T. C.

GAZİANTEP ÜNİVERSİTESİ

SAĞLIK BİLİMLERİ ENSTİTÜSÜ

FARKLI KARMAŞIKLIK SEVİYELERİ İÇEREN PROTEİN MODELLERİ İÇİN PARALEL PROGRAMLAR VE ALGORİTMALAR GELİŞTİRİLMESİ

Gökhan SELAMET

YÜKSEK LİSANS

TIBBİ BİYOLOJİ

DANIŞMAN

Doç. Dr. Hüseyin KAYA

Gaziantep

2015

T.C.
GAZİANTEP ÜNİVERSİTESİ
SAĞLIK BİLİMLERİ ENSTİTÜSÜ
TIBBİ BİYOLOJİ ANABİLİM DALI

FARKLI KARMAŞIKLIK SEVİYELERİ İÇEREN PROTEİN MODELLERİ İÇİN PARALEL PROGRAMLAR VE ALGORİTMALAR GELİŞTİRİLMESİ

GÖKHAN SELAMET

Tez Savunma Tarihi: 08 Ocak 2015
Sağlık Bilimleri Enstitüsü Onayı

Prof. Dr. Mehmet TARAKÇIOĞLU
Sağlık Bilimleri Enstitüsü Müdürü

Bu tez çalışmasının bir "Yüksek Lisans" derecesi için uygun ve yeterli bir çalışma olduğunu onaylıyorum.

Prof. Dr. Ahmet ARSLAN
Tıbbi Biyoloji Anabilim Dalı Başkanı

Bu tez tarafımca okunmuş, kapsamı ve niteliği açısından bir "Yüksek Lisans" tezi olarak kabul edilmiştir.

Doç. Dr. Hüseyin KAYA
Tez Danışmanı

Bu tez tarafımca okunmuş, kapsamı ve niteliği açısından bir "Yüksek Lisans" tezi olarak kabul edilmiştir.

Tez Jürisi	**İmzası**
Prof. Dr. Ahmet ARSLAN	
Doç. Dr. Hüseyin KAYA	
Yrd. Doç. Dr. Tuğba Taşkın TOK	

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığı beyan ederim.

15-01-2015

Gökhan SELAMET

ÖNSÖZ

Bu tez çalışması “TUBİTAK 2211 Yurtiçi Lisansüstü Burs Programı“ tarafından desteklenmiştir. Bu araştırmada yer alan simülasyonların büyük kısmı, TÜBİTAK ULAKBİM, Yüksek Başarım ve Grid Hesaplama Merkezi (TRUBA kaynaklarında) kullanılarak elde edilmiştir. Bu çalışmada bana her türlü desteği sağlayan danışman hocam Doç. Dr. Hüseyin KAYA’ya ve dostum Mustafa TÜLÜ’ye teşekkürü bir borç bilirim.

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

SEMBOLLER LİSTESİ

D	Açık durum
Å	Angstrom
k_B	Boltzmann Sabiti
Φ	Bükülme Açısı
θ	Eğilme Açısı
E	Enerji
H	Entalpi
ε	Epsilon
V	Hız
r	İki Atom Arası Mesafe
a	İvme
N	Katlı durum
F	Kuvvet
m	Kütle
V	Potansiyel Enerji
T	Sıcaklık

KISALTMALAR LİSTESİ

CUDA	Birleşmiş Hesaplamalı Cihaz Mimarisi
GDDR	Çift Kanallı Grafik Hafıza Erişimi
MIC	Çok Çekirdekli Mimari
EDP	Etkileşim Düzen Parametresi
GNU	Gnu Unix Değildir
GPU	Grafik İşlemci Ünitesi
pH	Hidrojenin Gücü (Power of Hydrogen)
PC	Kişisel Bilgisayar
LJ	Lennard Jones
MB	Megabayt
CPU	Merkezi İşlemci Ünitesi
MD	Moleküler Dinamik
NMR	Nükleer Manyetik Rezonans
NVCC	Nvidia Cuda C Derleyicisi
B	Ortalama Kontak Bağlantısı
PDB	Protein Veri Bankası
PDB	Protein Veri Bankası
SCOP	Proteinlerin Yapısal Olarak Sınıflandırıldığı Veri Tabanı
T.C.	Türkiye Cumhuriyeti
NASA	Ulusal Havacılık ve Uzay Dairesi
IBM	Uluslararası iş makineleri
YBH	Yüksek Başarımlı Hesaplama

TABLOLAR LİSTESİ

ÖZET

FARKLI KARMAŞIKLIK SEVİYELERİ İÇEREN PROTEİN MODELLERİ İÇİN PARALLEL PROGRAMLAR VE ALGORİTMALAR GELİŞTİRİLMESİ

Gökhan SELAMET

Tıbbi Biyoloji

Doç. Dr. Hüseyin KAYA

Ağustos 2014, **55 Sayfa**

Günümüzün en sıcak araştırma konularından bir tanesi proteinlerin fiziksel davranışlarının anlaşılmasıdır. Bu alanda yapılan modelleme çalışmalarında iyi sonuçlar elde edilse de, hala proteinlerin katlanma mekanizmasını tam olarak açıklayan bir model geliştirilememiştir. Bunun temel sebeplerinden bir tanesi kullanılan kuvvet alanlarının proteini modellemek için yeterli olmaması ve hesaplama kaynaklarının kısıtlılığıdır. Günümüz bilgisayarlarını tam performans ile kullanmak, geleneksel seri programlama teknikleri yerine ancak paralel programlama teknikleri ile sağlanmaktadır. Bu sebeple, yeni paralel olarak çalışan protein modellerinin geliştirilmesi kritik öneme sahiptir. Bu tez çalışmasında, basitleştirilmiş zincir temsili içeren ve kristal yapı bilgisine dayalı protein modeli geliştirilmiştir. Geliştirilen protein modeli için moleküler dinamik simülasyon yöntemi ve CUDA programlama dili kullanılarak paralel çalışan programlar geliştirilmiştir. Bu programlarla 15 farklı proteinin termodinamik ve kinetik davranışları incelenmiştir. Sonuçlarımız göstermektedir ki; amino asit başına düşen ortalama kontak sayısı veya ortalama bağlantı miktarı protein katlanma termodinamiğini ve kinetiğini belirleyen en temel parametredir.

Anahtar Kelimeler: Protein Katlanması, Paralel Programlama, Moleküler Dinamik Simülasyon

ABSTRACT

DEVELOPING PARALLEL PROGRAMS and ALGORITHMS for PROTEIN MODELS CONTAINING DIFFERENT TYPES of COMPLEXITY

Gökhan SELAMET

Master of Science, Medical Biology

Hüseyin KAYA

January 2015, **55 Pages**

Understanding the physical behaviors of proteins, at present, is one of the most active research topics. Although modeling efforts in this field has provided some encouraging results, a computational model capable of explaining the folding mechanism of proteins was not developed yet. One of the main reasons behind this failure is that the force fields are not good enough for modeling proteins, and limitation of computational power. To get full performance from today's computers can be achieved by using parallel programming algorithms instead of traditional serial programming. Therefore, it is critically important to develop new protein models, which can run with parallel algorithms. In this dissertation study, a coarse-grained protein model with explicit chain representation, which also relies on crystal structure information has been formed. Then, by taking into account the molecular dynamic simulation technique and by using CUDA programming language, new parallel codes were developed for this protein model. Thermodynamic and kinetic behaviors of 15 different proteins were investigated with these programs. Our results indicate that the main parameter, which determines protein folding thermodynamics and kinetics, is the average contact amount per amino asit, or, average connectivity.

Key words: Parallel Programming, Protein Folding, Molecular Dynamic Simulation.

1. GİRİŞ ve AMAÇ

Bilgisayar teknolojilerindeki hızlı gelişmeler sayesinde; bilgisayarların hesaplama ve veri depolama kapasitelerinde inanılmaz artışlar sağlanmıştır. Bu gelişmelerin sonucu olarak bilgisayarlar olabildiğince kompleks bilimsel problemlerin çözümünde ve endüstriyel uygulamalarda kullanılmaya başlanmıştır. Bunun en tipik örneklerinden birisi, sağlık bilimleri ve temel bilimlerinin üzerinde yoğunlaştığı; biyolojik sistemlerin davranışlarının moleküler seviyede anlaşılması çalışmalarıdır. Biyomoleküler sistemlerin fizyolojik şartların (mesela pH, sıcaklık, çözücü vd.) değişimine gösterdiği hassaslık (termodinamik ve kinetik davranış değişkenliği) ve içerdikleri fiziko-kimyasal etkileşim, yapı ve fonksiyon çeşitliliği; bu sistemlerin ne kadar kompleks ve spesifik olduklarını anlamak için yeterlidir. Mesela, her biyomolekülün kendine has kararlı katlanmış yapısı ve yine sadece kendilerine has biyolojik fonksiyonları olmaktadırlar. Diğer taraftan, biyomoleküllerin konformasyonel değişimleri çok hızlı gerçekleştiği ve de halihazırdaki deneysel tekniklerin (NMR, X-ışını kristalografi vd.) bu konformasyonel değişimleri takip edebilmemize olanak sağlamadığı için bilgisayara dayalı modelleme çalışmaları kritik öneme sahiptir.

Örneğin, 90'lı yılların başında basitleştirilmiş model yaklaşımları ile anlaşılabileceği düşünülen protein katlanması mekanizması, maalesef hala tam olarak anlaşılmış durumda değildir. Konformasyonlar arası geçişleri belirleyen amino asitler arası ve/veya atomlar arası etkileşmeler; farklı araştırma gruplarınca gerek fiziko-kimyasal özellikleri gerekse farklı komplekslik seviyeleri içerecek şekilde tasvir edilmeye çalışılmıştır. Fakat geliştirilen bu etkileşim kuvvet alanları ile yapılan modelleme çalışmalarından elde edilen sonuçlar, çoğunlukla deneysel gözlemlerle örtüşmemektedir.

Geliştirilen süper bilgisayarlar 70'li yıllara kadar tek işlemcili üretilmiş olsa da, maliyet performans oranının artması, ve çekirdek başına hızlanmanın fiziksel limitlere ulaşmasıyla, birden fazla işlemcinin kullanıldığı küme (cluster) süper bilgisayarların kullanılmasına başlanmıştır. Bu süreçte mevcut seri programlama paradigmaları, üretilen yeni küme sistemlerin efektif kullanılmasını sağlayamamaktadır. Ayrıca, şu anda kurulan süper bilgisayar merkezlerinde ki hesaplama gücü, sadece işlemci tabanlı

değil, GPU ve MIC gibi yardımcı kartlar ile artırılmaktadır. Bu hibrit sistemlerin kullanılması için de yeni programlara ihtiyaç duyulmaktadır. Yeni küme yapılarının efektif kullanılması ancak hibrit paralel programlar ve algoritmalar geliştirilmesi sayesinde mümkün olmaktadır.

Diğer taraftan, bilgisayar teknolojileri hızla ilerlese ve hesaplama kapasiteleri inanılmaz şekilde artsa da birçok bilimsel problemin çözümüne ulaşmakta hala sınırlı kalmaktadır. Bu nedenle, ilgilenilen sistem veya problem ile ilgili bilgisayara dayalı model geliştirme çabalarında, gerçeğe en yakın sonuçlar elde edilebilecek, en basit model yaklaşımları tercih edilmelidir.

Bu tez çalışmasında, bir sistemdeki her parçacığın hareket denkleminin her zaman adımında ayrı ayrı çözülerek (integrasyonu alınarak), parçacıklar arası etkileşmelerin, parçacıklara etkiyen kuvvetlerin, parçacıkların yer değiştirmelerinin ve dolayısıyla hızlarının hesaplandığı moleküler dinamik simülasyon yöntemi kullanılarak; protein katlanması dinamiği için geliştirdiğimiz ve amino asitlerin tek bir atom (C_α) ile temsil edildiği basitleştirilmiş protein modelimiz paralelleştirecektir. Bu çalışma daha kompleks protein modelleri için zemin hazırlayacaktır. Basitleştirilmiş zincir temsili içeren protein modeli için CUDA paralel programlama dili ve kütüphanesi kullanılarak, daha hızlı moleküler dinamik simülasyon programı ve algoritması geliştirilecektir. Geliştirilen bu yeni algoritma gerçek proteinler kullanılarak, amino asit başına düşen ortalama etkileşim sayısının (connectivity) protein katlanması termodinamiği ve kinetiğine etkileri incelenecektir.

2. GENEL BİLGİLER

Basitleştirilmiş zincir temsili içeren protein modeli için paralel program ve algoritma geliştirmeyi hedefleyen ve disiplinler arası bir bilgi birikimi gerektiren bu tez çalışmasında; bilinmesi gereken paralel programlama ve proteinlerin yapısı, katlanması ve diğer dinamik davranışları hakkında en temel bilgiler verilecektir.

2.1. Paralel Programlama

Geleneksel seri programlamada, işlemci çekirdeğine verilen komutlar veriliş sırasıyla yapılır. Bunun sebebi işlemci çekirdeklerinin aynı anda birden fazla işi yapabilme kabiliyetlerinin olmamasıdır. Paralel programlama, birden fazla işlemci çekirdeğinin tek bir problemi çözmek için eş zamanlı ve beraber kullanılmasıyla sonuca ulaşılan programlama türüdür. Paralel programlamayı sadece yazılım olarak düşünmek yanlıştır. Donanımlarında paralel programlama için uygun olması ve donanımın efektif kullanılması, paralel programlama için çok önemlidir. Bazı paralel programlar, uygun donanımlar için geliştirilmiş özel programlama dilleri ile sağlanmaktadır.

2.1.1. Donanım

Bilimsel problemlerin kompleksliği her geçen gün artmaktadır. Bu problemleri çözmek için gereken kaynaklarda her geçen gün artmalıdır. Hesaplamalı bilimlerde en temel kaynak işlemcinin hesaplama gücüdür. Her işlemcinin hesaplama kapasitesi, mimarisi ve fiziksel donanım özelliklerine göre değişmekle birlikte sabittir. Daha hızlı hesap yapılabilmesi için yeni işlemcilerin üretilmesi gerekmektedir. Intel'in kurucularından Gordon E. Moore'un, Moore kanuna göre bir bilgisayar işlemcisinde ki transistor sayısı her iki yılda, iki katına çıkmaktadır. [1]

Günümüzde kullanılan x86 mimarisine sahip modern işlemciler tek çekirdekte 12GFlop işlem yapabilirler. Geliştirilen en hızlı tek çekirdekli süper bilgisayar ise 100GFlop işlem kapasitesine sahiptir. [2] Maalesef bu hızlar, güncel bilimsel problemlerin çözümü için yeterli değildir. Karşılaştırmak gerekirse günümüz yüksek başarımlı hesaplama merkezlerinde 54000TFlop hızına ulaşılmıştır.[top500] Bu tek çekirdek ile ulaşılan maksimum hızın 540 bin katıdır. Bunun için araştırmacılar ve üreticiler çok çekirdekli

işlemcilerin kullanıldığı sistemlere yönelmiştir. 1968 yılında ilk çok çekirdekli işlemci IBM tarafından üretilmiştir [3]. 1994 yılında NASA tarafından eski kişisel bilgisayarlardan (PC) oluşan bilgisayar kümesi ile hesaplama oluşturulmuştur. [4]

Günümüzde işlemci üreticileri, artan hesaplama ihtiyacını karşılamak için tek çekirdekli işlemler yerine çok çekirdekli işlemciler üretmeyi tercih etmektedirler. Bunun sebebi, tek çekirdekli işlemcilerde fiziksel limitlere ulaşılmasıdır. Artan saat hızları ile birlikte (clock speed) işlemcilerin ısınma oranı artmış, dolayısı ile enerji verimliği düşmüştür. Geliştirme ve üretim maliyeti beklenenin çok üzerine çıkmıştır. Bu sebeplerden, üreticiler birden fazla çekirdeği bir işlemci üzerinde barındıran, çok çekirdekli işlemciler üretmeye başlamıştır. Şu an kullandığımız kişisel bilgisayarda ve hatta akıllı telefonlarda dahi 4 çekirdekli modeller bulunmaktadır. Bu çok çekirdekli işlemcilerin tam performans ile kullanmak geleneksel seri programlama ile mümkün değildir. Bu işlemcileri tam performans ile kullanmak ancak paralel programlar ile ulaşılabilir..

