne

Struktura krystalograficzna jest normalną formą, którą przyjmują metale schłodzone do temperatury poniżej ich temperatury topnienia. Gdy substancja staje się ciałem stałym podczas chłodzenia ze stanu ciekłego, może zawierać amorficzne lub krystalograficzne struktury. Pierwsze z nich są losowe. Drugie są normalną formą w metalach poniżej temperatury topnienia. Typowymi przykładami amorficznych są szkło oraz związki organiczne. Oprócz szkła, prawie wszystkie materiały ceramiczne mają strukturę krystalograficzną. Istnieją specjalne stopy metali, w których mogą występować struktury amorficzne, poddawane ekstremalnie szybkiemu chłodzeniu. W wyniku tego powstają tak zwane amorficzne metale lub też metaliczne szkło. [2]

Symulacje wykonywane w pracy będą wykorzystywały dwie zaimplementowane nanostruktury: A1 oraz A2. Zostaną one opisane w podrozdziałach 2.1.1. i 2.1.2.

### 2.1.1. A1 – regularna powierzchniowo centrowana

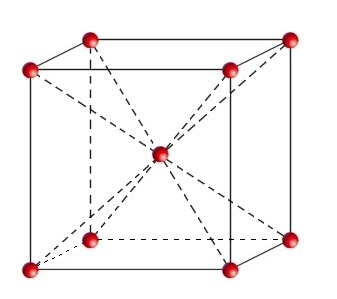


Rys 2.1. Struktura krystalograficzna A1 [3]

*Struktura krystalograficzna A1* - regularna powierzchniowo centrowana (rys 2.1). Struktura składa się z 12 atomów. Rozmieszczone są one w 8 rogach oraz w centralnych punktach ścian sześcianu. Jest to struktura spotykana metalach takich jak: Cu, Ag, Au, Al. Oprócz metali występuje również w gazach: Ne, Ar, Kr, Xe. Każdy atom posiada 12 najbliższych sąsiadów. Jest to najbardziej zwarta struktura. Nazywana jest często *fcc* (ang. face-centered cubic). [4]

Pozycje atomów: 0, 0, 0; ½, ½, 0; 0, ½, ½; ½, 0, ½;

### 2.1.2. A2 – regularna przestrzennie centrowana



Rys 2.2. Struktura krystalograficzna A2 [3]

*Struktura krystalograficzna A2* - regularna przestrzennie centrowana (rys. 2.2). Struktura składa się z 9 atomów. Posiada atomy rozmieszczone w 8 rogach oraz centralnym punkcie sześcianu. Jest to spotykana metalach takich jak: Fe, Mo, W, Na. Każdy atom posiada 8 najbliższych sąsiadów oraz 6 bliskich na odległości 15% większej. Jest często nazywana *bcc* (ang. body-centered cubic). [4]

Pozycje atomów: 0, 0, 0; ½, ½, ½;

## 2.2. Potencjały

Potencjał energetyczny wpływa na pozycję, układ lub umiejscowienie atomów w przestrzeni. Jest ponadto związany energią przyciągania i odpychania między obiektami.

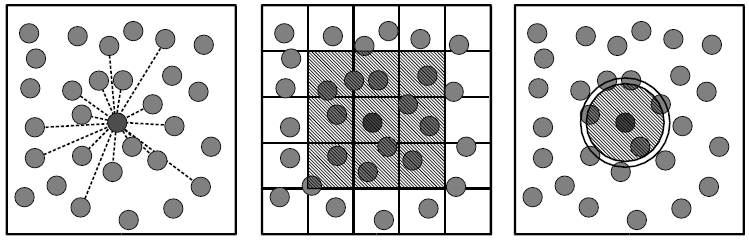
(1)

Poniższe równanie pokazuje przykład potencjału grawitacyjnego:

(2)

Każdy obiekt unoszący się w polu, będąc w spoczynku posiada zmagazynowany potencjał i dlatego ma energię do wykonania pracy po uwolnieniu. Przykładem może być kulka umieszczona na pewnej wysokości. Upuszczona zostanie przyciągnięta przez grawitację i energia potencjalna jest wtedy zamieniana na energię kinetyczną w czasie spadania. Warto zauważyć, że w czasie konwersji energii nie zostaje ona utracona czy też nie jest tworzona nowa (prawo zachowania energii). [5]

Energia na poziomie molekularnym jest zmagazynowana w wiązaniach atomowych takich jak: wiązania kowalencyjne, wiązania elektrostatyczne czy wiązania jądrowe. W czasie egzotermicznych reakcji wiązania pękają, a nowe są tworzone. Protony i elektrony przechodzą z wysokiego stanu energetycznego do niskiego. W tym czasie energia potencjalna jest zamieniana na kinetyczną, a wydzielane jest ciepło. Odwrotnie dzieje się w czasie reakcji endotermicznych. [5]

Do prowadzenia obliczeń można stosować różne podejścia.

Rys 2.3. a) Obliczanie odległości. b) Automaty komórkowe. c) Lista sąsiedztwa. [6]

W dalszej części przedstawione zostaną dwa potencjały Lennard’a – Jones’a oraz Sutton’a – Chen’a. Zakładają one dwa różne podejścia, co dobrze odzwierciedlają wyniki obliczeń z użyciem każdego z algorytmów.

### 2.2.1. Lennard – Jones

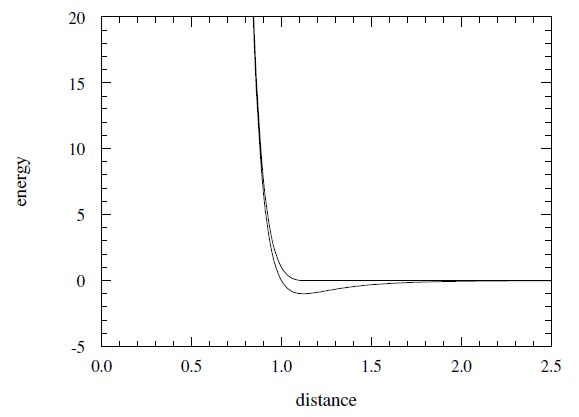
Jednym z potencjałów określających interakcje pomiędzy atomami jest potencjał Lennard’a –Jones’a. Charakteryzuje się silnym oddziaływaniem między cząsteczkami będącymi blisko siebie, a słabym dla cząsteczek bardziej oddalonych. Aby utrzymać obliczenia na rozsądnym poziomie, interakcja jest zredukowana do krótkiego dystansu. Typowy promień odcięcia wynosi rc= 2.5 jednostki. Odziaływanie to tylko 0,016 w najniższym punkcie wykresu. Potencjał LJ w oryginalnej postaci może występować w ciałach stałych, cieczach i gazach.[5]

(3)

**V -** jest wewnątrzatomowym potencjałem pomiędzy dwiema cząstkami

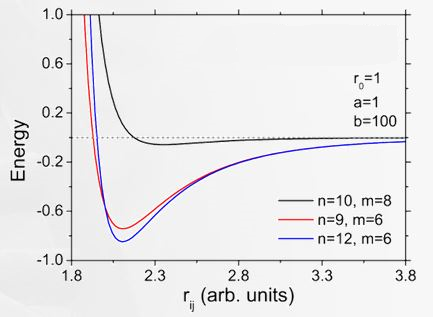
**ℇ** - reprezentuje odległość od najniższego punktu wykresu i określa jak silne jest oddziaływanie

**σ –** oznacza dystans pomiędzy cząstkami, na którym potencjał wynosi 0; daje wyobrażenie jak blisko znajdują się dwa niepowiązane atomy

**r** – jest to dystans, jaki dzieli obydwie cząstki (jest mierzony od centralnego cząstek)

Rys 2.4. Potencjał Lennard’a – Jones’a w bezwymiarowych jednostkach. [5]

### 2.2.2. Sutton – Chen

Potencjał Sutton’a-Chen’a bardzo dobrze opisuje metale o strukturze A1 (fcc). Został przystosowany do prowadzenia symulacji z wykorzystaniem komputera na nanostrukturach składających się z dużej liczby atomów.

Rys. 2.5. Potencjał Sutton’a-Chen’a [8]

Przedstawia się go za pomocą równania:

(4)

(5)

(6)

**ℇ** - reprezentuje odległość od najniższego punktu wykresu i określa jak silne jest oddziaływanie

**a** – jest to stała równowagi i ma wymiar odległości

**n, m** – dodatnie liczby całkowite spełniające zależność n > m

**r** – jest to dystans, jaki dzieli obydwie cząstki (jest mierzony od centralnego cząstek)

**c -**  stała wyznaczana na podstawie parametrów równowagi

**ρ** – funkcja gęstości

Parametry są wyznaczane na drodze eksperymentalnej oraz korzystając ze stałych parametrów równowagi dla poszczególnych pierwiastków.[7]

Potencjał SC jest tak dobrze przystosowany do implementacji ze względu na dużą liczbę parametrów, którymi można manewrować w celu uzyskania zadowalających rezultatów. Wyniki uzyskane przy pomocy tego równania mają odpowiednie odzwierciedlenie w wynikach uzyskanych na drodze eksperymentalnej.[7]

Potencjały zarówno SC jak i LJ dawały bardzo zbliżone wykresy (Rys. 2.4), (Rys. 2.5).