2.1.1.1. Çok çekirdekli merkezi işlemciler

Merkezi işlemciler (CPU), bir bilgisayar için gerekli ana işlem ünitesidir. Bir bilgisayar için olmazsa olmaz bileşendir. Paralel hesaplamada yaygın olarak x86, itanium ve PowerPc mimarileri kullanılır.

2.1.1.1.A. X86

Kişisel bilgisayarlarımızda dahil olmak üzere bir çok bilgisayarda x86 mimarisi ile üretilmiş işlemciler kullanılır. x86 mimarisi 32 bit olarak tasarlanmıştır. Fakat gelişen teknoloji ile birlikte 32bit işlemciler özellikle ram adreslemedeki yetersizliği sebebi ile yerini 64bit işlemcilere bırakmıştır. Bu noktada, geçmiş yazılımlar ile uyumluluğu kaybetmemek için x86 mimarisinden 64bit uyumlu x86_64 mimarisi üretilmiştir. Bu mimari esasen 48bit fiziksel, 16 bit mantıksal olarak tasarlanmış ve 64bit uyumlu hala gelmiştir..

2014 yılında Intel'in x86 uyumlu tek bir işlemcide 15 çekirdek barındıran bir işlemci üretilmiştir. Bu işlemciden bir ana karta en fazla 8 adet takılabilir. Yani x86 mimarisinde tek bir bilgisayarda 120 çekirdeğe kadar çıkabilen bilgisayarlar artık mevcuttur.

2.1.1.1.B. Itanium

Itanium mimarisi intel tarafından piyasaya sürülen, gerçek 64 bit adresleme kapasitesine sahip bir mimaridir. 2014 yılında tek işlemcide 8 çekirdekli itanium işlemciler piyasada mevcuttur. Bu işlemciler tek ana karta 32 adete kadar takılabilmeye uygundur. Yani itanium mimarisinde tek bir bilgisayarda 256 çekirdeğe kadar çıkabilen bilgisayarlar üretilebilir.

Itanium işlemciler, pahalı olması ve geçmişte geliştirilen x86 uyumlu uygulamalardan gerekli performans alınamaması sebebi ile yüksek başarımlı hesaplama merkezleri tarafından çok tercih edilmemektedirler. Fakat gerçek 64 bit mimariye sahip olması ve tek bir bilgisayarda 256 çekirdeğe kadar desteklemesi sebebi ile özel bir takım uygulamalarda kullanılmaktadır.

2.1.1.2. Yardımcı işlemciler

Yüksek başarımlı hesaplamada, merkezi işlemcilerin yanında, hesap yapma kabiliyetine sahip kartlar kullanılmaya başlamıştır. Bu kartlar her türlü işlemi yapma kabiliyetine sahip değillerdir ve çalışmak için merkezi işlemciye ihtiyaç duyarlar. Özel olarak geliştirilen bu kartların uygun kullanımında merkezi işlemciye nazaran çok iyi performans sergilemektedirler. Günümüzde yaygın olarak kullanılan hızlandırıcılar, GPGPU olarak adlandırılan grafik kartları ve Xeon Phi kartı dır.

Her yıl, en hızlı 500 yüksek başarımlı hesaplama merkezinin listelendiği listede, 75 tane yardımcı işlemci kullanılan sistem vardır. Ayrıca bu listenin birinci sırasında, 54PFlop performansı ile Xeon Phi işlemcili MilkyWay-2 kümesi vardır. [top500]

2.1.1.2.A. Grafik kartları

Standart grafik kartlarındaki donanım, bilgisayarın görüntü çıktısı olan ekran üzerindeki her bir görüntü noktasının (piksel) renk değerlerinin, ve alfa değerinin gerçek zamanlı olarak hesaplaması için gereklidir. Ekrandaki görüntü noktasının fazlalığı ve her saniyede minimum 30 kez görüntünün yenilendiğini düşündüğümüzde, ekran kartlarında ciddi bir hesaplama kapasitesi vardır. Bu hesaplama kapasitenin, diğer hesaplama işlemler için kullanılması fikri ile grafik işlemciler paralel programlama kazanılmıştır.

Günümüzde GPGPU adı altında sadece hesap yapmak için özel olarak tasarlanmış ekran kartları mevcuttur. Aynı zamanda geliştiriciler, bu kartlar içi özel programlama dilleri ve kütüphaneler tasarlamışlardır. Bunlardan en yaygın olarak kullanılan CUDA ve OpenCL programlama dilleridir.

CUDA, Nvidia firması tarafından geliştirilmiş bir programlama dilidir. Sadece Nvidia tarafından geliştirilen grafik kartları ile uyumludur.

OpenCL, öncülüğünü Apple'ın yaptığı Khronos Group tarafından geliştirilen bir programlama dilidir. Grafik kartları başta olmak bir çok işlemci ile uyumludur.

CUDA, OpenCl'e nazaran çok daha az donanım ile çalışabilmesine rağmen, bilimsel hesaplamalar için OpenCL'den daha fazla tercih edilmektedir. Bunun sebepleri; CUDA ile uygulama geliştirmek OpenCl'e nazaran daha kolay olması, daha fazla hazır kütüphanenin bulunması, ve Nvidia firmasının ilgili donanımlarının performanslı olmasından kaynaklanmaktadır. Şu anda Nvidia'nın en gelişmiş grafik kartında, 2880 işlemci çekirdeği vardır. Bu kartın 1.43 TFlops hesaplama hızı vardır.

2.1.1.2.B. Xeon Phi kartı

Xeon Phi kartı Intel tarafından geliştirilen çok işlemcili bir yardımcı işlemcidir. Bu kartın üzerinde yaklaşık 60 çekirdek vardır. Bu karttaki işlemciler, mimari olarak merkezi işlemciye çok benzerdir. Böylelikle CPU için üretilmiş uygulamaların bu kartta sadece küçük değişiklikler ile çalışması amaçlanmıştır. Grafik kartlarındaki mimari ve bellek yapısındaki programlama karmaşıklığı göz önünde bulundurulduğunda, Phi kartları için uygulama geliştirmek daha kolaydır.

2.1.2. Parallel programlama türleri

Paralel programlama bellek tipine göre; paylaşımlı veya paylaşımsız bellek olarak sınıflandırılır.

2.1.2.1. Paylaşımsız bellek modeli

Düğüm (node), tek başına çalışabilen bir bilgisayardır. Paylaşımsız bellek modeli, birden fazla düğümün tek bir problemi çözmek amacı ile kullanıldığı paralel programlama çeşididir. Bu model, tek bir düğümdeki çekirdek veya bellek miktarının,

mevcut problemi çözmek için yeterli olmadığı durumda kullanılır. Küme ve grid veya bulut sistemler bu model ile çalışabilecek yapıda oluşturulur.

Kurulan kümelerde hesap kaynakları düğüm sayısını artırarak sağlanabilir. Bu yüzden paylaşımsız bellek modelinin, genişletilebilirliği diğer süper bilgisayarlara nazaran daha gelişmiştir. Günümüzdeki en hızlı yüksek başarımlı hesaplama merkezinde paylaşımlı bellek kullanan 3 milyon işlemci çekirdeği vardır.

2.1.2.1.A. Haberleşme protokolleri

Paylaşımsız bellek kullanan süper bilgisayarlarda hesaplama kapasitesi ile birlikte önemli bir diğer faktör haberleşme hızlarıdır. Düğümler hesap yaparken, elde edilen sonuçlar farklı düğümler ile haberleşme ihtiyacı duyar. Bu ihtiyacın karşılanması için hızlı haberleşmelere donanımlarına ihtiyaç vardır. Paylaşımsız bellek modelindeki en zayıf halka haberleşme hızıdır. Günümüzde en hızlı protokollerden infiniband'ın maksimum haberleşme hızı 40Gbit'tir. Infiniband'in yanı sıra, yüksek başarımlı hesaplama merkezlerinde özel tasarlanmış donanımlarda kullanılmaktadır.

2.1.2.2. Paylaşımlı bellek modeli

Paylaşımlı bellek modeli, tüm çekirdeklerin aynı bellek alanını kullandığı paralel programlama çeşididir. Genellikle bu modeli kullanan süper bilgisayarlar, tek bir anakart üzerinde toplanmış işlemcilerin, tek bir işletim sistemi tarafından yönetilmesiyle gerçekleşir. Bu süper bilgisayarların üretimi pahalı ve ulaşabilecek maksimum işlemci ve ram kapasitesi sınırlıdır.

2.2. Proteinler

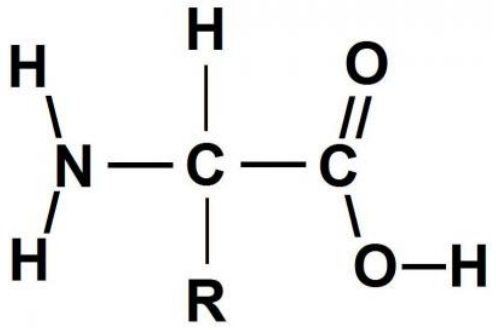
Proteinler, canlılardaki bir çok yapısal ve kimyasal görevi gerçekleştiren organik moleküllerdir. Hücrelerdeki bir çok fonksiyon proteinler tarafından gerçekleştirilmektedir. Enzimlerin, kasların, antikorların, hormonların hepsi birer proteindir.

Proteinlerin yapıtaşı amino asitler olup, ribozom tarafından 20 farklı amino asidin art arda eklenmesi ile oluşturulurlar. Bu işlem sırasında, hangi amino asitlerin hangi sıra ile dizileceği bilgisi mRNA tarafından belirlenir. Proteinlerin sahip olduğu amino asit içeriği ve miktarı o proteinin biyolojik olarak aktif olduğu düşünülen katlanmış yapısı,

fonksiyonu ve kararlılığını belirlemektedir. Diğer taraftan, 20 amino asidin farklı miktar ve dizilişleri ile inanılmaz sayılarda protein dizileri oluşturulabilme ihtimali varken, doğal seçicilik dolayısıyla doğada sentezlenen protein çeşidi düşünülenden çok daha düşüktür.

2.2.1. Amino asitler

Doğa da pek çok farklı amino asit mevcut olmasına karşın, protein sentezinde sadece aşağıda Tablo 2.1 de isimleri verilen amino asitler kullanılmaktadır. Yüksek biyolojik öneme sahip olan bu amino asitler; alfa karbon atomuna (C_α) bağlı fonksiyonel karboksil grubu ($-COOH$), amin grubu ($-NH_2$), ve R ile temsil edilen kenar zincirinin bağlanması oluşan bir organik moleküldür (Şekil.2.1). Vücudumuzdaki hücreler bu amino asitlerin yarısını hücre içinde sentezlerken diğer yarısını vücudumuza aldığımız besinlerden sağlarız.



Sekil 2.1 Amino asitlerin örnek temsili

Amino asitleri birbirinden ayırt eden ve farklı fiziko-kimyasal özellikler kazandıran kısmı R kenar zinciridir. Kenar zincirler içerdikleri atom türü ve çeşidine göre farklı özelliklere sahip olmaktadırlar. Örneğin sadece karbon atomu içeren amino asitler hidrofobik özelliğe sahip olurken, hidrofobiklik seviyeleri sahip oldukları toplam karbon atomu sayısı ile orantılıdır. Diğer taraftan, pozitif yüklü azot (arjinin ve lisin) ve negatif yüklü oksijen içeren (aspartik asit ve glutamik asit) amino asitler yüklü amino asitlerdir. Zıt yüklü amino asitler arasındaki etkileşmeler çok güçlü olup birbirlerine yeterince yaklaştıklarında tuz köprüleri (salt bridges) oluştururlar.

Tablo 2.1 Amino asit çeşidi listesi

Amino Asit Adı	**Üç Harfli Kısaltma**	**Tek Harfli Kısaltma**
Alanin	Ala	A
Arjinin	Arg	R
Asparajin	Asn	N
Aspartik Asit	Asp	D
Sistein	Cys	C
Glutamik Asit	Glu	E
Glutamin	Gln	Q
Glisin	Gly	G
Histidin	His	H
İzolösin	Ile	I
Lösin	Leu	L
Lisin	Lys	K
Metiyonin	Met	M
Fenilalanin	Phe	F
Prolin	Pro	P
Serin	Ser	S
Treonin	Thr	T
Triptofan	Trp	W
Tirozin	Tyr	Y
Valin	Val	V

Diğer taraftan, oluşturdukları kovalent bağlardaki eşit olmayan elektron ortaklaşmasından dolayı elektronegatiflik (kalıcı dipol özelliği) gösteren amino asitler polar amino asitler adlandırılmaktadır. Polar ve yüklü amino asitler genellikle proteinlerin katlanmış yapısında protein yüzeyinde gözlemlenirler. Fakat hidrofobik amino asitler su molekülleri ile hidrojen bağı oluşturmadıklarından dolayı, proteinlerin katlı yapısında iç bölgelerde bulunurlar.

2.2.2. Proteinlerin yapısı

Yukarıda ifade edilen amino asitlerin farklı fiziko-kimyasal özellikleri beraberinde etkileşim çeşitliliği getirmektedir. Proteinler hücre içinde sentezlendikten sonra içinde bulunduğu fizyolojik şartların elverdiği ölçüde ulaşabileceği en düşük enerjili bir yapıya (konformasyona) ulaşmaya çalışır. Genel olarak ulaşılan en kararlı konformasyon en düşük enerjili konformasyon ve katlanmış yapı olarak adlandırılır. X-ışını kristalografi

ve NMR deneylerinden elde edilen proteinlerin katlanmış yapıları incelendiğinde; bazı geometrik yapıların tekrarlandığı gözlemlenmiştir. Buna göre, proteinler yapısal olarak dört yapıya ayrılarak tanımlanmıştır. Bu yapılar sırası ile birincil (primer) yapı, ikincil (sekonder) yapı, üçüncül (tertiary) yapı ve dördüncül (quarternary) yapıdır.

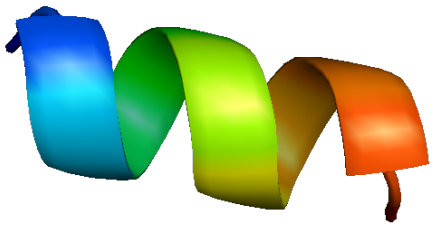
Bir proteinin amino asit dizi (dizilim) bilgisi birincil yapı olarak tanımlanır ve proteinin en basit yapısıdır. Aslında birincil yapı çok bilgi içermiyor gibi görünse de, diğer yapıların belirlenmesinde ve oluşmasında anahtar rol oynamaktadır.

2.2.2.1. İkincil yapı

Zincir üzerinde birbirine yakın olan amino asitlerin oluşturduğu; sarmal, β-kırmalı tabaka ve dönüşler olarak tanımlanan üç boyutlu yapılara ikincil yapı denir. Bu yapılar amino asitler arası yapılan hidrojen bağları sayesinde oluşurlar.

2.2.2.1.A. Sarmallar

Proteinin bir eksen etrafında kıvrılarak (helezonlar) oluşturduğu yapıya sarmal yapı denir. Genellikle 5 ile 14 amino asit uzunluğundadır. En sık gözlemlenen sarmal türü alfa sarmalıdır. Protein zinciri üzerinde komşu amino asitlerin omurga atomlarının aralarında yaptığı hidrojen bağları sarmalların oluşmasını ve kararlı olmalarını sağlar.



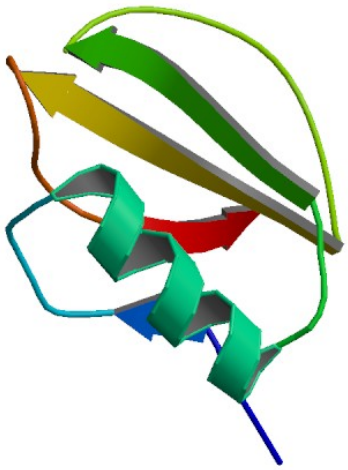
Sekil 2.2 Sarmal yapı gösterimi

2.2.2.1.B. β-Kırmalı tabaka

Protein zincir üzerinde birbirini takip amino asitlerin düz akordiyon (helezon oluşturmayan) şeklinde bir tabaka veya levha oluşturduğu ve yine zincir üzerinde kendilerinden uzakta olan amino asitlerle yaptıkları hidrojen bağları ile kararlı hale gelen yapılardır. Paralel ve paralel olmayan kırmalı tabakalar mevcuttur.