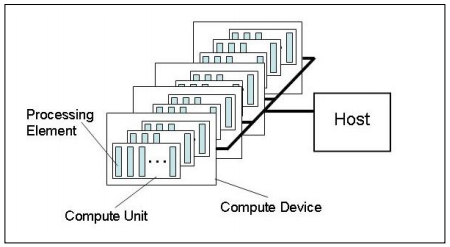
# 3. OpenCL jako narzędzie implementacyjne

## 3.1. Architektura OpenCl

OpenCL jest otwartym standardem programowania heterogenicznych kolekcji urządzeń złożonych z CPU i GPU oraz innych przeznaczonych do prowadzenia obliczeń, będących częścią jednej maszyny. OpenCL to framework do programowania równoległego zawierający język, API, biblioteki oraz system uruchomieniowy. Dzięki temu narzędziu można uruchamiać programy pisane pod GPU bez konieczności mapowania ich algorytmów na wykorzystujące grafikę 3D API takie jak OpenGL czy DirectX. OpenCL skłąda się z modelu hierarchicznego:

* Model Platformy
* Model Pamięci
* Model Wykonania
* Model Programowania

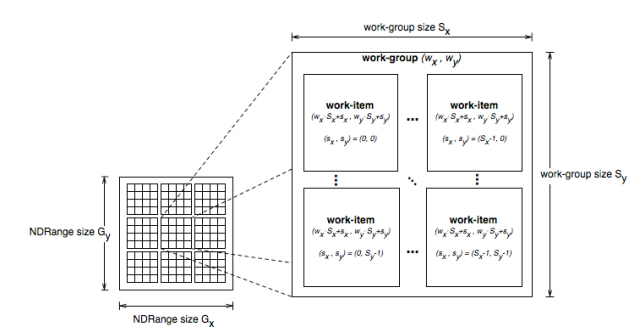
*Model Platformy*

Składa się z hosta podłączonego do jednego lub więcej urządzeń obsługujących OpenCL.. Urządzenia są podzielone na jedną lub więcej jednostek obliczeniowych (CUs), które później dzielą się na jednostek przetwarzających (PEs). [9]

Rys. 3.1. Model Platformy w OpenCL [9]

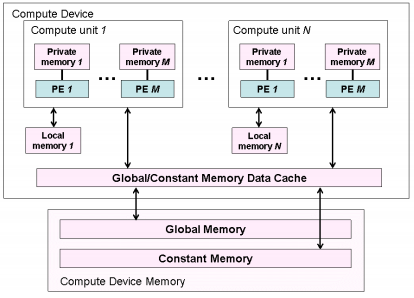
*Model Wykonania*

Uruchomienie programu napisanego w OpenCL przebiega w dwóch częściach: kernel’e, które są wykonywane na jednym lub wielu urządzeniach; program host’a uruchamiany na urządzeniu gospodarza. Instancja kernel’a wykonywana jest dla każdego indeksu przestrzeni jakie zostały zdefiniowane. Nazywają się work-item’ami. Work-item’y są organizowane w work-group’y.

Rys 3.2. Model Wykonania w OpenCL [9]

*Model Pamięci*

Work-item’y mają dostęp do czterech odrębnych regionów pamięci: pamięć globalna, pamięć stała, pamięć lokalna oraz pamięć prywatna. Aplikacja uruchomiona przez host’a używa APL OpenCL’a, aby stworzyć obiekty w pamięci globalnej i kolejkuje komendy, które będą na tych obiektach operowały. Pamięć host’a i pamięć urządzeń są niezależne od siebie. Aby dane pojawiły się na urządzeniu należy je jawnie przekopiować lub zmapować. Aby zapisać wyniki uzyskane na urządzeniu również należy je przekopiować, ale tym razem w drugą stronę, czyli z urządzenia do host’a. [9]



Rys. 3.3. Model Pamięci w OpenCL [9]

*Model Programowania*

OpenCL wspiera dwa rodzaje zrównoleglenia: zrównoleglenie ze względu na dane oraz zrównoleglenie ze względu na zadanie. Można również mieszać obydwie metody. Pierwszy z nich opiera się na sekwencji instrukcji obliczeniowych przekazywanych do wielu elementów z pamięci. Przestrzeń indeksowania jest związana z Modelem Wykonywania definiującym work-item’y. Zrównoleglenie ze względu na zadanie polega na uruchamianiu kernel’i niezależnie od indeksu w pamięci. Jest to odpowiednikiem uruchomienia kernel’a na work-group’ie jednostki obliczeniowej zawierającej pojedynczy work-item. [9]

Istotnym elementem programowania równoległego jest synchronizacja. OpenCL udostępnia synchronizację pomiędzy work-item’ami za pomocą barier. Nie ma jednak mechanizmu synchronizacji pomiędzy work-gourp’ami.[9]

## 3.2. Przenośność kodu źródłowego

OpenCL został zaprojektowany tak, aby był przenośny. Wykorzystuje język bazujący na C99. Operacje zmiennoprzecinkowe opierają się o standardy IEEE-754 oraz IEEE-754-2008.

Jednak różnorodność architektur obliczeniowych może oznaczać, że dana pętla może wykonywać się z zadowalającą szybkością na CPU, ale bardzo słabo na GPU. Procesory CPU zaprojektowane są tak, aby dobrze działać na opóźnienia we wrażliwych algorytmach, jednak GPU może napotykać na bardzo duże opóźnienia nawet o rzędy wielkości gorzej. Deweloperzy piszący przenośny kod muszą się liczyć z różnorodnością sprzętu. Odpowiedni dobór work-item’ów może być kluczowy, ale jednocześnie wymaga miesięcy czy nawet lat doświadczenia.

Trochę uwagi należy poświęcić również sposobowi adresowania w rejestrze. W większości maszyn obsługujących OpenCL zastosowane jest adresowanie typu little-endian, jednak spotyka się również wykorzystujące big-endian. Programiści muszą przetestować swoje kernel’e na obydwu powyższych typach. Deweloper musi określić, wykorzystując język OpenCL C, czy dane korzystają z adresowania hosta czy urządzenia OpenCL. Dzięki temu OpenCL będzie mógł dokonać odpowiedniej konwersji, wczytać dane i wykonać odpowiednie operacje.

# 4. Oprogramowanie

Projekt przedstawiony w niniejszej pracy wykorzystuje fragmenty pracy magisterskiej Daniela Bachniaka [10]. Zajmował się on modelowaniem defektów strukturalnych w skali nano na heterogenicznych architekturach sprzętowych. Jako struktury bazowe wykorzystywał regularne siatki atomów bez konkretnych struktur krystalograficznych. Użycie struktur realnie występujących w materiałach będzie jednym z moich celów pracy. Daniel Bachniak zaimplementował również solwery wykorzystujące potencjały Lennard’a-Jones’a oraz Sutton’a-Chen’a, które zostały opisane w rozdziale 2. W kolejnych rozdziałach postaram się przybliżyć zasadę działania solwerów oraz własną implementację struktur krystalograficznych wraz z wizualizacją.

## 4.1. Solwer metody statyki molekularnej

Program Daniela Bachniaka rozpoczynał się od wygenerowania struktury. Były dostępne dwa rodzaje: struktura 2D oraz 3D. W trakcie generowania atomy przyjmowały różne stany. Można było unieruchomić część z nich (ich pozycja nie zmieniała się w czasie minimalizacji potencjału), ukryć je (nie brały udziału w obliczeniach) lub ustawić stan, w którym normalnie brały udział w obliczeniach i zmieniały swoją pozycję. Kolejnym krokiem algorytmu było uruchomienie funkcji ‘*reducePotential()’,* zajmującej się minimalizacją potencjału. W podstawowym założeniu funkcja ta zawierała pętlę, która wykonywała się tyle razy, ile było atomów w strukturze robiąc obliczenia nowego położenia dla każdego z nich (funkcja ‘*calculateAtomNewPosition(Atom \* atom)*’. Na końcu pozycja każdego atomu była aktualizowana

void Structure::reducePotential()

{

for(int i=0; i<AtomsCount; ++i)

{

if(Atoms[i]->status != 2 // atom unieruchomiony

&&

Atoms[i]->status != -99) // atom usuniety

calculateAtomNewPosition(Atoms[i]);

}

for(int i=0; i<AtomsCount; ++i)

{

Atoms[i]->X += Atoms[i]->Dx;

Atoms[i]->Y += Atoms[i]->Dy;

Atoms[i]->Z += Atoms[i]->Dz;}

Listing 4.1. Funkcja „reducePotential()” [10]

Przed obliczeniem nowej pozycji następował wybór potencjału, który miał zostać użyty. Nowa pozycja atomu była ustalana przez poniższą funkcję.

void Structure::calculateAtomNewPosition(Atom \* atom)

{

/\* wyliczenie wartości sumy   
 sił oddziaływania z wszystkimi atomami \*/

if(PotentialType == LENNARD\_JONES)

{

calculateLennard\_jonesGradient(atom, this->CutDistance, forceGradient);

}

else if(PotentialType == SUTTON\_CHEN)

{

calculateSutton\_chenGradient(atom, this->CutDistance, forceGradient);

}

...

}

Listing 4.2. Funkcja „calculateAtomNewPosition(Atom \*atom)” [10]

Następnym krokiem było wprowadzenie defektu do struktury. Należało wtedy ponownie zminimalizować potencjał wykorzystując powyższe funkcje. Po zakończeniu obliczeń program kończył działanie zapisując nowo powstałą strukturę do pliku.

Kluczowym punktem programu było obliczanie sił międzyatomowych. Obydwa potencjały są liczone z podobnego algorytmu. Do funkcji przekazywany jest atom, dla którego mają zostać wykonane obliczenia. Na podstawie ustalonego promienia odcięcia (opis w części teoretycznej, rozdział 2.2.1) tworzona jest lista sąsiadujących atomów, które będą uwzględniane w obliczeniach. Istotny jest również kierunek siły, wyznaczany w z zależności czy atom jest odpowiednio blisko lub daleko od bazowego. Wartości są obliczane osobno w każdym kierunku X, Y, Z.

/\* rozbicie siły na 3 składowe \*/

double distanceX = forceAtom->X - atom->X;

double distanceY = forceAtom->Y - atom->Y;

double distanceZ = forceAtom->Z - atom->Z;

forceGradient[0] += - (distanceX / distance) \* force;

forceGradient[1] += - (distanceY / distance) \* force;

forceGradient[2] += - (distanceZ / distance) \* force;

...

}

Listing 4.3. Obliczanie końcowych wartości sił [10]

## 4.2. Implementacja struktur A1, A2

Struktury używane w pracy Daniela Bachniaka były dobrą bazą do stworzenia algorytmów opisanych we wcześniejszym rozdziale. Jednak, aby obliczenia były bliższe rzeczywistości zaimplementowano dwie ze struktur występujących w metalach.

Obydwie struktury zaczynają się od stworzenia pojedynczego sześcianu z odpowiednio ułożonymi atomami w przestrzeni.

Podstawowa implementacja struktury A1 wygląda następująco.

// set of core A1 atoms

// ignore 0 index for better numeration

Atom \*\* coreAtoms;

coreAtoms = new Atom \*[15];

// center

coreAtoms[11] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + 0 \* distanceBetweenAtoms, zPos + 0 \* distanceBetweenAtoms, status);

coreAtoms[12] = new Atom(xPos + 0 \* distanceBetweenAtoms, yPos + 0 \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[13] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + 0 \* distanceBetweenAtoms, zPos + 0 \* distanceBetweenAtoms, status);

coreAtoms[14] = new Atom(xPos + 0 \* distanceBetweenAtoms, yPos + 0 \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

// bottom

coreAtoms[1] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[2] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[3] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

coreAtoms[4] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

coreAtoms[5] = new Atom(xPos + 0 \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + 0 \* distanceBetweenAtoms, status);

// top

coreAtoms[6] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[7] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[8] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

coreAtoms[9] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

coreAtoms[10] = new Atom(xPos + 0 \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + 0 \* distanceBetweenAtoms, status);

Listing 4.4. Podstawowa implementacja struktury A1

Podstawowa implementacja struktury A2 wygląda następująco.

// set of core A2 atoms

// set of core starting atoms

Atom \*\* coreAtoms;

coreAtoms = new Atom \*[9];

// center

coreAtoms[0] = new Atom(xPos + 0 \* distanceBetweenAtoms, yPos + 0 \* distanceBetweenAtoms, zPos + 0 \* distanceBetweenAtoms, status);

// bottom

coreAtoms[1] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[2] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[3] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

coreAtoms[4] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + (-0.5) \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

// top

coreAtoms[5] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[6] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + (-0.5) \* distanceBetweenAtoms, status);

coreAtoms[7] = new Atom(xPos + (0.5) \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

coreAtoms[8] = new Atom(xPos + (-0.5) \* distanceBetweenAtoms, yPos + (0.5) \* distanceBetweenAtoms, zPos + (0.5) \* distanceBetweenAtoms, status);

Listing 4.5. Podstawowa implementacja struktury A2

Kolejnym krokiem było przystosowanie struktur do obliczeń i generowania w nich defektów. W tym celu powieliłem podstawowe struktury w każdym kierunku X, Y, Z tak, aby stworzyły sześciany z upakowanymi wewnątrz mniejszymi wcześniej przygotowanymi strukturami. Powiększyła się jednocześnie liczba atomów. Bo bliższej analizie okazała się ona być zbyt duża biorąc pod uwagę rozmiar aktualnie testowanych struktur. Problemem były nakładające się na siebie atomy przy dodawaniu kolejnej struktury obok już istniejącej. Rozwiązaniem było dodawanie tylko tych atomów, których aktualnie brakowało, aby powstała kolejna kopia powielanej struktury. Poniżej, zaprezentowany zostanie schemat tworzenia struktur

Schemat końcowy struktury A1.

for (int i=1 ; i<structureMultiplication ; i++) {

// UP: 6, 7, 8, 9, 10 | 11, 12, 13, 14 [Y++]

// DOWN: 1, 2, 3, 4, 5 | 11, 12, 13, 14 [Y--]

// LEFT: 1, 4, 6, 9, 11 | 5, 10, 12, 14 [X--]

// RIGHT: 2, 3, 7, 8, 13 | 5, 10, 12, 14 [X++]

// BACKWORD: 3, 4, 8, 9, 14 | 5, 10, 11, 13 [Z++]

// FORWARD: 1, 2, 6, 7, 12 | 5, 10, 11, 13 [Z--]

// UP&LEFT + DOWN&LEFT + UP&RIGHT + DOWN&RIGHT

for (int j=0 ; j<structureMultiplication -1; j++)

{

// XxY

// ZxY

// ZxX

for (int k=1 ; k<structureMultiplication ; k++)

{

// Z--

// Z++

}

}

}

Listing 4.6. Schemat powielania struktury A2

Schemat końcowy struktury A2.

for (int i=1 ; i< structureMultiplication ; i++)

{

// UP: 0, 5, 6, 7, 8 [Y++]

// DOWN: 0, 1, 2, 3, 4 [Y--]

// LEFT: 0, 1, 4, 5, 8 [X--]

// RIGHT: 0, 2, 3, 6, 7 [X++]

// BACKWORD: 0, 3, 4, 7, 8 [Z++]

// FORWARD: 0, 1, 2, 5, 6 [Z--]

// UP&LEFT + DOWN&LEFT + UP&RIGHT + DOWN&RIGHT

for (int j=0 ; j<structureMultiplication -1; j++)

{

// XxY

// ZxY

// ZxX

for (int k=1 ; k<structureMultiplication ; k++)

{

// Z--

// Z++

}

}

}

Listing 4.7. Schemat powielania struktury A2

Schematy tworzenia struktur są podobne. Określenia: UP, DOWN, LEFT itd. określają kierunek, w którym aktualnie są powielane atomy. Liczby są indeksami atomów, które aktualnie są powielane. Oznaczenia w kwadratowych nawiasach pokazują kierunek i znak, w którym kierunku powielenie jest wykonywane. W kolejnej części uzupełniane brakujące atomy w płaszczyznach: XxY, ZxY oraz ZxX. Ostatnim krokiem jest wypełnienie w kierunku Z. Listingi 4.6 oraz 4.7 są bardzo podobne. Jedyną różnicą są współrzędne atomów, które są w każdym kroku powielane. Sprawia to, że po zaimplementowaniu innych podstawowych struktur w łatwy sposób można wykorzystać powyższe schematy do ich powielenia.

W ten sposób przygotowane dane do obliczeń zostały zapisane do pliku tekstowego w formacie, który jest kompatybilny i wspierany przez program Daniela Bachniaka. Można, zatem używać zamiennie danych wejściowych. Plik tekstowy składa się ze ściśle określonych elementów.

atoms count:

2

atoms x positions:

-1.25

1.25

atoms y positions:

-1.25

-1.25

atoms z positions:

-1.25

-1.25

atoms status:

0

0

Listing 4.8. Plik tekstowy z danymi wejściowymi

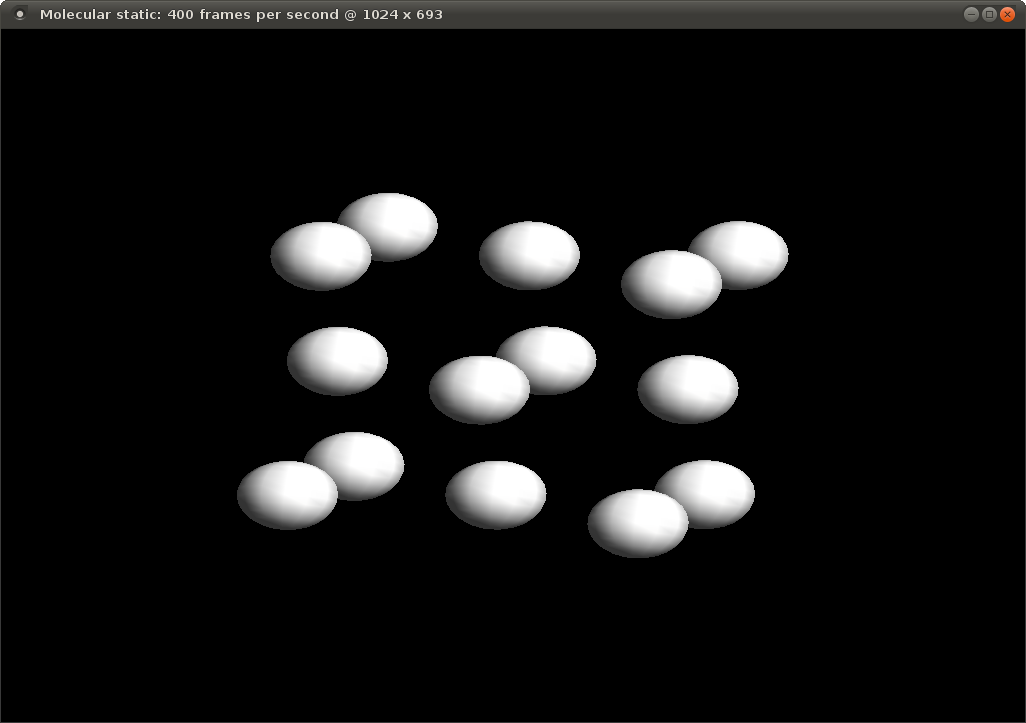
Powyższe struktury mają odzwierciedlenie w rzeczywistości. Nie spotyka się jednak czystych struktur A1 lub A2. Zawsze mamy do czynienia z domieszkami czy wtrąceniami, które je zaburzają, przez co stają się nie tak idealne i regularne.

W kolejnym rozdziale opisana zostanie wizualizacja powstałych struktur.

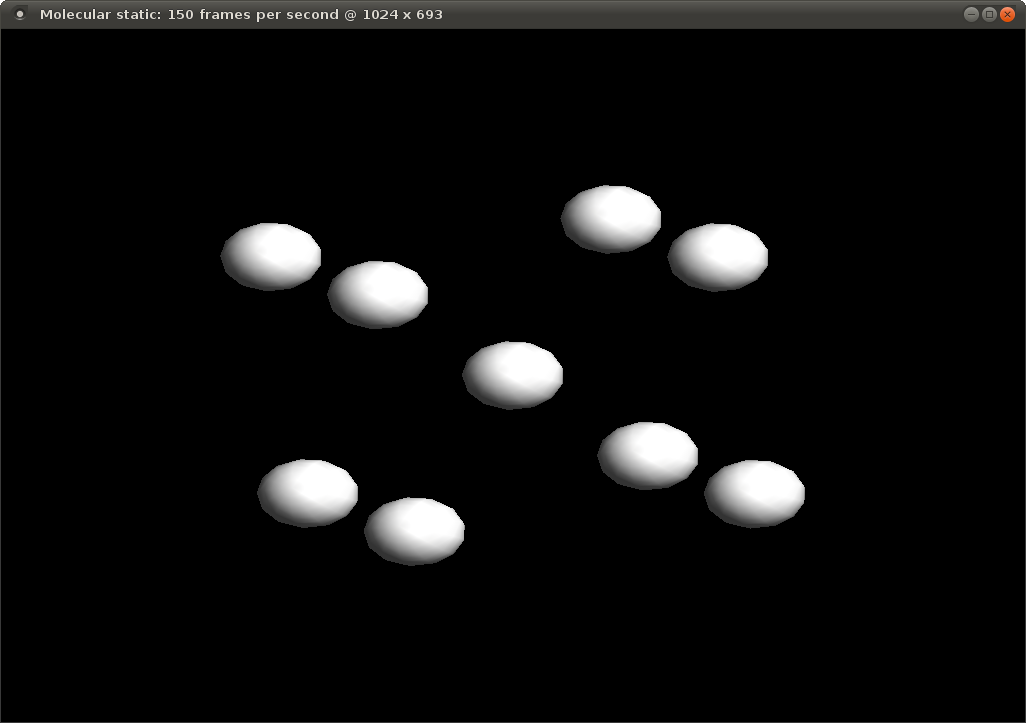
## 4.3. Wizualizacja struktur

Jak wspomniano we wcześniejszym rozdziale, rozwiązanie generowania danych jest przechowywane w kompatybilnym formacie z programami Daniela Bachniaka. Aby zwizualizować moje struktury wykorzystałem fragmenty jego kodu. Jest to napisany w OpenGL’u silnik wyświetlający atomy w przestrzeni 3D (widok z boku daje efekt wizualizacji 2D). Do wyświetlenia potrzebne są dane takiej jak: ilość atomów, oraz współrzędne X, Y, Z każdego atomu jak i jego status.