2.2.2.2. Üçüncül yapı

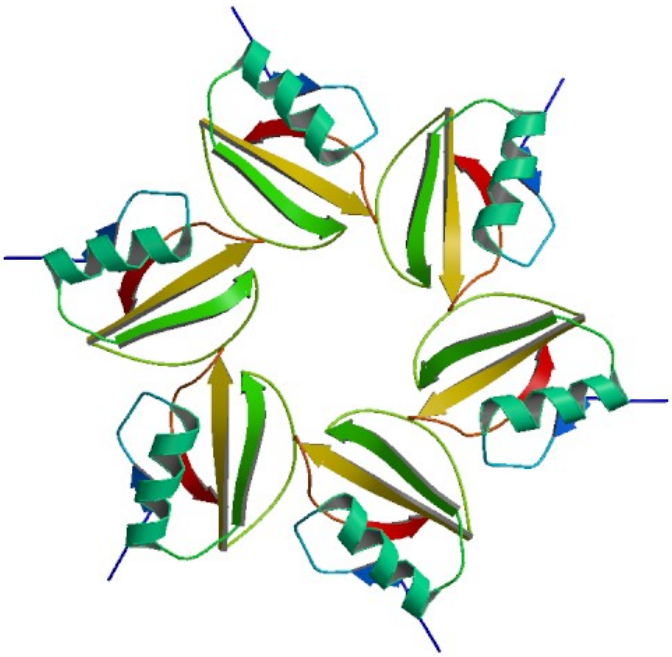
İkincil yapıların bir araya gelerek oluşturdukları geometrik yapıya üçüncül yapılar denir. Bu yapılarda zincirin herhangi bir bölgesinde bulunan bir amino asit, zincir üzerinde kendisinde çok uzakta olan bir amino asit ile etkileşme fırsatı bulur. Dolayısıyla, üçüncül yapıların oluşmasında veya belirlenmesinde bu etkileşmelerin rolü büyüktür.



Sekil 2.3 Üçüncül yapılara örnek, 2CI2

2.2.2.3. Dördüncül yapı

Üçüncül yapısı oluşmuş proteinlerin, başka proteinler ile oluşturdukları daha büyük kompleks yapılardır. Bu yapılar farklı çeşit proteinlerin etkileşmesi ile oluştuğu gibi, aynı proteinin birden fazla üçüncül yapı kopyasının bir araya gelmesi ile de oluşabilmektedirler.



Sekil 2.4 Dördüncül yapılara örnek, 2CI2

2.2.3. Protein aileleri

Proteinler sahip oldukları ikincil yapıların çeşitliliğine göre üç ana ailede incelenirler. Sadece sarmal yapıya sahip olan proteinler "tamamıyla heliks" ailesinde, sadece kırmalı tabaka olan proteinler "tamamıyla beta kırmalı" ailesinde, hem sarmal hem de kırmalı yapı içeren proteinler "karışık yapılı" ailesindedir. Bu aileler sınıflandırılmasında, ve yapı detaylarına göre daha detaylı kategorize edilmesinde yaygın olarak SCOP (Structural Classification of Protein Database) adlı veri bankası kullanılır. SCOP veri bankasında, sınıf katlanma, aile, süper aile gibi hiyerarşik seviyeler ile proteinler kategorize edilmiştir. [5]

2.2.4. PDB veri bankası

"Protein Databank" (PDB), X-ışını kristalografi ve NMR deneylerinden elde edilen proteinlerin üç boyutlu yapılarının arşivlendiği bir veri bankasıdır. Global ölçekte kullanılan en büyük protein veri bankasıdır. Deneysel olarak elde edilen her yeni kristal yapılar bu veri bankasına yüklenir ve kullanıma sunulur. Yüklenen her yapının kendine has dört karakterli bir kodu vardır.

Protein veri bankasında, yüklenen dosyalarının daha kolay işlenmesi için bir dosya tipi oluşturulmuştur. Bu dosya tipinde, ilgili proteinin deneysel sonuçlarından elde edildiği çalışma detaylarına, güncelleme detaylarına, zincir bilgisine, ikincil yapılara, atom koordinatlarına kadar bir çok önemli bilgi bir arada bulunmaktadır. Bu tez çalışmasında proteinlerin 3 boyutlu yapısını almak için kullanacağımız bölümden bir kaç satır aşağıda verilmiştir.

```
ATOM      1  N   SER A   1       9.388  10.581  -6.335  1.00  5.36   N
ATOM      2  CA  SER A   1      10.739  10.012  -6.580  1.00  4.79   C
ATOM      3  C   SER A   1      11.077   8.937  -5.553  1.00  4.03   C
ATOM      4  O   SER A   1      11.455   9.243  -4.422  1.00  4.41   O
```

Bu satırlarda bize 3 boyutlu kristal yapıdaki atomların fiziksel özellikleri hakkında bilgi veriliyor. Detaylı olarak inceleyecek olursak veriliş sırası ile, ilgili satırda bir atom bilgisi olduğunu, atomun sıra numarasını, hangi amino aside ait olduğu, hangi zincir olduğu, amino asit sıra numarasını, atomun x, y, z koordinat değerlerini, işgal (occupancy) değerini ve hangi element olduğu her satırda sırayla veriliyor.

2.2.5. Protein katlanması

Canlıların yaşamsal fonksiyonlarının yerine getirilmesinde rol oynayan proteinlerin, bu biyolojik fonksiyonları hangi konformasyonlarda yerine getirdikleri, biyolojik olarak aktif oldukları bu yapılara ne tür konformasyonel değişimler altında ulaştıkları ve hangi etkileşimlerin ve/veya amino asitlerin belirleyici (dominant) olduğunun anlaşılması kritik önem taşımaktadır.[6] Proteinlerin hücre içinde sentezlendikten sonra biyolojik olarak aktif ve kararlı oldukları kabul edilen, ve her proteinin kendine has 3 boyutlu bir geometriye sahip olduğu katlanmış yapılarına ulaşma süreci protein katlanması olarak adlandırılmaktadır. Sıcaklık, pH ve çözücü miktarları gibi fizyolojik şartların değiştirildiği durumlarda proteinler açılıp tekrar katlanabilmektedir.

Miyoglobin proteinin üç boyutlu kristal yapısının keşfinden bu yana, protein katlanması araştırmacılar tarafından üzerinde yoğun çalışılan bir alan haline gelmiştir. Protein katlanması çok hızlı süreçlerde (ortalama olarak milisaniye mertebesinde) gerçekleştiği için mevcut deneysel yaklaşımlar, proteinlerin konformasyonel değişimlerini takip etmemize imkan sağlamamaktadır. Katlı yapıya ulaşmaya çalışılırken ziyaret edilen ara durum konformasyonları kararsız olduğu ve bu nedenle o konformasyonlarda bir relaksasyon gerçekleşmediği için ziyaret edilen ara durum konformasyonlarında atom

koordinatları elde edilememektedir. Sadece daha kararlı oldukları için katlanmış yapıdaki atom koordinatları ve hangi amino asitlerin birbirleri ile etkileştikleri hakkında bilgi elde edilebilmektedir.

Son yıllarda insan dahil bir çok canlının gen haritaları (genomları) tamamlanmış ve bu canlıların sahip oldukları milyonlarca proteinin dizi bilgileri elde edilmiştir. Fakat şu ana kadar sadece 100.000 civarında proteinin katlanmış yapısı elde edilebilmiştir. Dolayısıyla protein katlanma yolaklarını ve doğru katlı yapıları hızlı ve efektif şekilde belirleyebilecek bilgisayara dayalı yaklaşımlara ihtiyaç duyulmaktadır.

Proteinler içerdiği amino asit sayısı (zincir uzunluğu), amino asit içeriğine bağlı olarak tamamıyla farklı termodinamik ve kinetik davranışlar sergilemektedirler. Mesela, amino asit sayısı 120'den fazla olan proteinler çok durumlu, 120'den küçük olanlar ise iki durum termodinamik davranış gösterirler. Diğer taraftan, amino asit sayısı 40'tan düşük olan bir kaç protein (örneğin, 1BBL) bariyersiz (downhill) davranışı göstermektedir. Malesef proteinlerin bu farklı davranışlarını deneylerle örtüşecek şekilde betimleyecek etkileşim tasvirleri hala tam olarak oluşturulamamıştır. Bu nedenle bu tez çalışmasında geliştirdiğimiz protein modelinin paralelleştirilmesi ve etkileşim yoğunluğunun protein dinamiğine katkıları incelenecektir.

2.2.5.1. Levinthal paradoksu

Proteinlerde amino asitlerin birbirine kovalent bağ ile bağlı olmasına ve dolayısıyla serbestlik derecelerini (girilebilecek durum sayısını) düşürse bile, toplam amino asit sayılarına bağlı olarak girilebilecek konformansyonların (durum) sayısı çok fazladır. Bir amino asidin geometrik olarak sadece 3 farklı duruma girebileceği farz edildiğinde, gayet küçük sayılabilecek 50 amino asit içeren bir proteinin ziyaret edebileceği yaklaşık 10^{24} farklı konformasyon vardır. Proteininin katlı yani kararlı hali bu konformasyonlardan sadece bir tanesidir. Eğer konformasyonların hepsinin eşit olasılık ile ziyaret edilebileceğini düşünecek olursak, katlı yapıya ulaşılması inanılmaz uzun zaman alabilecektir. Yukarıda verilen örnek için her konformasyonel geçişin $10^{-14}\ s$ aldığı düşünülürse, bütün konformasyonların ziyaret edilerek katlı yapıya ulaşma süresi $10^{24} \times 10^{-14} = 10^{10} s \cong 317$ yıl alması gerekir. Oysa ki deneysel çalışmalarda, proteinlerin katlanma süresinin ortalama olarak mikro saniyeler ile mili saniyeler mertebesinde gerçekleştiği gözlemlenmiştir. Gözlemlenen bu farklılık Levinthal

paradoksu olarak adlandırılmaktadır. Bu gözlemler bize konformasyonel geçişlerin gelişi güzel olmadığını, tam tersine proteinleri katlı yapıya sürükleyen bir eğilim olduğunu söylemiştir.

2.3. Simülasyon Teknikleri

Bilgisayara dayalı modelleme ve simülasyon (benzeşim) yöntemleri, gerçek hayatta yapılması pahalı veya kısmen imkansız problemlerin, belirli bir takım yaklaşımlar yardımı ile çözülmesine dayalı yöntemlerdir. 1975 yılından itibaren protein katlanması alanında bilgisayara dayalı hesaplamalar yaygın olarak (ref computer simulation of protein folding) kullanılmaya başlanmıştır. En yaygın kullanılan simülasyon teknikleri Monte Carlo ve Moleküler Dinamik simülasyon teknikleridir.

2.3.1. Monte Carlo simülasyon yöntemi

Monte Carlo (MC) simülasyon yaklaşımı stokastik bir yöntemdir. Yani, bu yöntemde dinamikte karar verici yapıya sahip tüm hareketler ve hareket mekanizmaları gelişi güzel bir şekilde gerçekleşmektedir. Zincir temsili içeren protein modellerinde hareket setleri konformasyon uzayını efektif örnekleme gerçekleştirecek şekilde tanımlanmalıdır. Konformasyonel geçişler bu hareket setleri üzerinden gelişi güzel olarak gerçekleştirilir. Ama gerçekleştirilen bir hareketin veya konformasyonel geçişin kabul edilip edilmeyeceği Metropolis algoritmasına göre belirlenmektedir. Eğer gelişigüzel seçilen hareket ile m konformasyonundan, n konformasyonuna gidildiğinde; gidilen n konformasyonunun enerjisi daha düşük ise hareket koşulsuz kabul edilir ve n konformasyonuna geçiş tamamlanır. Eğer gidilen n konformasyonunun enerjisi daha yüksek ise o geçişin kabul edilip edilmemesi; m ve n konformasyonlarının enerji farkı ve simülasyon sıcaklığı ile tanımlanan bir fonksiyona bağlı olarak yine rastgele şekilde belirlenir. Konformasyonel geçişlerin kabul veya ret edilmesini belirleyen Metropolis kriterinin veya olasılığının matematiksel formu aşağıda verilmiştir.

$$p_{mn} = \min\left\{1, \exp\left(-\frac{E(n) - E(m)}{k_B T}\right)\right\}$$

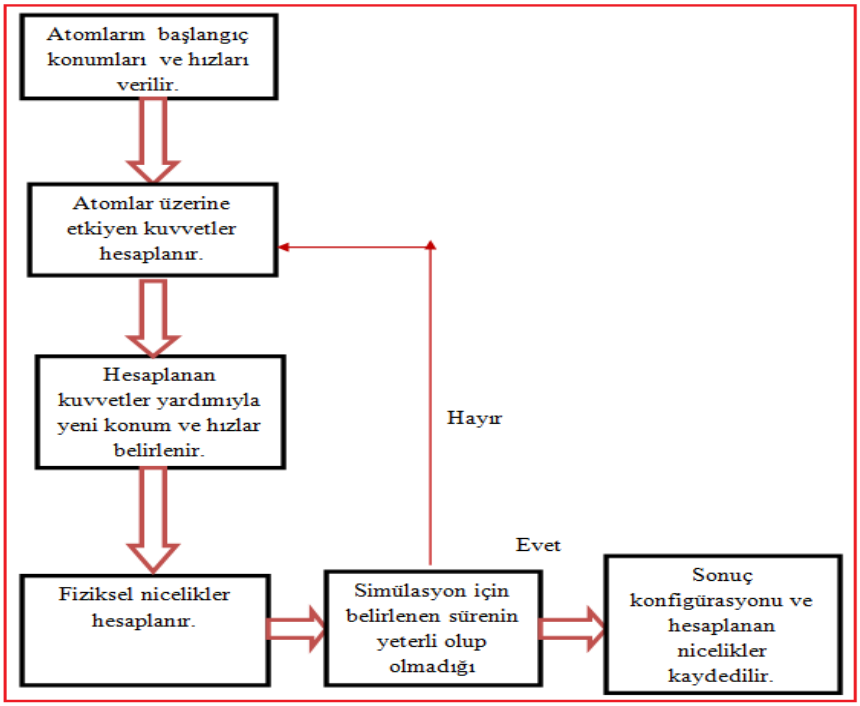
Monte Carlo simülasyon yöntemi kullanılarak gerçekleştirilen protein modelleme çalışmalarında karşılaşılan en temel zorluk hareket setlerinin oluşturulmasıdır. Diğer taraftan, hareket ettirilecek atom sayısı da gelişi güzel belirlendiği için proteinlerin

zamana bağlı davranışlarını tam betimlemek mümkün olmamaktadır. Fakat proteinlerin termodinamik davranışlarının incelenmesi ve yeni etkileşim senaryolarının test edilebilmesi bakımından moleküler dinamik simülasyon yönteminden daha efektiftir.

2.3.2. Moleküler dinamik simülasyon yöntemi

Moleküler dinamik (MD), her parçacığın hareket denkleminin ayrı ayrı çözüldüğü, deterministik bir simülasyon yöntemidir. Alder ve Wainwright 1950 yılında bu metodu ilk defa kürelerin etkileşimini hesaplamak için kullanmıştır. Bu method A. Rahman tarafından Argon sıvısı için gerçek etkileşim simülasyonu için kullanıldığında oldukça ilgi çekmiştir. Biyolojik sistemlerde; protein kararlılığı, konformasyonel değişimler, protein katlanması, molekül tanıması, biyolojik sistemlerde iyon geçişleri gibi birçok alanda moleküler dinamik simülasyon tekniği kullanılmaktadır.

Moleküler dinamik simülasyon algoritması şekil 2.5'deki gibidir.



Sekil 2.5 Moleküler dinamik simülasyon akış şeması

Newton'un ikinci kanunu moleküler dinamik simülasyon tekniğinin temelini oluşturulmaktadır.

$$F_i = m_i a_i = -\nabla_i V \qquad 2.1$$

F bir parçacığa etki eden kuvvet, m parçacığın kütlesi ve a da parçacığın ivmesidir. Parçacığa etki eden kuvvet F, parçacığın potansiyel enerjisi $(V)'$nin negatif gradiyenti alınarak hesaplanmaktadır.

2.3.3. Protein spesifik Go modeli yaklaşımı

Mevcut model yaklaşımları arasında protein katlanma davranışlarını ve işlevlerini anlamak için yaygın olarak Go modeli kullanılmaktadır. Nobuhiro Go bu modeli ilk kez 1983 yılında protein çalışmaları için kullanmıştır. Fakat Go modeli, iki durum davranışı gösteren proteinler ele alınarak; kristal yapıda etkileşen amino asitlerin zincir üzerindeki mesafelerinin ortalama ölçüsü olarak tanımlanan etkileşim düzen parametresi (Contact Order, CO) ile deneysel katlanma oranları arasında yüksek korelasyon olduğu gösterilmesinden sonra yaygın olarak kullanılmaya başlanmıştır.[7] Bu gözlem bize katlı yapıdaki etkileşimlerin daha dominant olduğunu işaret etmektedir.

Go modeli kristal yapı bilgisine dayalı bir modeldir ve katlanmış yapı en düşük enerjili yapıdır. Bu model yaklaşımında basitleştirilmiş zincir temsilleri için proteindeki amino asitler kadar farklı amino asit, tüm atomların temsil edildiği zincir modellerde atom sayısı kadar farklı atom türü olduğu kabul edilir.

2.3.4. Kuvvet alanları

Dinamik bir sistemde parçacıkların etkileşim tasvirleri ve enerjileri o sistemin kuvvet alanlarının belirlenmesini sağlar. Yukarıda ifade edildiği gibi potansiyel enerjinin negatif gradiyenti kuvvete eşittir $(F = -\nabla V)$. Dolayısıyla parçacıkların yer değiştirmesi veya diğer bir deyişle konformasyonel geçişler tamamen kuvvet alanları tarafından belirlenmektedir. Kullanılan etkileşim tasvirleri yanlış veya yeterli değilse geliştirilen model deneylerle uyuşmayan sonuçlar verir.

Yapılan bu tez çalışmasında protein katlanması dinamiği Go modeli yaklaşımı kullanılarak incelenecektir. Proteinlerin konformasyon enerjileri aşağıda verilen matematiksel eşitlik ile hesaplanacaktır.

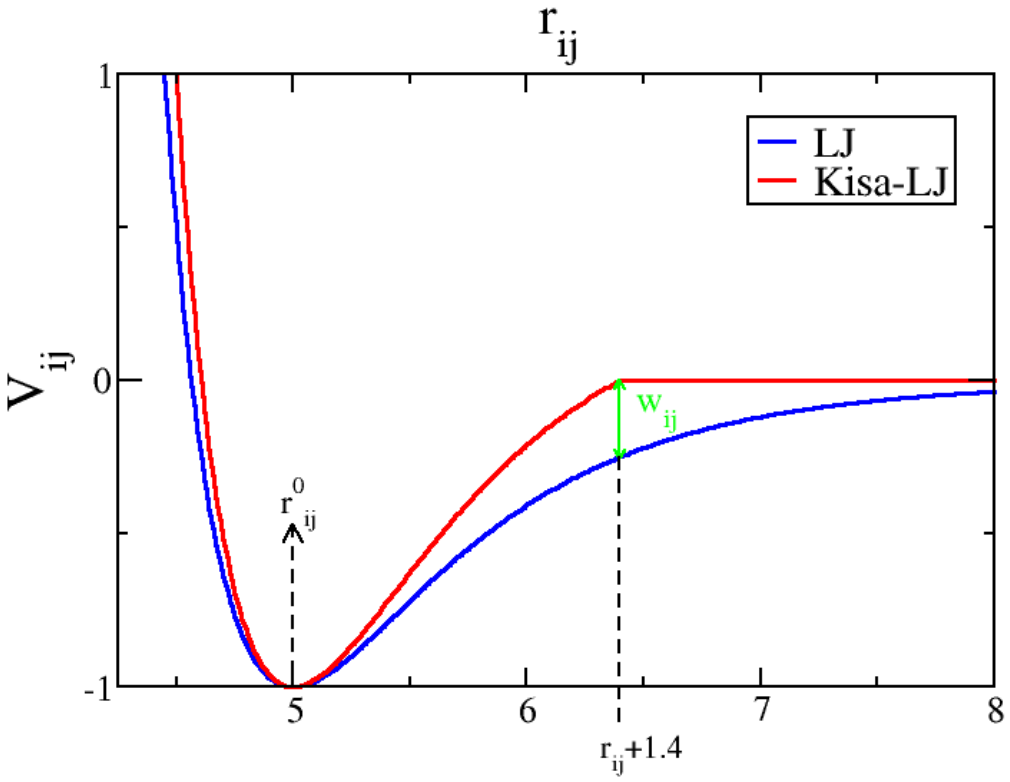
$$V_{\text{toplam}} = V_{\text{gerilme}} + V_{\text{eğilme}} + V_{\text{bükülme}} + V_{\text{bağsız}}$$

$$= \sum_{\text{bağlar}}^{N-1} K_r (r - r_0)^2 + \sum_{\substack{\text{eğilme} \\ \text{açıları}}}^{N-2} K_\theta (\theta - \theta_0)^2$$

$$+ \sum_{\substack{\text{bükülme} \\ \text{açıları}}}^{N-3} \left\{ K_\emptyset^1 [1 - \cos(\emptyset - \emptyset_0)] + K_\emptyset^3 [1 - \cos 3(\emptyset - \emptyset_0)] \right\}$$

$$+ \sum_{i<j-3}^{\text{katlı}} \varepsilon_{ij}^{\text{katlı}} \left[5 \left(\frac{r_{\text{itici}}}{r_{ij}} \right)^{12} - 6 \left(\frac{r_{\text{itici}}}{r_{ij}} \right)^{10} \right] + \sum_{i<j-3}^{\substack{\text{katlı} \\ \text{olmayan}}} \varepsilon_{ij}^{\substack{\text{katlı} \\ \text{olmayan}}} \left(\frac{r_{\text{itici}}}{r_{ij}} \right)^{12}$$

Yerel etkileşimler protein zinciri üzerinde birbirine takip eden amino asitler arasında etkileşimler olarak tanımladığımızda, yukarıdaki eşitliğin ilk üç terim yerel etkileşimleri ifade etmektedir. Bunlar sırasıyla gerilme, eğilme ve bükülme terimleridir. Eşitlikte r_0, θ_0 ve $\emptyset_0$ parametreleri proteinin katlanmış yapısındaki denge mesafesi, denge eğilme açısı ve denge bükülme açılarını temsil etmektedir. K_r, K_θ, $K_\emptyset^1$ ve $K_\emptyset^3$ parametreleri ise etkileşim büyüklükleridir. Bu etkileşim büyüklükleri sırasıyla 100ε, 20ε, ε ve 0.5ε değerlerini almaktadır [8,9]. Yapılan tüm matematiksel işlemlerde $\varepsilon = 1$ olarak alınmıştır. Yukarıda verilen eşitlikte geriye kalan tüm etkileşmeler bağsız etkileşmeleri tasvir eder. Bağsız etkileşmeler zincir üzerinde birbirine en az 4 amino asit mesafede bulunan amino asitler arasında hesaplanmaktadır. Bağsız etkileşmelerin ilki olan 4.ncü terim 10-12 Lennard-Jones etkileşim potansiyelidir. Bu potansiyel katlı yapıda olan etkileşimleri tasvir etmek için kullanılmıştır. Eşitlikte $\varepsilon_{ij}^{\text{katlı}}$ katlanmış yapıda olup birbiri ile etkileşen amino asit çiftleri için etkileşim büyüklüğünü ifade etmektedir ve değeri 1 olarak alınmıştır. 5'nci terim katlı yapıda birbirleriyle etkileşmeyen amino asit çiftlerinin konformasyonel geçişler sırasında birbirlerine çok yaklaşmasını engellemek için kullanılmıştır. Bu terimde $\varepsilon_{ij}^{\substack{\text{katlı} \\ \text{olmayan}}} = 1$ ve $r_{\text{itici}} = 4$ Å olarak alınmıştır.



Sekil 2.6 Kısa ve Uzun Erişimli Lenard Jones Etkileşmesi

Şekil 2.6 da kırmızı çizgi 10-12 Lennard-Jones potansiyelini temsil etmektedir. Lennard-Jones potansiyeli $r \to \infty$ yaklaşırken $U(r) = 0$ değerine sahip olmaktadır. Amino asitler uzak mesafelerde olsalar dahi birbirlerini hissedebilmektedirler. İki amino asit arasındaki mesafe r, protein kristal yapısından elde edilen denge mesafeleri r_{ij}^0 ile gösterilirse; bu iki amino asidin kontak halinde olup olmadığı $r < r_{ij}^0 + 1.4$ şartını gerçekleştirip gerçekleştirmediğine bağlı olarak tespit edilir. [10]

Bu tez çalışmasında kısa erişimli etkileşim Lennard-Jones potansiyeli kullanılmıştır. Bu potansiyel uzun erişimli 10-12 Lennard-Jones etkileşme potansiyelinin $r = r_{ij}^0 + 1.4$'de $U\left(r = r_{ij}^0 + 1.4\right) = 0$ olmasını sağlayacak şekilde aşağıda verilen denklemde ki gibi yeniden düzenlenmiştir. Bu sayede $r > r_{ij}^0 + 1.4$ 'den itibaren etkileşim olmaması sağlanmıştır. [10]

$$E_{ij}^{\text{Kısa-LJ}}(r_{ij}) = \begin{cases} \dfrac{E_{ij}^{\text{LJ}}(r_{ij}) - \varepsilon_{ij}\omega_{ij}}{1-\omega_{ij}}, & \text{eğer } \; r_{ij} \le r_{ij}^{\max} \\ 0, & \text{eğer } \; r_{ij} > r_{ij}^{\max} \end{cases}$$

Eşitlikte $r_{ij}^{\max} = r_{ij}^{0} + 1.4$ ve $\omega_{ij} = 6\left(r_{ij}^{\max}/r_{ij}\right)^{10} - 5\left(r_{ij}^{\max}/r_{ij}\right)^{12}$'dir.

3. GEREÇ ve YÖNTEM

3.1. Uygun Paralelleştirme Platformu Seçimi

Paralel programlama donanımlarının hızlı gelişmesi ve yeni nesil teknolojilerin çıkması, paralel programlamada uygun donanım ve platformların seçimini zorlaştırmaktadır. Özellikle, bilimsel problemlerin çözümünde sürdürebilirlik, genişletilebilirlik, modülerlik gibi konuların düşünülüp planlanması, problemin çözümünü kolaylaştıracak, maliyet ve zaman tasarrufu sağlayacaktır. Bu yüzden uygun platform seçimi, problemin çözümü için kritik öneme sahiptir.

3.2. Cuda ile Programlama

Cuda ile program yazabilmemiz için öncelikle Cuda grafik kartlarının donanım özelliklerini iyi anlamamız gerekir. İyi bir Cuda programı, grafik belleğini en optimum kullanan programdır. Her bir program için açılacak iş parçacığı sayısı, CPU ile karşılaştırıldığında, GPU'larda oldukça fazladır. Fakat bu çekirdeklerin bellek okumalarında ki gecikme CPU'dan çok daha fazladır.

3.2.1. Donanım özellikleri

Cuda grafik kartları, duraksız çok çekirdek (SMX) bölümlerinden oluşur. Her bir SMX'te 192 adet cuda çekirdeği vardır. Bu çekirdeklerden her biri yaklaşık 1Ghz hızında çalışırlar. SMX'de toplamda 65536 adet 32 bit yazmaç (register) bulunur. Her bir SMX de toplam; 64KB paylaşımlı L1 bellek, 48KB sadece okuna bilir bellek, ve 48KB doku (texture) belleği vardır.

Gelişmiş bir Cuda kartında 15 adet SMX bulunur. Yani bu kartta toplamda 2688 adet Cuda çekirdeği vardır. Bu çekirdekler için ayrılmış 1.5 MB L2 önbellek vardır. Ayrıca 384bit GDDR5 veri yollu bütün çekirdeklerin kullanabileceği genel bellek vardır. Bu bellek alanı Gpu da en yavaş ulaşılan alandır. GDDR5 bellek miktarı karttan karta değişmek ile birlikte güncel modellerde 12GB kadar çıkmıştır.

3.2.2. Derleyici ve Kütüphaneler

Cuda programlarını derlemek için Nvidia tarafından özel olarak geliştirilen NVCC derleyicisi kullanılır. Derleyicinin yaygın kullanılan, Linux, Mac-OS ve Microsoft işletim sistemleri uygun sürümleri vardır. Nvidia derleyiciyi ücretsiz olarak herkesin kullanımına açmıştır. Bu derleyici hem CPU tarafında hem de GPU tarafında çalışacak kodu tek başına derler. Fakat CPU tarafında çalışacak kodun derlenmesinde işletim sisteminde varsayılan C derleyicisini kullanır. Cuda için en yaygın kullanılan programlama dili C dir. NVCC derleyicisi, gnu C derleyicisi, intel C derleyicisi, Microsoft C derleyicisi gibi yaygın olarak kullanılan C derleyicileri ile uyumludur.

Cuda avantajlarından yararlanmak için bir çok programlama ve betik dillerine uygun kütüphaneler yazılmıştır. Fortran, Haskell, Java, Perl, Python, Ruby bunlardan bazılarıdır. Ayrıca yüksek seviye programlama dillerinde, akademik olarak en fazla kullanılan Mathematica ve Matlab'ın Cuda desteği vardır.

3.2.3. Programlama yapısı: tek talimat çok data

Cuda ile programlamada, tek talimat çok data yapısı kullanılır. Bu yapıda bütün çekirdeklerde kullanılacak kod talimatları aynıdır, fakat çekirdekler farklı datalara aynı talimatları uygular.

Bu yapıyı bir örnek yardımı ile daha detaylı inceleyelim. A ve B tek boyutlu birer dizi olsun. Amacımız A ve B dizisindeki elemanları toplayıp bir C dizisine atayan Cuda programı yazmak olsun. Buradaki her çekirdekte çalışacak talimat aşağıdaki formdadır.

```
C_i = A_i + B_i
```

Her çekirdek, yukarıdaki tek bir talimatı dizinin farklı elemanları için uygular. Verilen talimatlar çekirdeklere eş zamanlı dağıtıldığından, farklı direktiflere sahip talimatlar bütün çekirdeklerin aynı anda işlem yapmasını garanti etmez. Örnek olarak aşağıdaki talimatı inceleyelim.

```
eğer i tek sayı ise
      C_i = A_i + B_i
eğer i çift sayı ise
      C_i = A_i - B_i
```

Verilen örnekte i sayısının tek veya çift olmasına göre bazı çekirdekler toplama işlemi yaparken bazı çekirdekler çıkarma işlemi yapmaktadırlar. Bütün çekirdeklere aynı talimatlar bildirildiği için, i sayısının tek olduğu durumda, i sayısının çift olan çekirdekler boşta olarak beklemektedir. i sayısının çift olduğu durumda ise, i sayısının tek olan çekirdekler boşta olarak beklemektedir. Bu durum tek talimat çok data yapısının dezavantajlarından biridir. Boşta bekleyecek çekirdekleri minimuma indirmek için, programdaki hesapların buna göre sınıflandırılması gerekmektedir.

3.2.4. Blok yapısı

Cuda'da çalışan iş parçacıkları bloklar halinde işlenir. Farklı bloklarda çalışan iş parçacıklarını senkronize etmek mümkün değildir. Sadece aynı blokta olan iş parçacıkları senkronize edilebilir. İş parçacıkları oluşturulmasından yok edilmesine kadar aynı blok içinde olmalıdır. Yani, çalışan bir iş parçacığını farklı bir bloğa taşımak mümkün değildir.

Bloklar aynı anda sadece bir SMX kullanabilirler ve bloklar çalışırken SMX değiştiremezler. Bir blokta çalışabilecek iş parçacığı sayısı 1024'ü geçemez. Birden fazla GPU kartı arasında blok oluşturulması iyi sonuç vermez.

3.2.5. Örnek Cuda programı yazılması

Her programlama dilinde verilen en basit temel örnek, ekran sadece çıktı veren "Merhaba Dünya" programıdır. Cuda ile programlamanın amacı, fazla hesap kapasitesi gerektiren programların hızlı çalışması olduğu için, cuda öğrenmek isteyen kullanıcılar genellikle bir veya daha fazla programlama dilene hakim kişilerdir. Bunun için bu bölümde tek boyutlu iki dizinin toplanmasını sağlayan örnek bir Cuda programı yazacağız. Direk kodlamaya başlamadan önce, her Cuda kodunda sık kullanılan temel kavramları anlamamız gerekmektedir.