Dwie pierwotne struktury przedstawiają rysunki 4.1, 4.2.

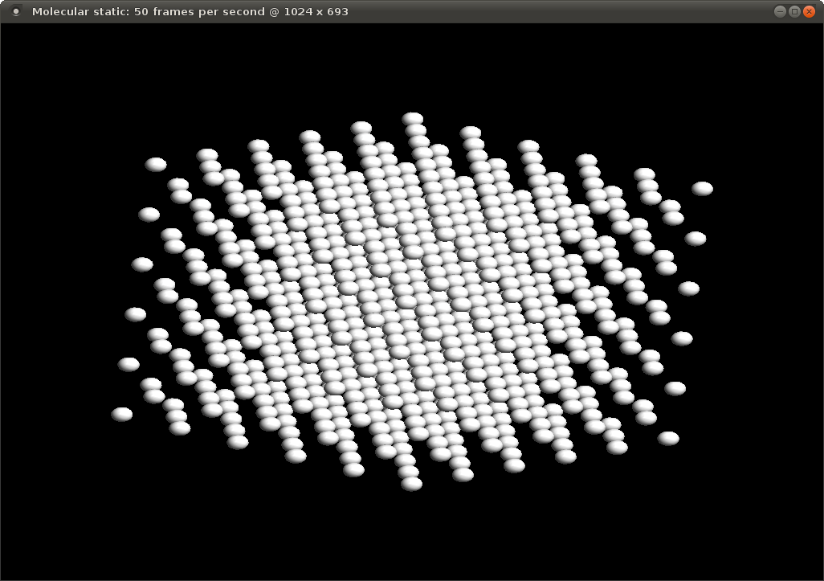


Rys. 4.1. Struktury A1 – pojedyncza

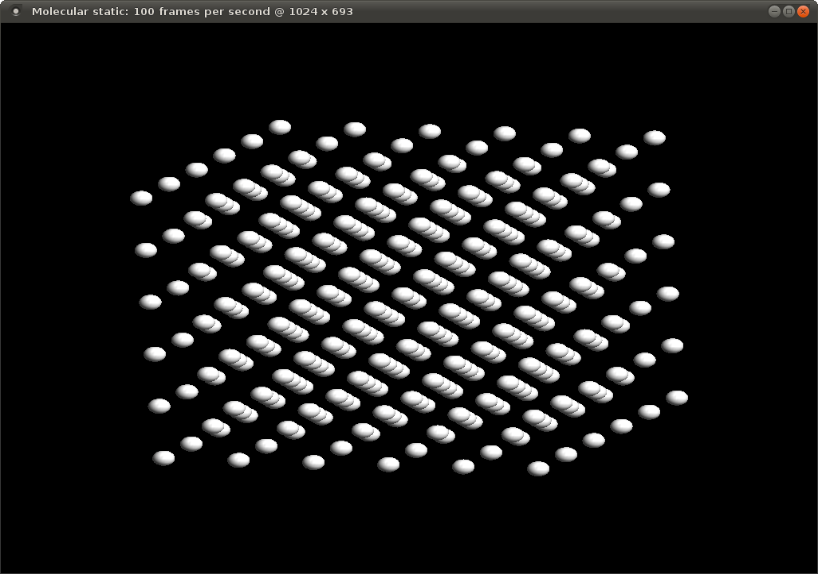


Rys. 4.2. Struktura A2 - pojedyncza

Po zastosowaniu kilkukrotnego powielenia struktur w każdym kierunku otrzymujemy efekt przedstawiony na rysunku 4.3, 4.4.



Rys. 4.3. Struktura A1 - powielona



Rys. 4.4. Struktura A2 – powielona

Wizualizacja w projekcie jest o tyle istotna, że bardzo łatwo będzie można zaobserwować powstałe defekty materiału. Zarówno defekty jak i obliczenia na GPU czy przedstawione są w kolejnym rozdziale. Analizie poddano rezultaty z różnych architektur sprzętowych, a następnie przedstawiono jak wygląda skalowalność algorytmów.

## 4.4. Oprogramowanie dla GPU

Do korzystania superkomputerów czy klastrów obliczeniowych z reguły wykorzystuje się terminal. Moje obliczenia będą wykonywane na komputerach, do których dostęp uzyskiwany jest zdalnie z wykorzystaniem SSH, pracując właśnie w terminalu. W związku z tym oprogramowanie dla tych komputerów przygotowane jest tak, aby korzystało się najwygodniej. Kolejnym powodem było mniejsze obciążenie karty graficznej w czasie obliczeń, gdy nie musi ona zajmować się wyświetlaniem obrazu. Z drugiej strony najnowsze karty graficzne GPGPU mają usuniętą ich pierwotną funkcję, jaką było wyświetlanie grafiki. Dlatego takie rozwiązanie jest bardziej uniwersalne.

Aby uruchomić program należy podać kilka istotnych parametrów. Jest możliwy wybór pliku z danymi wejściowymi, ustalenie ilości iteracji, zdefiniowanie wielkości GlobalWorkSize (GWS) oraz LocalWorkSize (LWS). Solwer należy umieścić w odpowiednim katalogu, aby został wykorzystany.

Usage:

./program [daneWejsciowe.txt] [ilośćIteracji] [GWS] [LWS]

Listing 4.9. GPGPU – użycie programu

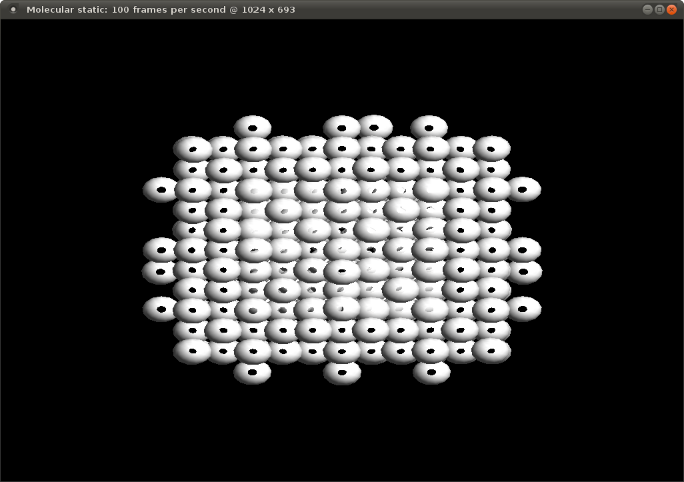
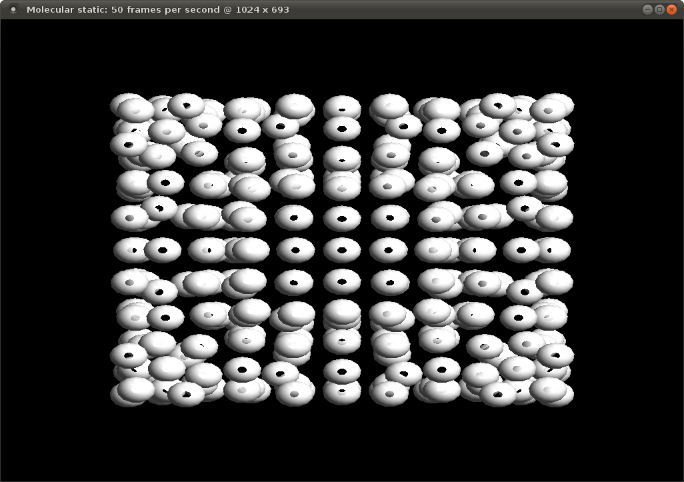
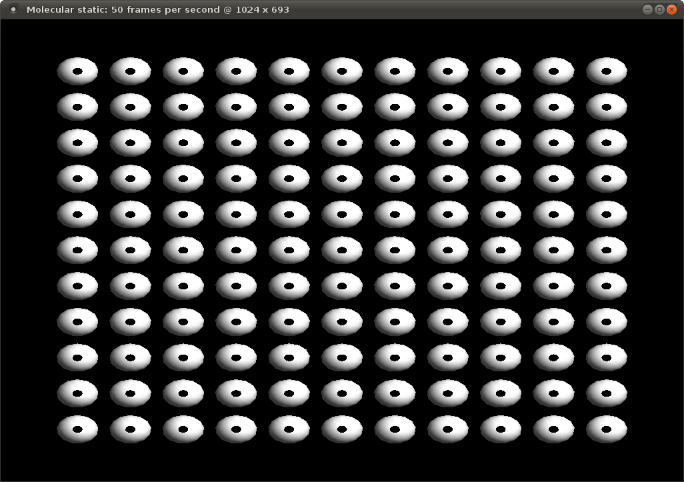
# 5. Analiza wyników