Her Cuda kodunun CPU ve GPU tarafında çalışan iki kısmı bulunur. CPU tarafında, çalışan kısmı, işletim sistemi ile ilgili olan protokollerin hazırlanmasını, hafıza ayrılmasını, GPU ve CPU hafızaları arasındaki veri transferini, GPU kodlarının çalıştırılması, ve program için sonuçların işlenmesi ile ilgili diğer seri işleri yapmak ile yükümlüdür. Yani GPU kodları bir CPU kodu olmadan tek başlarına çalışamazlar.

Şimdi, ön tanımlı tipleri, fonksiyon tiplerini, ön tanımlı değişkenleri, ve bellek yönetimini alt başlıklarda inceleyelim.

3.2.5.1. Cuda hata tipi

Cuda'da olası hataları yönetebilmek için ön tanımlı hata tipi vardır. Bu tipin ismi cudaError_t dir. Bir çok Cuda fonksiyonu geriye dönüş tipi bu tiptedir. Bu tipte, sıfır herhangi bir hata olmadığını temsil eder. Toplamda 36 farklı hata tanımı vardır.

Programda oluşabilecek hataların takibinin kolaylaşması için, hata tipinin yanında, hataların tertip edildiği ayrı bir mekanizma mevcuttur. Bu mekanizma sayesinde, herhangi bir fonksiyondan, en son oluşan hata tipinin değeri çağrıla bilir ve bu değer sıfırlanabilir. Böylelikle programlamanın herhangi bir adımında, bir önceki adıma ait hata kontrolü yapmak çok kolaydır. Aşağıda bu görevi yapan iki fonksiyonun prototipi verilmiştir.

```
cudaError_t cudaPeekAtLastError ()
cudaError_t cudaGetLastError ()
```

3.2.5.2. Cuda fonksiyon çeşitleri

Cuda'da fonksiyonların çağrıldığı ve çalıştığı yerin değişmesine göre 3 farklı fonksiyon tipi vardır. Bu tipler __device__, __global__ ve __host__ ismini almıştır. Tablo 3.1'de fonksiyonların çağrıldığı ve çalıştığı yer ile ilgili detaylı bilgi verilmiştir. Tablo 3.1'den

Tablo 3.1 Cuda fonksiyon tipleri

Fonksiyon Tipi	Çalıştığı yer	Çağrıldığı yer
__device__	GPU	GPU
__global__	GPU	CPU
__host__	CPU	CPU

de görülebildiği üzere, __host__ fonksiyonunun çağrıldığı ve çalıştığı yer CPU yer olduğu için, normal C fonksiyonundan farkı yoktur.

Cuda fonksiyonlarının yazım prototipi; fonksiyon tipi, dönüş veri tipi, fonksiyon ismi ve girdi parametreleridir. Aşağıda örnek prototipler verilmiştir.

```
__device__ float benim_toplamam (float sayi1, float sayi2 );
__global__ void benim_fonksiyonum ();
__host__ float benim_carpmam (float sayi1, float sayi2);
```

Kuşkusuz paralel programlama için en önemli fonksiyon tipi __global__ dir. Bu fonksiyon CPU'dan GPU ya iş gönderilmesini sağlayan asıl fonksiyondur. Bu tipi diğer fonksiyon tiplerinden ayıran bir diğer özellik geriye herhangi bir değer döndürememesidir. __global__ tipindeki fonksiyonların dönüş tipi void olmak zorundadır. Diğer fonksiyon tipleri için bu konuda herhangi bir kısıtlama yoktur.

3.2.5.3. Ön tanımlı değişkenler

Daha önce belirttiğimiz gibi Cuda, tek talimat çok veri yapısına uygundur. Talimatlar her iş parçacığında aynı olduğundan, verinin farklı elemanlarına ulaşılması, ön tanımlı değişkenler sayesinde mümkündür. Bu değişkenler her iş parçacığı için benzersizdir. Böylece verinin aynı yerine birden fazla iş parçacığının aynı işlemi yapması önlenmiş olur.

İş parçacıkları için iki önemli ön tanımlı değişken, blockIDx ve threadIDx dir. blockIdx iş parçacığının hangi blokta olduğunu belirtir. threadIDx ise o bloktaki kaçıncı iş parçacığına ait olduğunu belirtir.

blockIDx ve threadIDx değişkenleri, hesaplamaların 2 veya 3 boyutlu yapıldığı durumlarda, hesaplamalar arasında ilişkinin doğru ve kolay tasvir edilmesi için 3 boyutlu olarak tanımlana bilir. Burada Cuda da ön tanımlı 3 boyutlu yapılar vardır. Mesela aşağıda ki kod örneğinde, 10*10*10 büyüklüğünde 3 boyutlu bir blok oluşturulmaktadır. İş parçacıklarının 3 boyutlu yapıdaki farklı blockIDx'lerine x,y,z elemanları sayesinde ulaşılır.

```
dim3 Block (10,10,10);
blockIDx.x;
blockIDx.y;
blockIDx.z;
```

blockIdx ve threadIdx'e ek olarak her iş parçacığında, gridDim blockDim değişkenleri bulunur. Bu değişkenler iş parçacığına çalışan işlerin blok ve grid büyüklüğü ile ilgili bilgi verirler.

3.2.5.4. Bellek yönetimi

Cuda ile programlamada, CPU bellekleri ile birlikte GPU'daki bellekleri de yönetmemiz gerekmektedir. Bunun temel sebebi Cuda çekirdeklerinin, sistem belleğine doğrudan erişme ve yazma yetkisinin olmaması, ve bu işlemin çok zaman almasıdır. Cuda programı yazarken, gerekli bellek miktarı dinamik olarak GPU paylaşımlı belleğinden ayrılmalı, ve ilgili datamız bu alana kopyalanmalıdır. Bu işlemleri yapmak için iki fonksiyon kullanırız.

cudaMalloc fonksiyonu, GPU dan bellek ayırmak için gerekli olan fonksiyonumuzdur. Bu fonksiyonun kullanılış yöntemi C deki malloc fonksiyonuna çok benzemektedir. Parametre olarak verilen işaretçiye, istenilen miktarda bellek ayrılmasını sağlar. Bu fonksiyonun prototipi aşağıdaki gibidir.

```
cudaError_t cudaMalloc (void** isaretci, size_t size)
```

Bu fonksiyonun geri dönüş tipi cudaError_t dir. Bu tip önceki bölümlerde daha detaylı olarak açıklanmıştır. Fonksiyonun aldığı ilk parametre bellek ayrımı yapılacak işaretçiyi, ikinci parametre ise, ayrılacak bellek boyutunu belirler. İşaretçiler C programlamada da kullanılan, bellek adreslerin yerini tutan değişkenlerdir. Daha öncede anlatıldığı gibi GPU da çalışacak Cuda fonksiyonları ilk olarak CPU üzerinden çağrılır. Bunun için GPU bellek ayrımı yapan fonksiyonlarının işaretçileri CPU belleği üzerinde bulunur.

Cuda da, C programlamada olduğu gibi, her dinamik bellek ayrımı için, program sonlandırılmadan önce o alanı serbest bırakmamız gerekmektedir. Bu işlemi gerçekleştiren fonksiyon cudaFree fonksiyonudur. Bu fonksiyonun prototipi aşağıdaki gibidir.

```
cudaError_t cudaFree (void* isaretci);
```

Bir çok Cuda fonksiyonunda olduğu gibi bu fonksiyonunda geri dönüş tipi, cudaError_t dir. Bu tip önceki bölümlerde detaylı anlatılmıştır.

Cuda da bellek yönetiminde, bellek yerlerinin ayrılmasına ek olarak bilmemiz gereken kopyalama fonksiyonlarıdır. Cuda da sadece bir tane kopyalama fonksiyonu vardır ve ismi cudaMemcpy dir. Bu fonksiyon bizim bütün kopyalama ihtiyacımızı karşılayacak şekilde tasarlanmıştır. Bu fonksiyonunun prototipi aşağıdaki gibidir.

```
      cudaError_t cudaMemcpy ( void* hedef, const void* kaynak, size_t
boyut, cudaMemcpyKind tip);
```

Fark ettiğiniz üzere, diğer fonksiyonlarda olduğu gibi bu fonksiyonunda geri dönüş tipi cudaError_t tipindedir. "kaynak" kopyalanacak belleğin başlangıç adresini gösterir. "hedef" kopyalanacağı yerin başlangıç adresini gösterir. Boyut kopyalamanın ne kadar bellek kopyala cağını gösterir. C dilinde olduğu gibi, hedef adresteki bellek boyutunun kopyalama için yeterli olup olmadığını kontrol etmek programcının işidir. Daha küçük boyuta sahip bellek alanlarına yapılacak kopyalamalar doğru sonuç vermeyecektir.

Bu fonksiyonda yeni olarak Cuda da ön tanımlı cudaMemcpyKind tipini görmekteyiz. Bu tip kopyalama fonksiyonun nereden nereye kopyalama yapacağını gösteren tiptir. Kaynak bellek yerinin CPU veya GPU'da olmasına ve, hedef bellek alanının CPU'da veya GPU'da olmasına göre dört farklı değer almaktadır. Bu değerler ve karşılık gelen değişken isimleri Tablo 3.2 da detaylı olarak incelenmiştir.

Tablo 3.2 cudaMemcpy tipi değerlerinin çeşitleri

Veri değeri	Kaynak Bellek Alan	Hedef Bellek Alan
cudaMemcpyHostToHost	CPU	CPU
cudaMemcpyHostToDevice	CPU	GPU
cudaMemcpyDeviceToHost	GPU	CPU
cudaMemcpyDeviceToDevice	GPU	GPU

3.2.5.5. Örnek program

Aşağıda iki sayı dizisini toplayan Cuda programının kodu örnek olarak verilmiştir.

```
      #include <stdio.h>
      #include <stdlib.h>
      #define n 100
      __global__ void vectorAdd(double *A, double *B, double *C, int
size){
```

```
            int index = threadIdx.x + blockIdx.x * blockDim.x;
            if (index<size)
                  C[index] = A[index] + B[index];
      }
      int main(int argc, char **argv){
            double *hA, *hB, *hC, *dA, *dB, *dC;
            hA = (double *) malloc(n*sizeof(double));
            hB = (double *) malloc(n*sizeof(double));
            hC = (double *) malloc(n*sizeof(double));

            cudaMalloc(&dA, n*sizeof(double));
            cudaMalloc(&dB, n*sizeof(double));
            cudaMalloc(&dC, n*sizeof(double));

            for (int i = 0; i < n; i++){
                  hA[i] = i;
                  hB[i] = n - i;
            }

            cudaMemcpy(dA, hA, n*sizeof(double),
cudaMemcpyHostToDevice);
            cudaMemcpy(dB, hB, n*sizeof(double),
cudaMemcpyHostToDevice);

            dim3 block(512);
            dim3 grid((n - 1) / 512 + 1);
            vectorAdd<<<grid, block>>>(dA, dB, dC, n);


            cudaMemcpy(hC, dC, n*sizeof(double),
cudaMemcpyDeviceToHost);

            for (int i = 0; i < n; i++)
                  printf("%f + %f = %f\n", hA[i], hB[i], hC[i]);

            cudaFree(dA);
            cudaFree(dB);
            cudaFree(dC);

            free(hA);
            free(hB);
            free(hC);
            return 0;
      }
```

Şimdi bu örnek kodu parça parça inceleyelim.

```
__global__ void vectorAdd(double *A, double *B, double *C, int size){
      int index = threadIdx.x + blockIdx.x * blockDim.x;
      if (index<size)
      C[index] = A[index] + B[index];
```

```
}
```

Bu programa baktığımızda, bir adet vectorAdd isminde Cuda fonksiyonu bulunmaktadır. Bu fonksiyon CPU'dan çağrılan, işlemlerin ise GPU tarafından yapıldığı bir fonksiyondur. Bu fonksiyonda ki "index" değişkeninin içinde her iş parçacığına ait eşsiz bir tam sayı değişkeni tanımlanır. Görüldüğü gibi bu tam sayının değeri atanırken, Cuda'da ki ön threadIdx, blockIdx, ve blockDim değişkenlerinden yararlanılmıştır. Yani normal seri programlamada sıkça kullanılan döngülerdeki indeksleme yerine her iş parçacığının eşsiz indeksi kullanılarak işlem yapılmaktadır.

```
int main(int argc, char **argv){
      double *hA, *hB, *hC, *dA, *dB, *dC;
      hA = (double *) malloc(n*sizeof(double));
      hB = (double *) malloc(n*sizeof(double));
      hC = (double *) malloc(n*sizeof(double));

      cudaMalloc(&dA, n*sizeof(double));
      cudaMalloc(&dB, n*sizeof(double));
      cudaMalloc(&dC, n*sizeof(double));
```

Ana fonksiyonumuza baktığımızda, ilk önce toplama işlemi yapacağımız dizilerin işaretçilerinin tanımlanmasını görmekteyiz. Bu altı değişkenin üçü CPU'da kullanmak üzere, diğer üçü de GPU da kullanmak için tanımlanmıştır. Sonraki adımda bu ilgili değişkenlere yeterli bellek alanları sırayla ayrılıyor.

```
for (int i = 0; i < n; i++){
      hA[i] = i;
      hB[i] = n - i;
}
```

Programda toplanacak A ve B dizilerinin elemanlarını sırasıyla bir değer veriyoruz. Anlamlı bir toplama işlemi gerçekleştirebilmek için, A'nın elemanlarını sıfırdan birer birer artırarak, B'nin elemanlarını $n-1$ den birer birer azaltarak atıyoruz. Yani toplama işleminin sonucunda oluşacak C dizisinin her bir elemanın değeri n değerine eşit olmalıdır. Bu işlem CPU'da gerçekleşmektedir. n programda derleyici direktifi olarak programa verilmektedir. Bizim örneğimizde n değeri 100 olarak alınmıştır.

```
cudaMemcpy(dA, hA, n*sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(dB, hB, n*sizeof(double), cudaMemcpyHostToDevice);
```

İçi sayılarla dolu olan A ve B dizilerimizi, daha önceden GPU da ayırdığımız bellek yerlerine kopyalıyoruz. Buradaki kopyalama işlemi CPU dan GPU ya olduğu için cudaMemcpyHostToDevice değeri kullanılmıştır.

```
dim3 block(512);
dim3 grid((n - 1) / 512 + 1);
vectorAdd<<<grid, block>>>(dA, dB, dC, n)
```

Bizim için en önemli ve GPU da hesap yapılan kısım bu kısımdır. İlk iki satırda blok ve grid boyutları bir değişkene atanmıştır. Son satırda, daha önceden yazmış olduğumuz "vectorAdd" fonksiyonu çağrılmıştır. Bu fonksiyon çağrılırken üç tane küçüktür ve büyüktür (<<< >>>) işareti arasına fonksiyonun çalıştırılacağı blok ve grid sayısı verilir. Üsteki iki satırda bu değerleri bir değişkene atadığımız için direk değişkenler parametre olarak girilmiştir. Bu değerler çağrılan fonksiyonun kaç adet iş parçacığında çalışacağını gösterir. Kodda verilen örnekte blok boyutu 512, grid boyutu 1 olarak verilmiştir. Yani toplamda 512 adet iş parçacığı çalışacaktır.

```
cudaMemcpy(hC, dC, n*sizeof(double), cudaMemcpyDeviceToHost);
for (int i = 0; i < n; i++)
      printf("%f + %f = %f\n", hA[i], hB[i], hC[i]);
```

GPU ile işlem yapmayı bitirdikten sonra hesaplanan verileri tekrar CPUya almamız gerekmektedir. Bunun sebebi, işletim sistemi ile ilgili özellikleri GPU üzerinden kullanamayışımızdır. Yani ekrana yazdırma, dosya çıktısı alma gibi temel fonksiyonları GPU üzerinden kullanamayız. Ayrıca CPU üzerinden direk GPU belleğine erişim yetkimiz yoktur.

İlk satırdaki cudaMemcpy fonksiyonunu GPU'daki verileri CPU ya geri kopyalamak için kullanılmıştır. Sonraki satırda da ise basit bir döngü ile toplanan diziler ve sonuç ekrana bastırılmıştır.

```
cudaFree(dA);
cudaFree(dB);
cudaFree(dC);

free(hA);
free(hB);
free(hC);
return 0;
```

Programın son adımında, kullandığımız bütün bellek alanlarını, teker teker serbest bırakıyoruz. Burada hem CPU tarafında hem de GPU tarafında kullanılan alanları serbest bırakmamız gerektiğini unutmamamız gerekir.