## 5.1. Defekty strukturalne

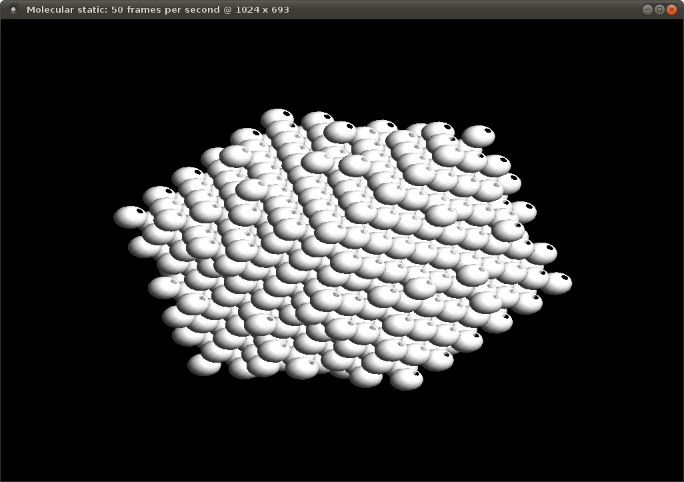
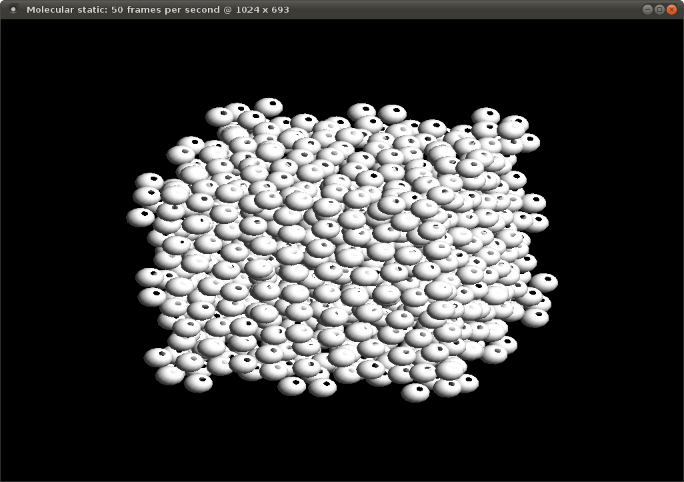
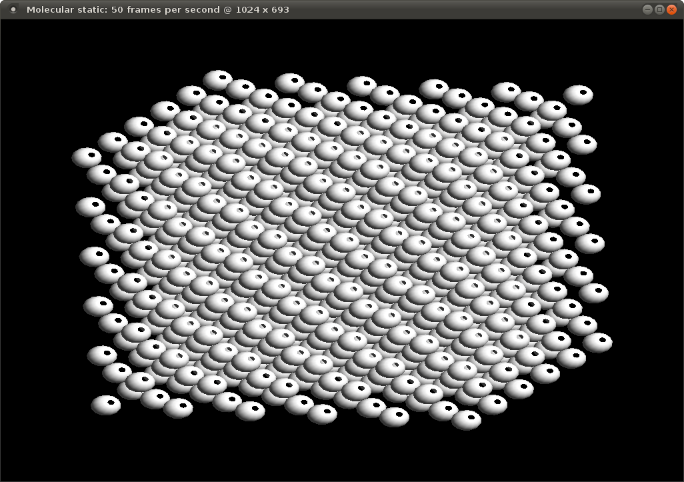
W pierwszej kolejności przedstawiony zostanie wpływ defektów na struktury A1 oraz A2, których implementacja została przedstawiona w poprzednim rozdziale. Zamieszczone zostaną wizualizacje wyników przed i po każdym teście. Dla porównania przedstawione zostaną wyniki bez defektów, aby pokazać zachowanie atomów w strukturach bez defektów. Każda z poniższych symulacji została przeprowadzona z wykorzystaniem potencjału Lennard’a – Jones’a.

### 5.1.1. Struktury bez defektów

Dla każdej struktury zostały przygotowane 2 lub 3 zrzuty ekranu z programu. Każdy zrzut ekranu reprezentuje kolejne zmiany struktur wraz z upływem czasu. Można zaobserwować stan początkowy, zachowanie atomów w trakcie minimalizacji potencjału oraz efekt końcowy działania solwera.

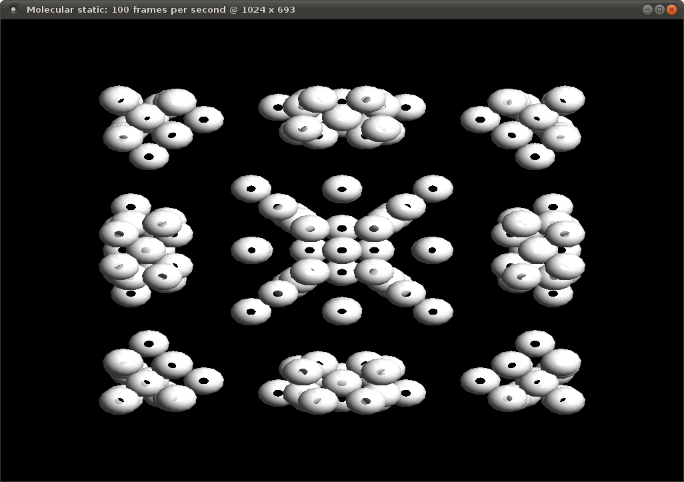
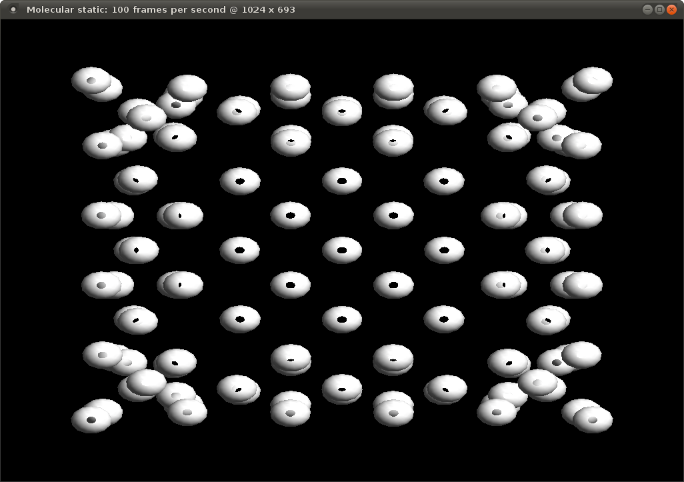
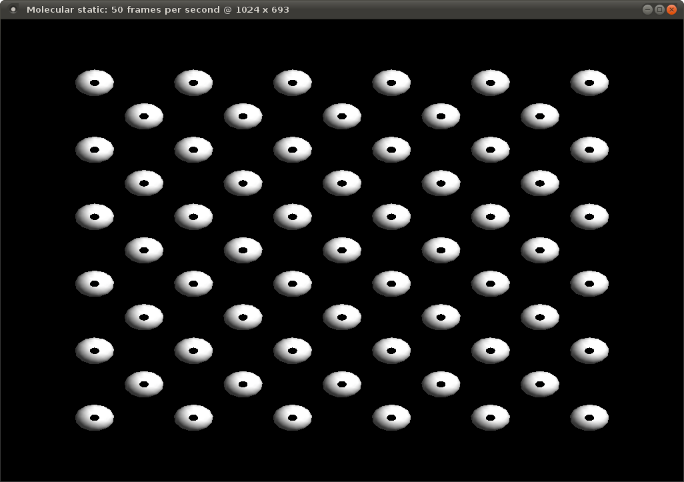


Rys. 5.1. A1 bez defektów – widok od przodu – działanie algorytmu z potencjałem Lennard’a-Jones’a

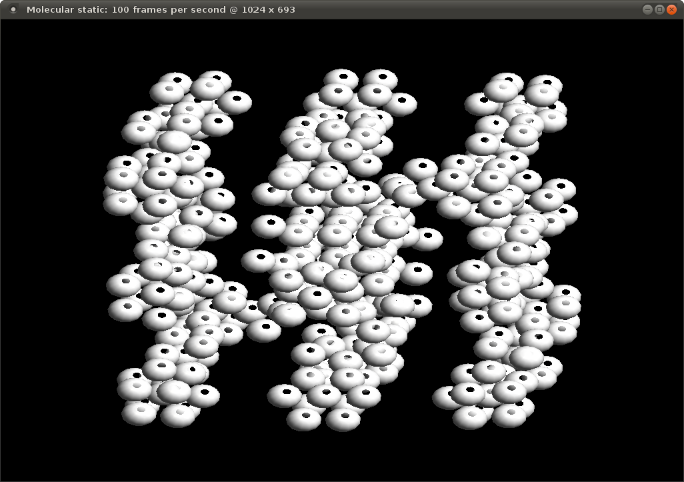
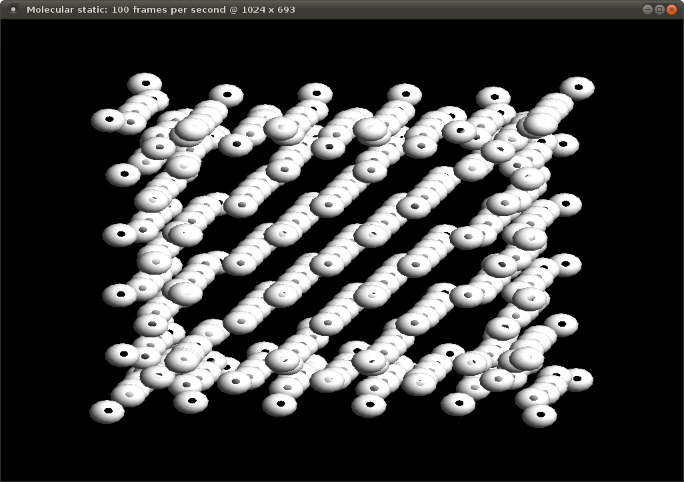
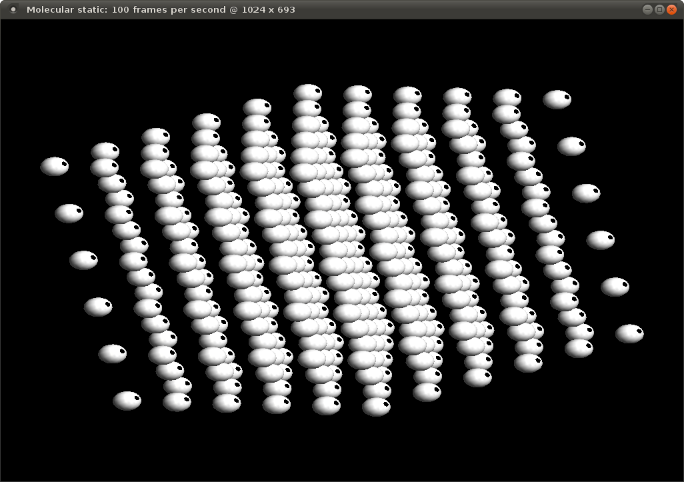


Rys. 5.2. A1 bez defektów – widok z perspektywy – działanie algorytmu z potencjałem Lennard’a-Jones’a

Rysunki 5.1 oraz 5.2 pokazują działanie algorytmu z potencjałem LJ na strukturę A1. Widać, że efekt końcowy to bardzo zwarta struktura. Jest podobna do początkowego ułożenia jednak atomy są tu położone dużo bliżej siebie. Efekt jest wynikiem bardzo gęstego ułożenia atomów w strukturze A1. Podczas każdej iteracji do obliczeń jest branych znacznie więcej sąsiadujących cząstek niż w strukturze A2, co ma odzwierciedlenie w efekcie końcowym.



Rys. 5.3. A2 bez defektów – widok z przodu – działanie algorytmu z potencjałem Lennard’a-Jones’a



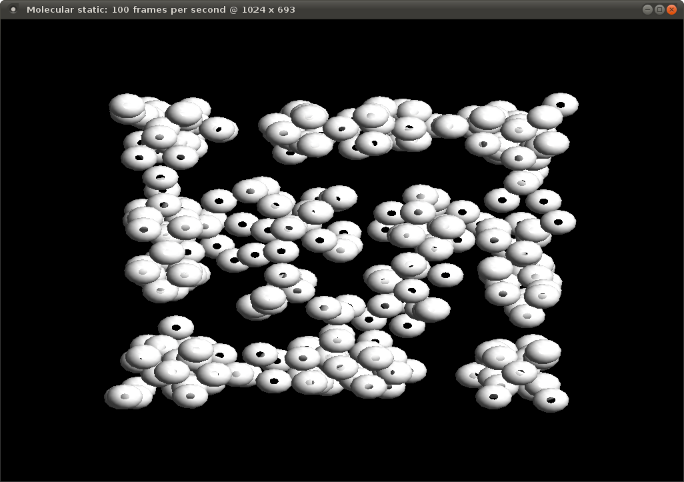
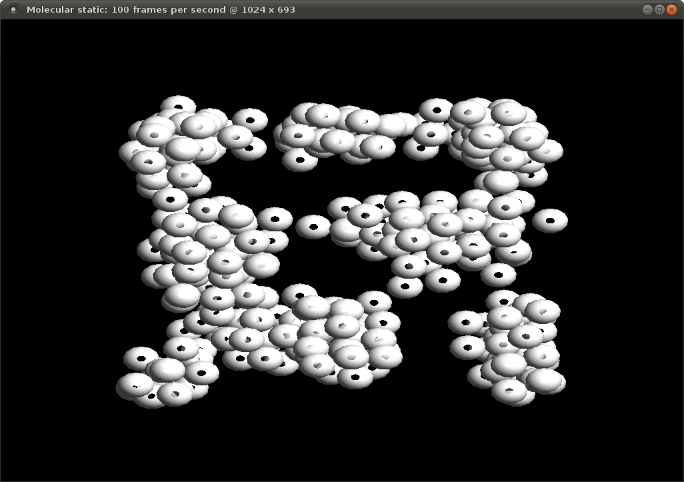
Rys. 5.4. A2 bez defektów – widok z perspektywy – działanie algorytmu z potencjałem Lennard’a-Jones’a

Rysunki 5.3 oraz 5.4 obrazujące działania algorytmu LJ na strukturę A2. Pokazują jak odmienne są efekty działania tego samego solwera na różne struktury gdy porównamy je z rysunkami 5.1 oraz 5.2. W strukturze A2 atomy widocznie grupują się w skupiska. Nie tworzą jednolitej zwartej struktury. Atomy są znacznie dalej od siebie i mają większe odległości.

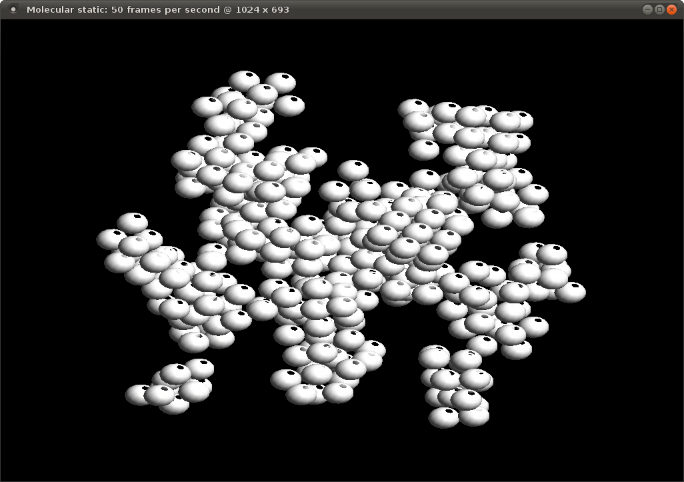
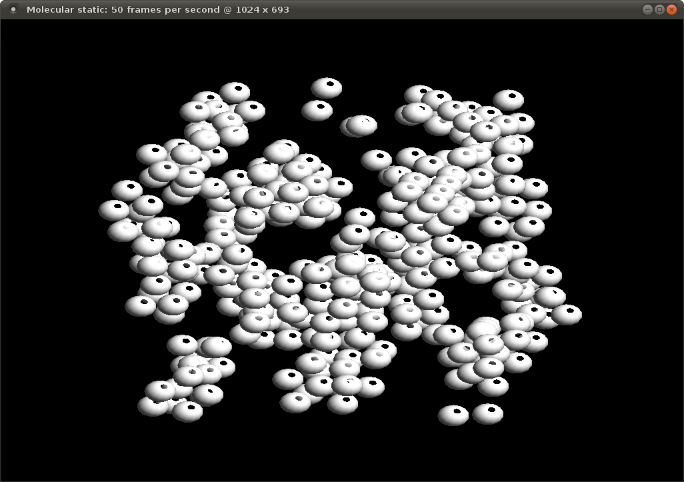
W dwóch kolejnych podrozdziałach wygenerowane zostaną dwa rodzaje defektów. Pokazany zostanie również ich wpływ na powyższe struktury.

### 5.1.2. Defekty struktury A2

Dla struktury A2 zostały wprowadzone punktowe defekty krystalograficzne w postaci wakancji. Ze struktury zostało usuniętych 17 atomów. Sprawiło to, że w pewnych miejscach odległości między cząstkami stały się jeszcze większe.



Rys. 5.5. Struktura A2 – widok z przodu - defekty punktowe: wakancje

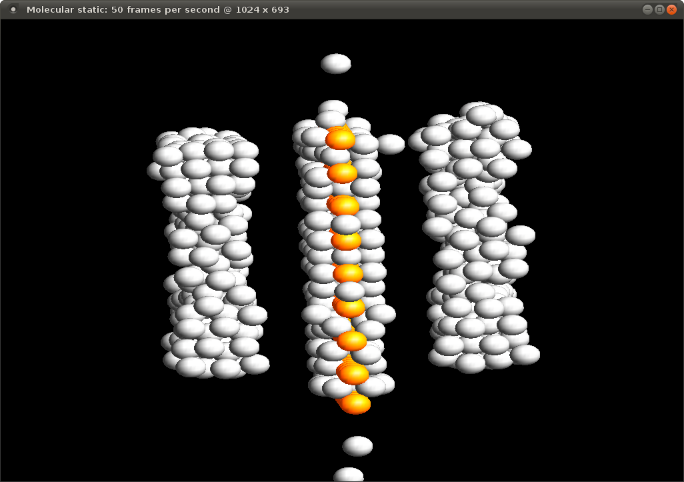


Rys 5.6. Struktura A2 – widok z perspektywy – defekty punktowe: wakancje

Efekt końcowy jest podobny do otrzymanego na rysunku 5.4. Atomy znów utworzyły skupiska. Bardzo wyraźne są miejsca, gdzie zostały wprowadzone defekty. Wygenerowały one większe przestrzenie między skupiskami i wprowadziły nieregularność do struktury, co dobrze obrazuje rysunek 5.5.

### 5.1.3. Defekty struktury A1

Defekt wprowadzonym do struktury A1 to dyslokacja krawędziowa. Dodatkowa płaszczyzna przecina strukturę w centralnej części (rys 5.7).



Rys. 5.7. Struktura A2 – widok z perspektywy, dwa stopnie powiększenia – dyslokacje krawędziowe

W wyniku wprowadzenia defektu struktura uległa znacznej deformacji względem uzyskanej na rysunku 5.2. Pomiędzy ciasno upakowanymi atomami powstały dwie spore przerwy. Prawy zrzut ekranu rysunku 5.7 obrazuje atomy, które zostały wyrzucone ze struktury na znaczną odległość. Można wyciągnąć wnioski, że struktura została zniszczona po zastosowaniu powyższego defektu. Jest to zarazem przykład na to, jak oprogramowanie komputerowe może pomóc w czasie wstępnego modelowania przed wykonaniem testów w rzeczywistości.

Modelowanie zagadnień z zakresu statyki molekularnej, wymaga dużej mocy obliczeniowej. Warto zbadać, czy każdy sprzęt może służyć do wydajnego modelowania. Wyniki testów zaprezentowano w kolejnych rozdziałach.

## 5.2. Porównanie wydajności na różnych platformach testowych

Porównywanie wydajności na procesorach GPU, jak i CPU przedstawię bez wizualizacji, pokazując wyniki obliczeń w postaci czasów obliczeń. Pozwoli to wybrać architekturę, która jest najbardziej wydajna w tym przypadku. Aby wyniki można było między sobą porównywać przygotowany został jeden zestaw danych wejściowych, na których przeprowadzono testy.

|  |  |
| --- | --- |
| **Typ struktury:** | A1 |
| **Ilość atomów w strukturze:** | 29 660 |
| **Ułożenie początkowe:** | Sześcian |
| **Ilość iteracji:** | 5 |
| **Jednostka czasu:** | milisekundy |

Tabela 5.1. Zestaw danych wejściowych do obliczeń

Testy podzielone zostały na kilka etapów.

Etap 1 - GPU

Testy na 1 multiprocesorze przy doborze różnych wartość Global Work Size (GWS) oraz Local Work Size (LWS). Najlepszy wynik będzie wykorzystany w etapie 2.

Etap 2 - GPU

Dla stałego LWS wyznaczonego w etapie 1, zwiększanie GWS. Najlepszy wynik będzie wykorzystany w etapie 3.

Etap 3 - GPU

Dla najlepszych parametrów GWS oraz LWS z etapu 2, zwiększana zostanie liczba atomów w strukturze.

Etap 4 - CPU

Wykonanie testów dla zmieniennej ilości rdzeni CPU. LWS zostanie ustawione na stałą wartość 2 ze względu na technologię Hyper Threading zastosowaną w procesorach. Najlepszy wynik wykorzystany zostanie w etapie 5.

Etap 5 - CPU

Dla najlepszych parametrów z etapu 4, testowany będzie wpływ zmiany ilości atomów.

Testy nie trwają długo, aby utrzymać mniej więcej stałe warunki podczas każdego z testów, dlatego dane wejściowe nie są duże. Większe zestawy wykorzystam przy zagadnieniu skalowalności.