3.3. Moleküler Dinamik Simülasyon Kodu Paralelleştirilmesi

Moleküler dinamik simülasyonun paralelleştirme işlemine, öncelikle düzgün veri yapılarının tasarlanması ile başlanmıştır. Cuda ile yazılan programlarda CPU ve GPU farklı bellek alanları kullandığı için, bu iki birim arasında düzgün haberleşmenin sağlanmasında doğru veri tipilerinin tasarlanması önemli rol oynamaktadır.

3.3.1. Simülasyonda kullanılan protein ile ilgili veri yapı (data structure) tanımları

PDB dosyasında bulunan atomların koordinatlarının ve diğer özelliklerinin öncelikle program tarafından okunup, uygun data yapısının oluşturulması gereklidir. Her programda birden fazla datayı hafızaya almak için gerekli olan durumlarda, bellekten yeterli kadar alanın ayrılması çözülmesi gerek bir problemdir. Proteinlerde zincir uzunluğu değişken olduğu için, bu yapı için oluşturacağımız bellek alanının statik olarak ayrılması mümkün değildir. Bu sebepten protein bilgilerinin saklandığı bu yapı bellekte dinamik olarak saklamak mecburiyetindeyiz.

Gerekli hafıza boyutunun bilinmediği durumlarda, birden fazla hafıza biriminin birbirine bağlanarak oluşturulan bellek yapılar mevcuttur. Fakat bu yapılar hafızada bütün verilerin art arda saklandığı dizi yapılarına nazaran daha yavaş çalışmaktadır. Simulasyon programımızda devamlı kullanılacak bu protein yapısının, daha hızlı çalışmasını sağlamak için bellek ayrımını dizi olarak yapmaktayız. Bu dizi yapısını oluşturmak için gerekli pdb dosyasını program tarafından iki kez işlenmekte. İlk okumada pdb dosyasından, gerekli zincir uzunluğu hesaplanmakta. Daha sonraki okumada ise, bu zincir uzunluğunda ayrılan bellek alanına atom bilgileri işlenmektedir.

Proteinlerin sahip oldukları, zincir, amino asit, atom ilişkisini, sanal olarak temsil edecek bir data tasarlanmıştır. Bu yapıların sahip oldukları sanal ilişki sayesinde, hesaplayacağımız fiziksel parametreler daha kolay hesaplana bilmektedir. Programda kullanılan yapılar aşağıdaki gibidir.

```
struct Atom {
```

```
        char name[5];
        int a_id;
        char element[3];
        double mass;
        struct Dim3 coor;
        struct Atom *next, *prev;
        struct Amino_Acid *self;
    };

    struct Amino_Acid {
        int aa_id;
        char name[5];
        double mass;
        struct Atom *atoms;
        int size;
        struct Amino_Acid *next, *prev;
    };

    struct Protein {
        struct Amino_Acid *a_acid;
        int size;
        char name[5];
    };
```

Burada ilk ve en küçük yapı "atom" yapısıdır. Her atom yapısında ismini, atom numarasını, elementini, kütlesinin, ve koordinatlarını saklamak için değişkenler mevcuttur. Ayrıca atom yapısının içinde, bir önceki ve bir sonraki atomu gösteren işaretçiler, ve o atomun hangi amino asitte ait olduğunu gösteren işaretçi bulunmaktadır.

İkici olarak "Amino_Acid" yapısı vardır. Bu yapıda, amino asit numarası, ismi, kütlesi, boyutu, sahip olduğu atomları barındırır. Burada ki atomlar, bir üstteki atom yapısını kullanırlar. Son olarak bütün amino asitleri bir arada, sırayla tutan bir "Protein" yapısı görmekteyiz. Ayrıca, zincir uzunluğu ve pdb kodu bu yapı içerisinde saklanmaktadır.

Yukarıda ki yapıların detayından da anlaşılacağı gibi bütün yapılar iç içe geçmiş bir şekilde sanal bir protein temsil etmeye yönelik oluşturulmuştur. Bu yapılar sayesinde, simülasyon öncesinde ve sırasında gerekli olan fiziksel bilgilerin rahatça ulaşılabilir.

3.3.2. Denge açıları ve mesafeleri

Daha önceki bölümlerde detaylı olarak açıkladığımız üzere, Go model yaklaşımında, kristal yapı en düşük enerjili yapıdır. Bunun için kristal yapıdaki bilgiler mesafeler bizim için denge mesafeleridir. Simülasyona başlamadan önce bu fiziksel bilgiler, hesaplanmalı ve hafızada tutulmalıdır.

Denge mesafelerinin daha kolay hesaplanması için, 3 boyutlu kartezyen koordinatlarında hesap yapan yardımcı fonksiyonlar oluşturulmuştur. Bu fonksiyonlar, bize 3 boyutlu uzayda bulunan noktaların vektörel toplamı, çıkartılması, çarpımını, aralarında ki mesafelerin hesaplanması gibi temel matematiksel işlemlerde yardımcı olacaktır. Bu işlemler, kullandığımız Go modellemesinde, sıkça kullanılmak ile birlikte aynı zamanda ileride geliştirilecek olan potansiyellerde de sıkça kullanılacak işlemlerdir. Bu fonksiyonların kodları aşağıda detaylı olarak verilmiştir.

```
      struct dim3f {
          float x,y,z;
      };

      float Distance (struct dim3f A, struct dim3f B) {
          return sqrtf ( (A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y) +
(A.z-B.z)*(A.z-B.z) );
      }

      struct Dim3 Add (struct dim3f A, struct dim3f B) {
          return { A.x+B.x , A.y+B.y , A.z+B.z  };
      }

      struct Dim3 Subs (struct dim3f A, struct dim3f B) {
          return { A.x-B.x , A.y-B.y , A.z-B.z  };
      }

      struct dim3f Subs (struct dim3f A, struct dim3f B) {
          return { A.x-B.x , A.y-B.y , A.z-B.z  };
      }

      struct dim3f Cross (struct dim3f A, struct dim3f B) {
          struct dim3f result;
          result.x = A.y * B.z  -  B.y * A.z;
          result.y = A.z * B.x  -  B.z * A.x;
          result.z = A.x * B.y  -  B.x * A.y;
          return result;
      }
      float Angle (struct dim3f i, struct dim3f j, struct dim3f k) {
          float eps_theta = 1.0 - 1.0e-12;
          struct dim3f ij = Subs(i, j);
          struct dim3f kj = Subs(k, j);
          float r_ij = Distance(i, j);
          float r_kj = Distance(k, j);

          float cos_theta = ( ij.x * kj.x  +  ij.y * kj.y  +  ij.z *
kj.z )  / ( r_ij * r_kj );

          if ( fabs (cos_theta) > eps_theta ) cos_theta = cos_theta >=
0 ? fabs(eps_theta) : -fabs(eps_theta);
```

```
        return acosf(cos_theta);
    }
```

Kullandığımız Go modelindeki bağsız LJ etkileşim tanımı, iki amino asitin zincir üzerindeki mesafelerinin minimum 4 veya daha fazla olması ve kartezyen koordinatlardaki mesafelerin ise belli mesafe tanımından daha küçük olması gerekmektedir. Bu etkileşim çiftlerinin ve kristal yapıdaki mesafelerinin hafızada tutulması ve simülasyon adımından kullanılması gerekmektedir. Bunun için özel bir data yapısı oluşturulmuş ve bu yapı aşağıdaki gibidir.

```
    struct contact_list {
        int aa1, aa2;
        float r0;
    };
```

Kullandığımız Go modelinde bağsız etkileşme olarak LJ tanımının haricinde hardcore itici etkileşmeleride kullanılmaktadır. LJ etkileşmesinde kullanılan yukarıdaki yapı hardcore itici etkileşmeler içinde kullanılmaktadır. Bağsız etkileşme listeleri aynı yapıda olduğundan, her simülasyon adımında hesaplanacak kuvvetlerin bağımlılıkları bir tutulmuştur. Böylece, birden fazla kuvvet hesabı tek bir bağımlılık formatı ile kontrol edilebilecektir. Bu listeler tek boyutlu tutularak, birden fazla etkileşimin daha hızlı hesaplanması sağlanmıştır. Ayrıca LJ ve hardcore için iki ayrı liste olması, simülasyon sırasında hesaplama kuvvetin türünü ayırt etme işlemini engellemiş, her zaman adımında sadece ilgili kuvvetlerin hesaplanması sağlanarak hız artışı sağlamıştır.

3.3.3. Atomic fonksiyonları

Programlamada en çok kullanılan yapılardan biri, bir değişkenin sahip olduğu değere bir sayı eklenmesi veya çıkartılmasıdır. Bu işlem seri programlamada aşağıdaki gibi bir kodla basit olarak yapılabilir.

```
    a = a + 1
```

Paralel programlamada ise yukarıdaki basit işlemin yapılması bir takım zorluklar barındırmaktadır. Birden fazla iş parçacığı eş zamanlı olarak çalıştığı için, "a" değişkenine birden fazla iş parçacığı erişmek, ve değerini değişmek isteyebilir. İki iş parçacığının aynı a değişkeninin değerini bir arttırmak isterse, a'nın değeri toplamda iki artması gerekirken, iş parçacıklarının a'ya erişim sırasına göre sadece bir arttığı durum

gözlenebilir. Bu durum bilimsel hesaplamalar için kabul edilemez niteliktedir. Bu yüzden paralel programlamanın çıkışından bu yana, bu tip hafızı erişim sorunlarını ortadan kaldırılmak için bir çok çalışmalar yapılmıştır.

CUDA bu tip hataların önüne geçmek için, dahili olarak "atomic" fonksiyonları bulundurur. Bu fonksiyonlarda yapılan işlemlerde, bellekten okunan veri üzerine fonksiyon tarafından sanal bir kilit uygulanır. Böylelikle, aynı bellek alanından okuma veya yazma işlemi başka fonksiyonlar veya iş parçacıkları tarafından yapılamaz. Bu bellek alanında yeni işlem yapılması sadece atomic fonksiyonun yapacağı işlemi bitirip, sanal olarak kitlendiği bellek alanını kullanıma geri açmasının ardından yapılabilir.

```
atomicAdd ( &(Out_Force [id].x) , Energy_Derivative * ij.x );
atomicAdd ( &(Out_Force [id].y) , Energy_Derivative * ij.y );
atomicAdd ( &(Out_Force [id].z) , Energy_Derivative * ij.z );
```

CUDA da ki atomic fonksiyonları sayesinde, bellek okunması sırasında yaşanacak hataların önüne geçmek çok kolay hale gelmiştir. Bu fonksiyonlar, bilimsel hesaplamalar gibi data bütünlüğünün çok önemli olduğu konularda kritik önem taşır. Fakat bu fonksiyonlar, ilgili bellek alanını işlem yapmaya kapattığı için, aynı bellek alanına erişmek isteyen iş parçacıklarının, işlem yapmasını engeller. Böylelikle paralel çalışması gereken iş parçacıkları, bir birinin işlerini bitmesini bekleyerek seri bir forma dönüşür. Bu da programın performansını olumsuz etkilemektedir.

4. BULGULAR

4.1. Paraleleştirme Sonuçları

GPU üzerinde çalışabilecek iş parçacığı sayısının çok fazla olabileceği göz önüne alındığında, basitleştirilmiş zincir temsili yaklaşım içeren protein modelindeki, parçacık sayısı GPU kartını tam yükle çalıştırmak için yeterli gelmemektedir. Ayrıca Cuda'nın tek talimat çok data yapısına uymak için parçacıklara etki eden kuvvetlerin ayrı ayrı paralel hesaplandığı bir algoritma kullanılması, her simülasyon adımında farklı kuvvetlerin hesaplanması serileştirmiş ve paralel programın performansı olumsuz etkilemiştir.

GPU kartlarında, tek kesinlik (single precision) veri tiplerinin, çift kesinlik (double precision) veri tiplerine göre çok daha performanslı çalıştığı için, hesaplamalardaki veri tipleri tek kesinlikli olarak seçilmiştir. Fakat bu tercih beraberinde yuvarlama hatalarında artışa sebep olmuş ve bu hataların minimize edilmesi için farklı tekniklerinin kullanılmasına yol açmıştır. Bunun neticesi olarak her adımda yapılması gereken iş yükü dolaylı olarak artmıştır.

CPU ve GPU'daki tek keskinlik çözünürlüğünün aynı olmaması, iki işlemci arasında hesaplanan değerlerin birbiri ile örtüşmemesini sağlamıştır. Bu yüzden CPU ve GPU'da yapılan işlemlerin birbiri ile aynı çözünürlüğe getirilmesine dikkat edilmiştir. Bu işlem de ekstra hesaplama yükü oluşturulmuştur.

Yapılan basit testlerde, GPU ile çalıştırılan simülasyonların CPU nazaran performanslı çalışmadığı görülmüştür. Mevcut kullanılan hesaplama kaynaklarında GPU kaynaklarının kısıtlılığı göz önüne alınarak, elde edilen bazı simülasyon sonuçları sadece CPU kullanıldığı durumda elde edilmiştir.

4.2. Simülasyon Sonuçları Ve Gözlemler

Bu tez çalışmasında tamamıyla alfa, tamamıyla beta ve karışık yapılı protein ailelerinden toplam 15 protein ele alınmıştır. Bu proteinlerin Protein Databank (PDB) kodları; tamamıyla alfa yapılı protein ailesinden Dihydrolipolylysine-Residue Acetyltransferase (1W4E) [11], Pyruvate Dehydrogenase E2 (1W4J) [11],

Immunoglobulin G binding protein A (1SS1) [12], Mouse C-Myb Dna-Binding Domain Repeat 3 (1IDY) [13] ve Trf2-Interacting Telomeric Rap1 Protein (1FEX) [14]; tamamıyla beta yapılı protein ailesinden Formin Binding Protein (1E0L[15]), Wwprototype (1E0M) [15], Actin Binding Protein (1JO8) [16], 65 kDa Yes-associated protein (1K9Q) [17] ve Twitchin 18th Igsf Module (1WIT) [18]; hem alfa hem de beta yapılı ikincil yapılar içeren karışık yapılı protein ailesinden 50s Ribosomal Protein L9 (1CQU) [19], Procarboxypeptidase A2 (1O6X) [20], Protein G (1PGB) [21], Chymotrypsin Inhibitor 2 (2CI2) [22] ve Protein L (2PTL) [23]'dir. Deneysel çalışmalardan bu proteinlerin iki durum termodinamik ve kinetik davranışı gösterdikleri bilinmektedir. Bu proteinlerden en kısa proteinler 37 amino asit içeren ve tamamıyla beta yapılı protein ailesi üyesi 1E0L ve 1E0M olup, en uzun protein 93 amino asitli ve tamamıyla beta yapılı 1WIT proteinidir.

Bu tez çalışmasında Go modeli yaklaşımı kullanıldığı için öncelikli olarak proteinlerin kristal yapılarındaki kontakların belirlenmesi gerekmektedir. Bunun için zincir üzerinde birbirine minimum dört amino asit mesafesinde olan amino asitlerin tüm ağır atomları (N, O, C, S) arasındaki mesafeler hesaplanmıştır. Bu bağlamda bir amino asit çiftinin katlı yapıda etkileşip etkileşmediğine; o amino asit çiftinin C_α atomları arasında ki mesafe $r_{C_\alpha - C_\alpha} \leq 6.4$ Angstrom'den olması veya diğer mümkün olan atom çiftleri arasındaki mesafelerden en az birisinin 4.5 Angstrom'den küçük olmasına göre karar verilmektedir.

Bu mesafe kriterleri kullanılarak her protein için toplam kontak sayısı $N_{\text{kontak}}^{\text{katlı}}$ hesaplandığında; ait olunan protein ailesine ve katlı yapı tıkızlığına göre $N_{\text{kontak}}^{\text{katlı}}$ değerlerinde yüksek değişkenlik gözlemlenmektedir. Katlı yapıdaki toplam kontak sayısı $N_{\text{kontak}}^{\text{katlı}}$, amino asit sayısı N ile normalize edildiğinde; amino asit başına düşen ortalama bağlantı (kontak sayısı) parametresi $B = N_{\text{kontak}}^{\text{katlı}}/N$ elde edilmektedir.

Aşağıda verilen Tablo 4.1'de dikkatli incelendiğinde; aynı mesafe kriteri kullanılmasına rağmen ortalama bağlantı parametresi değerlerinde çok yüksek değişkenlik gözlemlenmektedir. Özellikle küçük proteinler ve tamamıyla alfa yapılı proteinler için düşük bağlantı değerleri elde edilmiştir. Ortalama bağlantı değeri B'de ki farklılığın sebepleri; kullandığımız basitleştirilmiş model yaklaşımında her amino asidin sadece C_α

atomuyla temsil edilmesi ve su moleküllerinin ihmal edilmesidir. Bununla birlikte katlı yapı tıkızlığı da diğer önemli etkendir.

Tablo 4.1 4. 5 Å Mesafe kriterine göre, farklı proteinlerin topolojik parametreleri ve simülasyon sonuçları

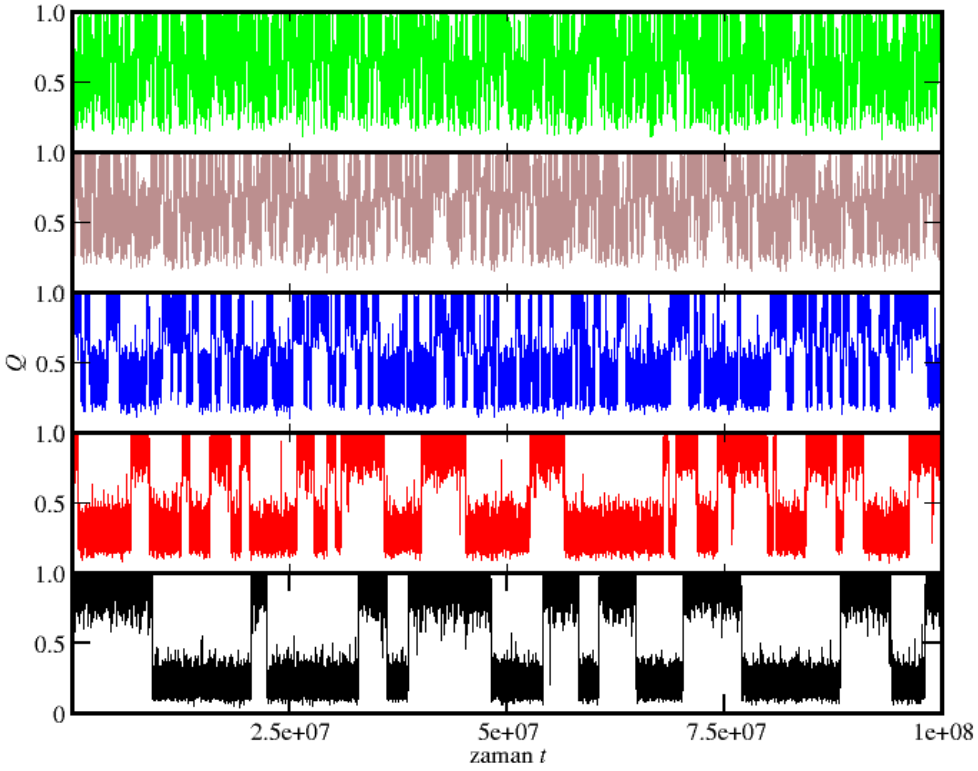
	PDB kodu	N	$N_{\text{kontak}}^{\text{katlı}}$	B	$T_{\max}$	κ_s	$\log_{10} k_f^{\text{sim}}$
sarmal yapılı	1W4E	45	81	1.80	0.883	0.96	-5.52
	1W4J	51	73	1.43	0.810	0.53	-5.24
	1IDY	54	71	1.31	0.747	0.50	-5.24
	1FEX	59	90	1.53	0.750	0.55	-5.32
	1SS1	60	91	1.52	0.789	0.73	-5.41
Beta yapılı	1E0L	37	59	1.59	0.959	0.56	-5.15
	1E0M	37	62	1.68	0.899	0.50	-5.12
	1K9Q	40	64	1.60	0.926	0.54	-5.38
	1JO8	58	149	2.57	1.011	1.00	-7.64
	1WIT	93	252	2.71	0.919	1.00	-7.28
Karışık yapılı	1CQU	39	96	2.46	0.992	1.00	-6.43
	1PGB	56	121	2.16	0.928	1.01	-6.67
	2PTL	62	134	2.16	0.915	1.01	-7.10
	2CI2	64	142	2.22	0.932	1.01	-7.09
	1O6X	81	148	1.83	0.900	1.01	-6.94

Proteinlerin ortalama bağlantı parametresi ile simülasyondan elde edilen geçiş sıcaklığı $T_{\max}$ arasında yüksek korelasyon ($r = 0.95$) gözlemlenmiştir. Burada geçiş sıcaklığı $T_{\max}$ ısı kapasitesinin maksimumuna karşılık gelen sıcaklık değeridir. Diğer taraftan, protein modellerinin termodinamik davranışlarının kooperatif olup olmadığını tayin etmekte kullanılan ve kalorimetrik deneylerde van't Hoff entalpi ile kalorimetrik entalpi oranı olarak tanımlanan termodinamik kooperativite parametresi $\kappa_s = \Delta H_{\text{vH}}/\Delta H_{\text{cal}}$ her protein için ayrı ayrı hesaplanmıştır. [24] Kooperatif davranış gösteren proteinlerde bu oran 1 civarındadır, yani $\kappa_s = \Delta H_{\text{vH}}/\Delta H_{\text{cal}} \approx 1$. Tablo 4.1 'de görüldüğü gibi düşük bağlantı değerlerine sahip proteinler kooperatif olmayan bir davranış sergilemektedir ve bu gözlem deneysel sonuçlarla uyuşmamaktadır. Diğer bir deyişle, düşük bağlantı değerlerine sahip proteinlerin termodinamik davranışlarının daha kooperatif hale getirilmesi gerekmektedir.

Bu nedenle ortalama bağlantı değerlerinin yükseltilmesine karar verilmiştir. Bunun için farklı mesafe kriterleri test edilmiştir. Tanımlanan 5 farklı mesafe kriterinde $r_{C_\alpha - C_\alpha} \leq$ 6.4 Å sürekli aynı olarak alınmış, sadece diğer atomlar arası mesafe tanımı

değiştirilmiştir. Buna göre tanımlanan mesafe kriterleri (MF) sırasıyla 4.5, 5.0, 5.5, 6.0, 6.4 Å'dür.

Aşağıdaki şekilde 1W4J proteininin farklı mesafe kriterlerine (yeşil MF=4.5, kahverengi MF=5.0, mavi MF=5.5, kırmızı MF=6.0 ve siyah MF=6.4) göre geçiş sıcaklıklarında elde edilen yolakları gösterilmiştir.



Sekil 4.1 1W4J Proteini için farklı mesafe kriterlerinde elde edilen simülasyon yolakları

Görüldüğü gibi ortalama bağlanma değeri *B*'nin değeri arttıkça hem geçiş sıcaklıkları artmakta hem de katlanma açılma geçişleri daha kooperatif olmaya başlamaktadırlar. Geçiş bölgesindeki konformasyon yoğunlukları azalmakta, diğer bir deyişle takip edilen yolak sayısı azalmaktadır.

Ele alınan tüm proteinler ve her proteinin farklı mesafe kriterleri için elde edilen geometrik, termodinamik ve kinetik veriler aşağıdaki Tablo 4.2'de verilmiştir.

Tablo 4.2 Farklı Mesafe kriterleri için elde edilen toplojik parametereler ve simülasyon sonuçları

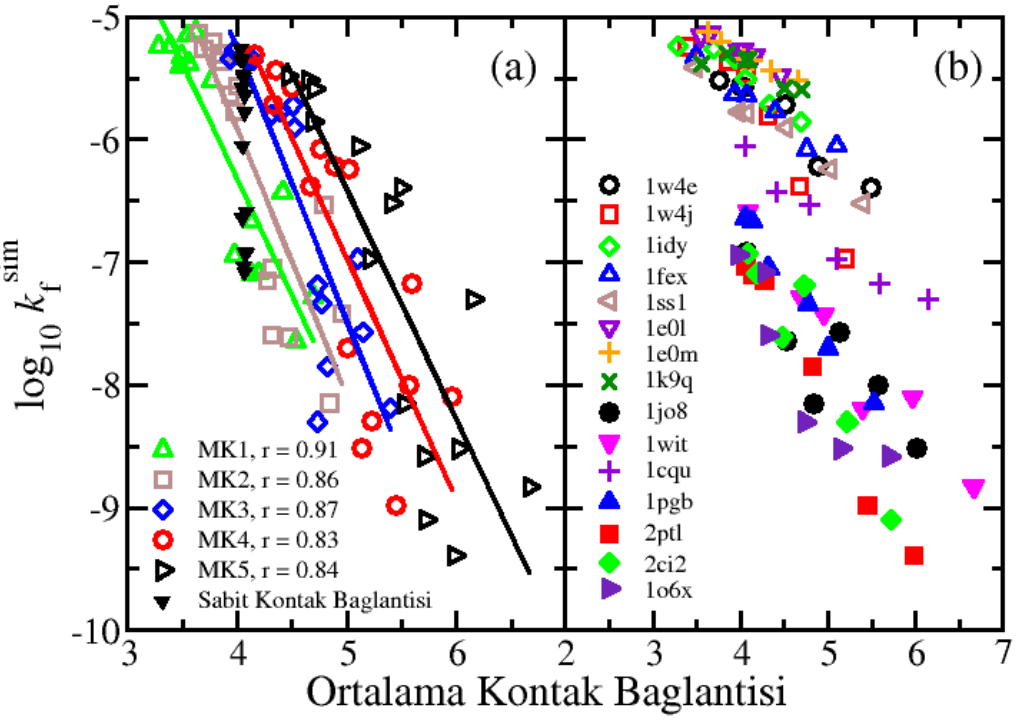
		N	Mesafe	$N_{\text{kontak}}^{\text{katlı}}$	B	EDP	σ_{energy}	$\Delta G^{\ddagger}$	κ_s	$T_{\max}$	$\log_{10} k_{\text{f}}^{\text{sim}}$	$\log_{10} k_{\text{f}}^{\text{exp}}$
Tamamıyla sarmal yapılı	1W4E	45	4.5	81	1.80	12.28	3.39	2.31	0.96	0.883	-5.52	4.44
			5.0	92	2.04	12.21	3.58	2.53	0.97	0.948	-5.56	4.44
			5.0590	94	2.09	12.26	3.59	2.53	0.97	0.961	-5.58	4.44
			5.5	115	2.56	11.79	4.07	2.96	0.98	1.084	-5.72	4.44
			6.0	132	2.93	12.86	4.71	4.47	1.00	1.209	-6.22	4.44
			6.4	159	3.53	13.54	5.65	6.23	1.00	1.402	-6.39	4.44
	1W4J	51	4.5	73	1.43	12.03	2.85	1.78	0.53	0.81	-5.24	5.32
			5.0	97	1.90	11.76	3.50	2.13	0.63	0.935	-5.36	5.32
			5.2711	107	2.10	11.90	3.75	2.40	0.96	0.986	-5.47	5.32
			5.5	120	2.35	12.22	4.00	3.51	0.99	1.061	-5.80	5.32
			6.0	138	2.71	12.92	4.46	5.04	1.00	1.178	-6.38	5.32
			6.4	165	3.24	13.86	5.40	7.05	1.00	1.368	-6.97	5.32
	1IDY	54	4.5	71	1.31	12.41	2.69	1.47	0.50	0.747	-5.24	3.79
			5.0	93	1.72	11.55	3.20	1.34	0.54	0.85	-5.26	3.79
			5.5	107	1.98	12.03	3.55	1.60	0.65	0.929	-5.34	3.79
			5.6742	113	2.09	11.99	3.65	2.16	0.89	0.961	-5.50	3.79
			6.0	128	2.37	12.78	3.99	2.95	1.00	1.068	-5.71	3.79
			6.4	147	2.72	13.16	4.39	3.56	1.00	1.172	-5.85	3.79
	1FEX	59	4.5	90	1.53	13.48	2.61	1.12	0.55	0.75	-5.32	3.56
			5.0	116	1.97	13.79	3.27	2.19	1.00	0.872	-5.63	3.56
			5.1188	124	2.10	13.90	3.40	2.30	0.97	0.906	-5.64	3.56
			5.5	143	2.42	14.29	3.82	2.59	1.00	1.004	-5.77	3.56
			6.0	165	2.80	15.09	4.37	3.42	1.00	1.102	-6.08	3.56
			6.4	185	3.14	15.77	4.84	3.37	1.00	1.216	-6.05	3.56

	1SS1	60	4.5	91	1.52	13.81	2.65	1.37	0.73	0.789	-5.41	4.98
			5.0	121	2.02	14.19	3.30	2.78	1.00	0.917	-5.77	4.98
			5.1188	126	2.10	14.21	3.45	3.02	1.00	0.942	-5.78	4.98
			5.5	153	2.55	14.41	3.95	2.99	0.99	1.082	-5.90	4.98
			6.0	183	3.05	14.73	4.74	4.20	1.00	1.249	-6.24	4.98
			6.4	206	3.43	15.02	5.23	4.91	1.00	1.358	-6.52	4.98
Tamamıyla kırmalı yapı	1E0L	37	4.5	59	1.59	12.53	3.25	0.87	0.56	0.959	-5.15	4.61
			5.0	63	1.70	12.49	3.44	1.39	0.63	1.014	-5.13	4.61
			5.5	75	2.03	13.44	3.66	1.60	0.65	1.053	-5.27	4.61
			5.8184	77	2.08	13.40	3.75	1.69	0.68	1.07	-5.26	4.61
			6.0	82	2.22	13.52	3.96	1.94	0.74	1.113	-5.31	4.61
			6.4	93	2.51	13.87	4.25	2.55	0.78	1.176	-5.48	4.61
	1E0M	37	4.5	62	1.68	11.58	3.18	0.60	0.50	0.899	-5.12	3.84
			5.0	68	1.84	11.41	3.33	0.99	0.61	0.964	-5.20	3.84
			5.2664	77	2.08	11.94	3.69	1.69	0.72	1.042	-5.34	3.84
			5.5	81	2.19	12.26	3.79	1.96	0.74	1.065	-5.35	3.84
			6.0	89	2.41	12.26	4.03	1.97	0.97	1.124	-5.44	3.84
			6.4	100	2.70	12.36	4.60	2.47	0.95	1.194	-5.52	3.84
	1K9Q	40	4.5	64	1.60	14.23	3.15	0.83	0.54	0.926	-5.38	3.63
			5.0	76	1.90	14.46	3.53	1.33	0.65	1.007	-5.30	3.63
			5.4374	84	2.10	14.85	3.81	1.76	0.71	1.048	-5.35	3.63
			5.5	86	2.15	15.13	3.79	1.86	0.70	1.055	-5.35	3.63
			6.0	102	2.55	15.36	4.33	2.53	0.74	1.169	-5.58	3.63
			6.4	110	2.75	15.61	4.60	2.57	0.78	1.216	-5.59	3.63
	1JO8	58	3.8062	122	2.10	21.53	3.35	5.53	1.00	1.011	-6.92	1.07
			4.5	149	2.57	21.50	4.06	7.09	1.00	1.011	-7.64	1.07
			5.0	167	2.88	21.74	4.46	8.23	1.00	1.099	-8.15	1.07