Przed każdym testem przedstawiono parametry techniczne platformy testowej. Platformy są dobrane tak, aby żadne z pozostałych komponentów komputera nie ograniczały wzajemnej wydajności. Pod uwagę wzięto trzy typy maszyn. Pierwsza z nich to sprzęt dedykowany do wykonywania zadań obliczeniowych GPGPU, drugą jest wysokiej klasy komputerem typu PC, trzecia to średniej klasy laptop.

Podczas testów ze zmienna ilością atomów, nie ma stałego przyrostu. Wykres 5.1 przedstawia jak będzie wyglądała ich ilość w kolejnych krokach testowych.

Wykres 5.1. – Zmiana ilości atomów w testowanej strukturze,

Powyższe dane będą wykorzystane w etapie 3 oraz 5 podczas wykonywania każdego testu.

### 5.2.1. Platforma testowa 1 – GPGPU

|  |  |
| --- | --- |
| **Procesor (CPU):** | 2 x Intel Xeon X5650 (12M Cache; 2.66@3,06 GHz)  [2x (6cores +Hyper Threading )] |
| **Karta graficzna (GPU):** | Nvidia Tesla M2090 |
| **Pamięć RAM:** | 16 GB DDR3 |
| **System operacyjny:** | Linux |

Tabela 5.2. Platforma testowa 1 – specyfikacja

Etap 1

Wykres 5.2. – Platforma 1 GPU – GWS = LWS; czas wykonania

Pierwszy test (wykres 5.2) wykazał, że najlepsza konfiguracja GPU jest dla GWS = LWS = 1024. Na wykresie wdać pewne wypłaszczenie dla ustawień parametrów od 256 do 869, gdzie czasy są bardzo zbliżone. Wartość 1024 była maksymalną, na jaką pozwalała architektura GPU.

Etap 2

Wykres 5.3. – Platforma 1 GPU – zmienny GWS; czas wykonania

Wykres 5.4. – Platforma 1 GPU – zmienny GWS; czas wykonania; zbliżenie końcowej części wykresu 5.3.

Testy wykonywane przy zwiększającym się GWS wykazały wyraźne skrócenie czasu działania (wykres 5.3). Z wykresu 5.3 ciężko wywnioskować, jaki jest najlepszy dobór parametrów. Wykres 5.4 jest zbliżeniem końcowej części wykresu 5.3. Widać wyraźnie, że dla GWS = 30720 program osiągnął krótszy czas od poprzednich punktów i od tego miejsca ustabilizował się, dlatego zostanie on uznany za najlepszy wynik. Warto zauważyć, że GWS w tym przypadku ma wartość zbliżoną do ilości atomów w strukturze.

Etap 3

Wykres 5.5. – Platforma 1 GPU – zmienna ilość atomów; czas wykonania

Wykres 5.5 z wynikami testów przy ustawieniach GWS = 30720 oraz LWS = 1024 i zmiennej liczbie atomów ma podobny kształt do wykresu 5.1. Pokazuje to, że czas zmienia się proporcjonalnie do zmiany liczby atomów. Można jedynie zauważyć, że dla granicy ok. 20000 atomów wykres zaczął rosnąć nieco szybciej

Etap 4

Wykres 5.6. – Platforma 1 CPU – zmienna ilość rdzeni; czas wykonania

Testy wykonane na CPU wykazały, że najlepszym ustawieniem jest ilość wątków równa ilości rdzeni procesora (biorąc pod uwagę technologię Hyper Threading). Do dyspozycji były dwa 6-cio rdzeniowe procesory Xeon co dało liczbę wątków równą 24. Zwiększenie ilości wątków sprawiło, że obliczenia były kolejkowane i przełączanie kontekstu wydłużyło czas obliczeń. Można zaobserwować lekkie wypłaszczenie wykresu dla ilości rdzeni od 14 do 18. Nie jest ono jednak tak wyraźne jak dla GPU.

Porównując wykresy 5.5 oraz 5.6 widać zdecydowaną przewagę wydajności GPU (minimalny czas 291.73 ms) nad CPU (minimalny czas 2028 ms) nawet wykorzystując 2 procesory CPU.

Etap 5

Wykres 5.7. – Platforma 1 CPU – zmienna ilość atomów; czas wykonania 1 rdzeń oraz 24 rdzenie

Wykonane testy przy zmiennej ilości atomów wypadły podobnie jak testy dla GPU (wykres 5.5). Wykorzystanie 24 rdzeni dało ok. 10-cio krotne przyspieszenie obliczeń.

### 5.2.2. Platforma testowa 2 – PC

|  |  |
| --- | --- |
| Procesor (CPU): | Intel Core i7 2600K (8M Cache; 3.4@3,8 GHz)  [4 cores +Hyper Threading] |
| Karta graficzna (GPU): | Nvidia GTX 570 |
| Pamięć RAM: | 16 GB DDR3 |
| System operacyjny: | Linux |

Tabela 5.3. Platforma testowa 2 – specyfikacja

Etap 1

Wykres 5.8. – Platforma 2 GPU – GWS = 2 \* LWS ; czas wykonania

W przypadku Platformy 2 nie było możliwości wykonania pierwszego etapu testu według założonego schematu. System zwracał błąd o braku pamięci na karcie przy próbie wykonania programu. Rozmiar danych był zbyt duży dla urządzenia, aby mógł wykonać program dla takich ustawień GWS i LWS. Jest to związane z architekturą karty graficznej, jaką zastosowano.

Etap 1 testu został, więc zmodyfikowany tak, aby GWS = 2 \* LWS. Test nie powiódł się dla LWS = 32 oraz GWS = 64 wykazując brak pamięci. Dalsza cześć przebiegła bez problemów.

Wykres 5.8 zachowuje się inaczej niż w przypadku GPU z Platformy 1 (wykres 5.2), ale również najlepszy wynik osiągnął dla maksymalnej wartości LWS dla danej karty i wyniósł 1024.

Etap 2

Wykres 5.9. – Platforma 2 GPU – zmiana GWS; czas wykonania

Wykres 5.10. – Platforma 2 GPU – zmienny GWS; czas wykonania, zbliżenie końcowej części wykresu 5.9.

Wykres 5.10 jest zbliżony do wykresu 5.4. Widać wypłaszczenie w momencie osiągnięcia takiej samej wartości GWS = 30720. Czas dla GPU z platformy testowej 2 są minimalnie lepsze od tych osiągniętych w przypadku platformy 1.

Etap 3

Wykres 5.11. – Platforma 2 GPU – zmiana ilość atomów; czas wykonania

Testy zwiększania liczby atomów ponownie pokazały granicę ok. 20000 atomów, dla której czasy zaczęły rosnąć szybciej. Wykresy 5.11 oraz 5.5 mają niemal identyczny kształt.

Etap 4

Wykres 5.12. – Platforma 2 CPU – zmiana ilość rdzeni; czas wykonania

W tym przypadku do testów został wykorzystany tylko jeden procesor typu CPU. Wykres 5.12 wykazuje pewną nieregularność. Najlepszy czas znów został osiągnięty dla ilości wątków równej ilości rdzeni procesora, jednak interesujący wynik został osiągnięty dla ilości wątków równej 3. Testy zostały wykonane kilkukrotnie, aby wykluczyć przypadkowość tego wyniku. W dalszej części zbadane zostaną wyniki uzyskane przy ustawieniu ilości wątków na 3.

Etap 5

Wykres 5.13. – Platforma 2 CPU – zmiana ilość atomów; czas wykonania – 1, 3, 8 rdzeni

Jak można było się spodziewać różnica w czasie wykonania na 1 rdzeniu, a na 8 rdzeniach jest wyraźna. Wykres 5.13 zawiera również testy dla 3 rdzeni. Wyniki są dość zaskakujące, ponieważ czasy uzyskane dla takiego ustawienia są nie wiele gorsze od tych otrzymanych przy wykorzystaniu 8 rdzeni. Podczas korzystania z tego procesora CPU, można poważnie rozważyć wykonywanie obliczeń używając algorytmu LJ z wykorzystaniem 3 rdzeni kosztem lekkiego wydłużenia czasu. Pozostałe zasoby komputera mogą być w tym czasie wykorzystane do zupełnie innych celów.

### 5.2.3. Platforma testowa 3 – laptop

|  |  |
| --- | --- |
| Procesor (CPU): | Intel Core i3 M350 (3M Cache; 2.26 GHz)  [2 cores + Hyper Threading] |
| Karta graficzna (GPU): | Radeon HD5650 M |
| Pamięć RAM: | 4 GB DDR2 |
| System operacyjny: | Linux |

Tabela 5.4. Platforma testowa 3 – specyfikacja

Etap 1

Wykres 5.14. – Platforma 3 GPU – GWS = LWS; czas wykonania

Platforma testowa 3 jest z założenia najsłabszą z całego zestawienia. Można to zauważyć już przy pierwszym teście, gdzie możliwości ustawienia LWS są niewielkie i wynoszą maksymalnie 256. Warto zauważyć, że udało się uruchomić program na tej platformie dla 1 multiprocesora, co było niemożliwe dla znacznie lepszego GTX’a 570.

Ponownie maksymalna wartość LWS okazała się najlepsza.

Etap 2

Wykres 5.15. Platforma 3 GPU – zmiana GWS; czas wykonania

Testy zwiększania GWS (wykres 5.15) wypadły bardzo źle. Najlepszy czas został osiągnięty już przy GWS = 512. Dalsze testy zwracały coraz gorsze wyniki.

Etap 3

Wykres 5.16. Platforma 3 GPU – zmiana ilości atomów; czas wykonania

Wykres otrzymany przy zwiększaniu ilości atomów wygląda dobrze, jednak zwracając uwagę na wartości, jakie zostały otrzymane, nie jest już tak dobrze. O ile czasy dla bardzo małej ilości atomów są porównywalne dla wszystkich platform to dla większych struktur platforma 3 i procesor GPU osiąga gorsze wyniki niż CPU dla platformy 1.

Etapy 4 oraz 5 nie mogły zostać wykonane na tej platformie testowej, ponieważ procesor Intel Core i3 nie obsługuje technologii OpenCL.

Kolejnym etapem będzie zbadanie skalowalności najlepszej z powyższych platform. Obliczeniami będą przeprowadzane na większych zestawach danych i sprawdzając przy tym wydajność danej platformy. Oprócz zmiany wielkości problemu, użyty zostanie również drugi solwer wykorzystujący algorytm Sutton’a – Chen’a, ponieważ jest on dużo bardziej wymagający dla sprzętu.

## 5.3. Skalowalność rozwiązania

Zagadnienie skalowalności przedstawione zostanie przy użyciu platformy testowej 1 (GPGPU) z wykorzystaniem procesora GPU. W danych wejściowych zmienna będzie ilość atomów. Początkowa struktura to A1 (14 atomów), a z każdym kolejnym testem ilość atomów będzie zwiększana, aż współczynnik powielenia osiągnie 20 (246 520 atomów). Ustawienia GPU będą w każdym teście identyczne. Jako konfiguracja testowa przyjęta została najlepsza z wcześniej przeprowadzonych testów, dla której sprzęt pracuje najwydajniej. Konfiguracja nie będzie identyczna dla każdego testu, ponieważ w większości przypadków znaczna ilość wątków byłaby bezczynna, niewykorzystana. Aktualna wielkość GWS będzie najmniejszą wielokrotnością LWS taka, aby przekroczyła aktualnie testowaną ilość atomów. Poniższa tabelka przedstawia konfigurację dla kolejnych testów.

|  |  |  |  |
| --- | --- | --- | --- |
| **współczynnik powielenia** | **ilość atomów** | **GWS** | **LWS** |
| 1 | 14 | 1024 | 1024 |
| 2 | 172 | 1024 | 1024 |
| 3 | 666 | 1024 | 1024 |
| 4 | 1688 | 2048 | 1024 |
| 5 | 3430 | 4096 | 1024 |
| 6 | 6084 | 6144 | 1024 |
| 7 | 9842 | 10240 | 1024 |
| 8 | 14896 | 15360 | 1024 |
| 9 | 21438 | 21504 | 1024 |
| 10 | 29660 | 29696 | 1024 |
| 11 | 39754 | 39936 | 1024 |
| 12 | 51912 | 52224 | 1024 |
| 13 | 66326 | 67584 | 1024 |
| 14 | 83188 | 83968 | 1024 |
| 15 | 102690 | 103424 | 1024 |
| 16 | 125024 | 125952 | 1024 |
| 17 | 150382 | 150528 | 1024 |
| 18 | 178956 | 179200 | 1024 |
| 19 | 210938 | 210944 | 1024 |
| 20 | 246520 | 246784 | 1024 |

Tabela 5.5. Konfiguracja GPU do testu skalowalności

Po przeanalizowaniu wcześniejszych wyników powyższy sposób doboru konfiguracji wydaje się być najlepszy. Wykorzystuje minimalną ilość zasobów przy najlepszej wydajności.

Wykres 5.17. Platforma 1 GPU – porównanie algorytmów SC i LJ

|  |  |  |  |
| --- | --- | --- | --- |
| **współczynnik powielenia** | **ilość atomów** | **Lennard - Jones** | **Sutton - Chen** |
| 1 | 14 | 0.26 | 1.27 |
| 2 | 172 | 2.19 | 43.47 |
| 3 | 666 | 7.41 | 257.449 |
| 4 | 1688 | 17.09 | 1007.05 |
| 5 | 3430 | 25.13 | 2112.28 |
| 6 | 6084 | 37.18 | 3066.61 |
| 7 | 9842 | 54.6 | 5620.85 |
| 8 | 14896 | 79.26 | 8306.01 |
| 9 | 21438 | 207.37 | 19733.1 |
| 10 | 29660 | 291.73 | 31507.5 |
| 11 | 39754 | 598.138 | 55125.6 |
| 12 | 51912 | 1013.1 | 89539.2 |
| 13 | 66326 | 1591.16 | 138273 |
| 14 | 83188 | 2548.99 | 223999 |
| 15 | 102690 | 3914.81 | 344313 |
| 16 | 125024 | 5579.24 | 480075 |
| 17 | 150382 | 8790.11 | 693112 |
| 18 | 178956 | 11644.8 | 969354 |
| 19 | 210938 | 1.64E+004 | 1.38E+006 |
| 20 | 246520 | 2.28E+004 | 1.88E+006 |

Tabela 5.6. Porównanie czasów algorytmów SC i LJ

Testy przeprowadzane z wykorzystaniem algorytmu SC były dużo bardziej wymagające dla sprzętu. Czas w każdym kolejnym teście wzrastał znacznie szybciej. Dobrze obrazuje to wykres 5.17, gdzie wyniki dla algorytmu LJ są reprezentowane przez niemal prostą poziomą linię. Kształt wykresu reprezentującego algorytm SC jest jednak zbliżony do tego reprezentującego przyrost ilości atomów w strukturze. Oznacza to, że karta graficzna dobrze radzi sobie z coraz większymi zestawami danych. Wykres nie posiada wyraźnych skoków czasowych, dla których karta graficzna gorzej radziłaby sobie z dużą ilością danych.

Trzeba zauważyć, że czas dla współczynnika powielenia 20 wynosi już ponad 30 minut (tabela 5.6) a dla struktury z wcześniejszych testów zaledwie 0.4 minuty. Sprzęt pracujący przez dłuższy czas zużywa jednocześnie więcej energii. Poniżej przedstawione zostanie zużycie energii przez kartę graficzną Tesla M2090.

Wykres 5.18. Wykres zużycia prądu przez Tesla M2090

Zużycie prądu przez GPU w czasie spoczynku, tzn. bez obciążania jej obliczeniami wynosi ok 33W. W czasie uruchomienia obliczeń nawet przy najmniejszym rozmiarze problemu dla 1 multiprocesora wzrasta od razu do ok 94W. Włączanie kolejnych multiprocesorów stopniowo zwiększa zużycie.

Jak widać zużycie prądu w czasie powyższych obliczeń wzrasta niemal dwukrotnie biorąc pod uwagę najmniejszą i największą strukturę (wykres 5.18). Warto zauważyć, że pobór mocy ustabilizował się ok. współczynnika powielenia wynoszącego 15. Można wnioskować, że od tego momentu zużycie prądu będzie już na porównywalnym poziomie. Biorąc pod uwagę, że maksymalne zużycie prądu przez kartę wynosi 225W jest to bardzo dobry wynik pod względem skalowalności urządzenia. Prowadząc obliczenia na dużo większych zestawach danych zmiana zużycia energii nie będzie już tak bardzo odczuwalna. Zużycie zależne jest od wielkości GWS ustawionej w każdym teście. Przy przeprowadzeniu testu dla współczynnika powielenia 10 wykorzystując GWS dla współczynnika 5 pobór mocy spadnie do ok 140W. Można więc regulować moc przez ustalanie liczby multiprocesorów, jakie będą wykorzystywane do obliczeń. O ile przy jednej karcie graficznej nie będzie to miało aż takiego znaczenia, to wykorzystując cały klaster obliczeniowy złożony z wielu procesorów GPU koszty poniesione za zużytą energię wzrosną znacząco.

# 6. Podsumowanie

Każda platforma testowa zachowywała się nieco inaczej. Miała swoje mocne i słabe strony. Podstawowym wnioskiem jest to, że laptopy średniej klasy lub porównywalny sprzęt typu PC nie nadają się do prowadzenia obliczeń. Są zupełnie niewydajne przy rozwiązywaniu zadań z wykorzystaniem dużej ilości danych. Korzystając ze sprzętu wysokiej klasy typu PC można osiągać zadowalające wyniki. Analizując wykresy 5.5 oraz 5.11 da się zauważyć, że karta graficzna GTX 570 osiąga lepsze wyniki niż Tesla M2090. Nie jest to jednak decydujący argument podczas wyboru sprzętu. GTX 570 ma dużą wadę w postaci elastyczności. W przypadku testów na 1 multiprocesorze karta wypada gorzej nawet od mobilnej karty graficznej będącej częścią platformy 3, nie uruchamiając się w ogóle. Jeśli elastyczność jest istotna w danym projekcie, lepiej wybrać GPU Tesli kosztem niewiele gorszych wyników czasowych. Patrząc z punktu widzenia procesorów CPU platforma wykorzystująca 2 takie procesory jest o wiele wydajniejsza. Konfiguracja z podwójnym CPU nie jest jednak powszechnie stosowana w komputerach typu PC. Takie konfiguracje sprzętowe można spotkać często w rozwiązaniach serwerowych.

Jeśli decydującym czynnikiem jest skalowalność, elastyczność i wydajność w pracy to najlepszą platformą będzie komputer wyposażony w sprzęt dedykowany do prowadzenia obliczeń. Biorąc pod uwagę, że posiada on z reguły więcej niż 1 kartę graficzna (4 w przypadku testowanej platformy), można spróbować wykorzystać jednocześnie każdy procesor GPU próbując osiągnąć jeszcze lepsze rezultaty lub uruchamiając wiele niezależnych obliczeń osobno na każdym multiprocesorze.

Istotną kwestią jest również zużycie energii prezentowane w rozdziale 5.3. Wykorzystując pełną moc sprzętu, zużycie energii przez GPU jest wyraźnie większe. Jeśli równolegle będzie pracowało kilka urządzeń tego typu zużycie to znacznie wzrośnie. Trzeba też wziąć pod uwagę zużycie energii przez CPU oraz inne komponenty platformy obliczeniowej. Przy rozwiązywaniu prostych zadań nie powinno stanowić to problemu, jednak prowadząc obliczenia trwające dni lub tygodnie sprawa zużycia energii może być odczuwalna dużo bardziej.

# 7. Bibliografia

[1] Brodtkorb, A.R., Dyken, C., Hagen, T.R., Hjelmervik, J.M., Storaasli, O.O., 2010, State-of-the-art in heterogeneous computing, *Scientific Programming*, 18, 1-33.z

[2]http://www.asminternational.org/portal/site/www/SubjectGuideItem/?vgnextoid=ad7cdc8cc359d210VgnVCM100000621e010aRCRD (13.12.2012)

[3] <http://zasoby1.open.agh.edu.pl/dydaktyka/fizyka/c_fizyka_metali/17.htm> (13.12.2012)

[4] Richard J. D. Tilley, *Crystals and Crystal Structures* 2008, Wybrane zagadnienia

[5] Dennis Rapaport, *The Art. of Molecular Dynamics Simulation* 2004, Wybrane fragmenty

[6]<http://chemwiki.ucdavis.edu/Physical_Chemistry/Quantum_Mechanics/Atomic_Theory/Intermolecular_Forces/Lennard-Jones_Potential> (13.12.2012)

[7]<http://www.uic.edu/labs/trl/1.OnlineMaterials/nano.publications/03.Nanostructures.InterMoleForce.pdf> (15.12.2012)

[8] http://www.mbnexplorer.com/users-guide/4-energy-and-force-calculation/41-pairwise-potentials/418-quasi-sutton-chen-potential (15.12.2012)

[9] http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf (15.12.2012)

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