			5.5	184	3.17	21.25	4.82	6.77	1.00	1.171	-7.57	1.07
			6.0	209	3.60	21.25	5.36	8.86	1.00	1.279	-8.00	1.07
			6.4	235	4.05	21.26	5.79	10.35	1.00	1.393	-8.52	1.07
	1WIT	93	3.4406	195	2.10	34.67	3.10	4.96	1.00	0.772	-6.59	0.17
			4.5	252	2.71	33.91	4.09	7.17	1.00	0.919	-7.28	0.17
			5.0	276	2.97	33.78	4.44	7.53	1.00	0.96	-7.42	0.17
			5.5	317	3.41	34.51	4.99	8.93	1.00	1.055	-8.19	0.17
			6.0	370	3.98	34.42	5.59	11.00	1.00	1.188	-8.10	0.17
			6.4	435	4.68	34.66	6.40	13.14	1.01	1.327	-8.83	0.17
Karışık Yapılı ($\alpha + \beta$)	1CQU	39	4.0500	82	2.10	16.06	3.55	4.17	1.00	0.901	-6.05	2.95
			4.5	96	2.46	16.21	3.87	5.05	1.00	0.992	-6.43	2.95
			5.0	111	2.85	15.89	4.31	5.16	1.00	1.096	-6.53	2.95
			5.5	123	3.15	16.07	4.71	6.13	1.01	1.177	-6.97	2.95
			6.0	142	3.64	15.80	5.34	7.21	1.00	1.301	-7.17	2.95
			6.4	164	4.21	15.60	5.91	7.82	1.00	1.446	-7.30	2.95
	1PGB	56	4.3676	117	2.09	20.01	3.52	4.78	1.01	0.905	-6.64	2.61
			4.5	121	2.16	20.31	3.67	4.80	1.01	0.928	-6.67	2.61
			5.0	132	2.36	19.67	3.83	6.02	1.01	0.973	-7.05	2.61
			5.5	157	2.80	18.94	4.28	7.08	1.01	1.072	-7.34	2.61
			6.0	170	3.04	19.55	4.58	7.98	1.02	1.132	-7.70	2.61
			6.4	199	3.55	19.70	5.24	10.10	1.02	1.254	-8.15	2.61
	2PTL	62	4.3875	130	2.10	21.24	3.40	5.59	1.01	0.892	-7.03	1.78
			4.5	134	2.16	21.61	3.53	5.40	1.01	0.915	-7.10	1.78
			5.0	143	2.31	21.27	3.73	5.67	1.01	0.952	-7.15	1.78
			5.5	177	2.85	21.35	4.44	8.41	1.02	1.1	-7.85	1.78
			6.0	216	3.48	21.50	5.25	12.01	1.02	1.271	-8.98	1.78
			6.4	249	4.02	21.70	6.07	13.68	1.02	1.408	-9.39	1.78

	2CI2	64	4.3324	135	2.11	22.40	3.61	5.66	1.01	0.898	-6.93	1.68
			4.5	142	2.22	22.28	3.79	5.58	1.02	0.932	-7.09	1.68
			5.0	160	2.50	22.43	4.02	7.43	1.02	0.996	-7.61	1.68
			5.5	177	2.77	21.81	4.31	6.38	1.02	1.058	-7.19	1.68
			6.0	208	3.25	22.12	4.84	9.47	1.02	1.187	-8.30	1.68
			6.4	240	3.75	22.79	5.50	11.92	1.02	1.329	-9.10	1.68
	1O6X	81	4.5	148	1.83	27.72	3.29	5.22	1.01	0.9	-6.94	2.95
			4.9172	170	2.10	27.82	3.62	6.51	1.02	0.968	-7.39	2.95
			5.0	173	2.14	27.73	3.66	7.06	1.02	0.978	-7.59	2.95
			5.5	204	2.52	26.93	4.12	8.94	1.02	1.088	-8.30	2.95
			6.0	234	2.89	27.01	4.72	9.49	1.02	1.19	-8.52	2.95
			6.4	276	3.41	26.53	5.37	9.60	1.02	1.335	-8.58	2.95

Farklı mesafe kriterlerine göre ortalama bağlanma parametresi B ile katlanma oranlarının logaritmasına göre değişimine bakıldığında (Şekil.4.2-a); tüm mesafe kriterleri için yüksek korelasyonlar elde edilmiştir. Aynı mesafe kriteri için en yavaş ve en hızlı katlanan proteinlerin katlanma oranları farklılığı incelendiğinde; en yüksek farklılık MF=6.4 kriteri için elde edilmektedir. Bu da deneysel sonuçlara en yakın farklılığı veren sonuçtur. Benzer analiz her protein için ayrı ayrı yapıldığında; aynı protein için farklı mesafe kriterlerinde en yüksek katlanma oranı değişkenliği ($10^{2.6}$~350 kat) 2PTL proteini için elde edilmiştir.



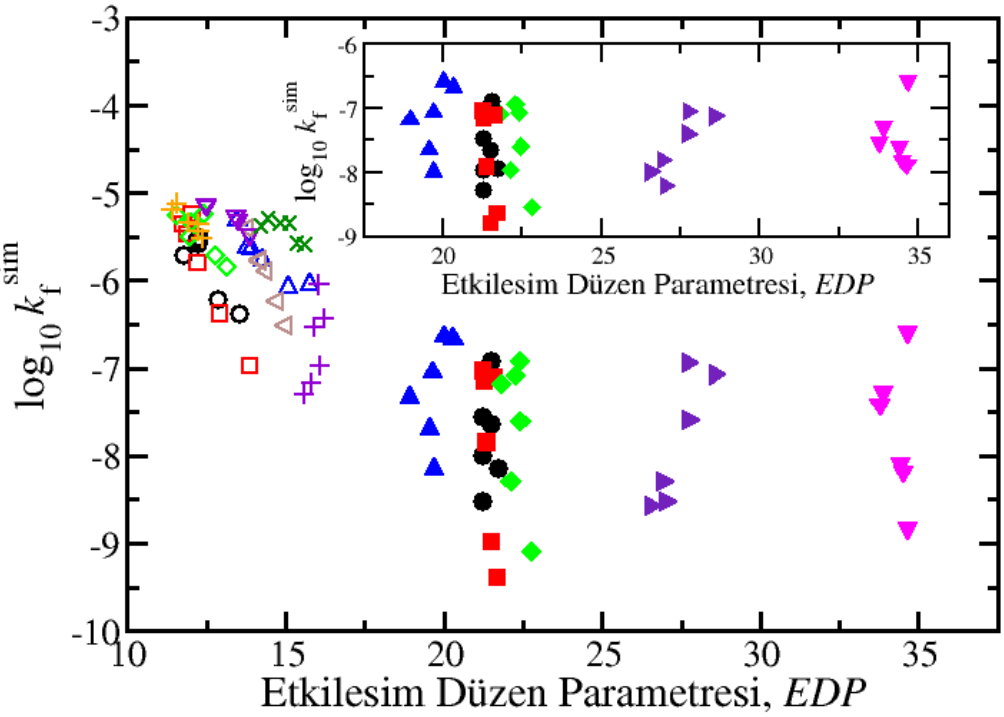
Sekil 4.2 Ortalama kontak bağlantısının katlanma oranları ile olan ilişkisi

Go modeli yaklaşımının yaygın kullanılmasını sağlayan ve katlanmış yapı topolojisini betimleyen etkileşim düzen parametresi EDP ile katlanma oranları arasındaki yüksek korelasyonu bu çalışmada yeniden irdelenmiştir. Etkileşim düzen parametresi EDP şu şekilde tanımlanmaktadır.

$$EDP = \frac{1}{N} \sum_{i,j>i+3} \delta_{ij}$$

Eşitlikte δ_{ij} iki amino asit arasında etkileşme olup olmadığına bağlı olarak 0 veya 1 değeri almaktadır. Etkileşiyorlarsa 1, etkileşmiyorlarsa 0 değeri almaktadır.

Aşağıdaki Şekil 4.3'de 15 proteinin farklı mesafe kriterlerinde hesaplanan EDP değerlerinin katlanma oranlarına göre nasıl değiştikleri gösterilmektedir. Farklı mesafe kriterleri için özellikle 2PTL, 1WIT, 1JO8, 1O6X ve 1PGB proteinleri incelendiğinde; EDP değerlerinde çok küçük değişimler gerçekleşirken, katlanma oranlarında çok yüksek değişimler gözlemlenmektedir.



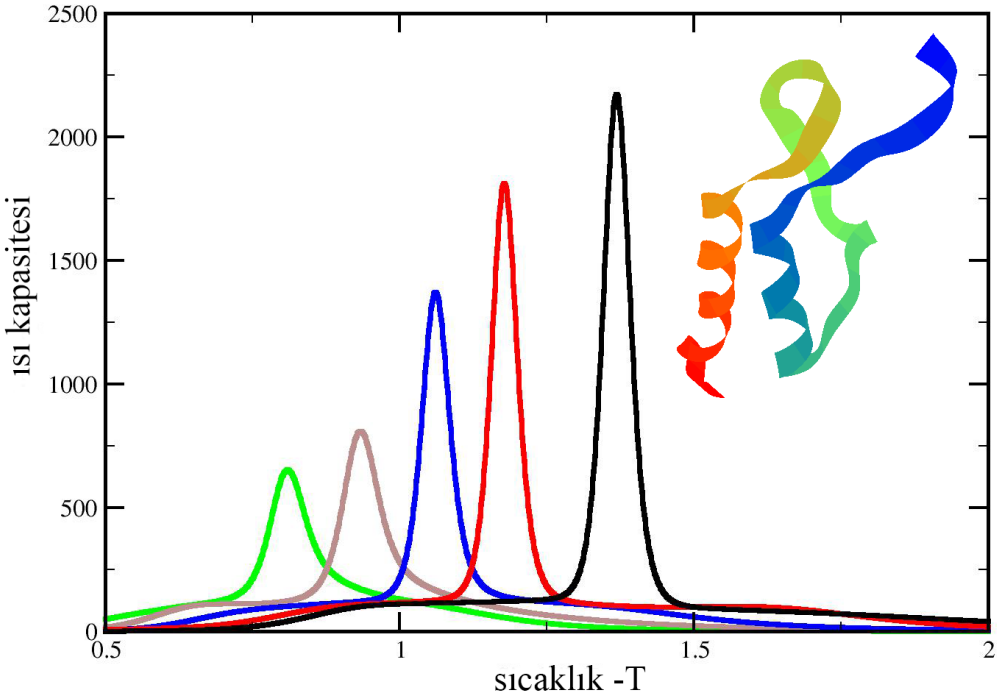
Sekil 4.3 Etkileşim düzen parametresinin, katlanma oranları ile olan ilişkisi

Bu gözlem etkileşim düzen parametresi EDP'nin protein kinetiğini belirlemekte yeterli olmadığını işaret etmektedir. Farklı protein ailelerine mensup proteinlerde, etkileşen amino asitlerin katlanma hızlarını göreceli olarak belirleyebilen EDP, aynı protein içerisinde hasıl olan kinetik değişkenliği yakalayamamaktadır.

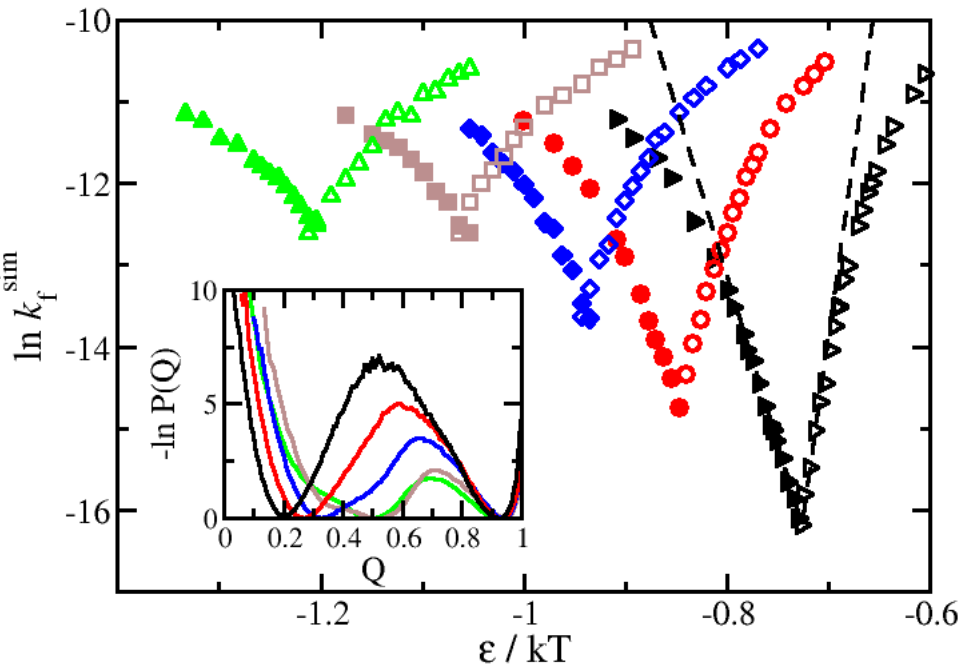
Diğer taraftan, elde edilen sonuçların kontak sayısının arttırılması ve dolayısıyla katlı yapı enerjisi azaltılarak (veya kararlılığı arttırılarak) elde edilen sonuçlar mı olduğu kontrol edilmiştir. Bunun için farklı mesafe kriterlerinde kontak sayısı arttırılsa dahi,

etkileşim büyüklükleri uygun bir şekilde ölçeklenerek katlı yapı enerjisi sabitlenmiştir. Böylelikle kontak sayısı arttırılsa dahi, protein dinamiğine bağsız etkileşmelerden ve açısal (zincir üzerinde yerel etkileşim) enerji terimlerinden gelen katkı oranı, ve katlı yapı enerjisi aynı tutulmuş olmaktadır (Şekil4.3'de ki gömülü grafik). Görüldüğü gibi, termodinamik kararlılığın ve katlanma sürelerinin artması; katlı yapının daha kararlı hale gelmesinden değil aksine protein etkileşim ağındaki bağlantı miktarının artmasından kaynaklanmaktadır.

Diğer taraftan, proteinlerin iki durum kinetiği; sıcaklık ve pH'ın sabit tutulduğu ve denaturant miktarlarının değiştirilmesi ile protein katlanma ve açılma oranlarının değişimin gösteren Chevron grafikleri ile incelenmektedir. İki-durum davranışı gösteren proteinlerin katlanma ve açılma oranlarının logaritması, denaturant değişimi ile lineer bir ilişki sergilemektedir. [9] Aşağıda Şekil. 4.5'de farklı mesafe kriterleri için elde edilen kinetik davranışlar gösterilmektedir. Mesafe kriteri (veya ortalama bağlantı) düşük iken katlanma ve açılma oranlarının lineer oldukları kısım olabildiğince azdır (2-4 k_BT). Bu değer küçük tek bölgeli proteinler için deneylerden 7-10 k_BT olarak gözlemlenmiştir. Mesafe kriteri arttıkça (özellikle MK=6.4 olduğunda) 1W4J proteini deneysel gözlemlerden elde edilen kararlılık ve iki-durum kinetik davranışını sergilemektedir.



Sekil 4.4 Farklı mesafe kriterlerinde, 1W4J'nin ısı kapasitesi



Sekil 4.5 1W4J'nin farklı mesafe kriterlerinde ki Chevron grafikleri

5. TARTIŞMA ve SONUÇ

Bu tez çalışmasında, Langevin dinamiği içeren moleküler dinamik simülasyon tekniği kullanılarak; basitleştirilmiş zincir temsili içeren protein spesifik Go modeli için paralel algoritma ve program geliştirilmiştir. Paraleleştirme platformu olarak CUDA kütüphanesi seçilmiş ve Nvidia ekran kartı kullanılmıştır. Paralelleştirme sonucu istenilen performans artışının sağlanmamasına rağmen, bu tez çalışmasında elde edilen gözlemler karmaşıklık seviyesi daha fazla olan, bütün atomların temsil edildiği ve su moleküllerinin dahil edildiği protein modelleri geliştirilmesinde karşılaşacak problemlerin ön hazırlığı niteliğindedir.

Ayrıca elde edilen simülasyonlar sonucunda; protein katlanması termodinamiği ve kinetiğini anlamakta, modellemekte çok önemli rol oynayan ortalama bağlantı parametresi keşfedilmiştir. Bu parametre şu ana kadar kullanılan bilgisayara dayalı protein modelleri sonuçları ile deneysel sonuçlar arasındaki tutarsızlıkların anlaşılmasında ve giderilmesinde çok önemli rol oynayacaktır. Özellikle yeni etkileşim senaryolarının geliştirilmesi, deney ve simülasyon sıcaklıkları arasındaki uyumsuzluk ilk etapta giderilebilecek problemler arasındadır.

KAYNAKLAR

1. Moore GE, others. Cramming more components onto integrated circuits. McGraw-Hill New York, NY, USA 1965http://web.eng.fiu.edu/npala/eee6397ex/gordon_moore_1965_article.pdf (accessed 10 Dec2014).

2. Zeiser T, Hager G, Wellein G. The world's fastest CPU and SMP node: Some performance results from the NEC SX-9. In: Parallel and Distributed Processing Symposium, International. Los Alamitos, CA, USA: IEEE Computer Society 2009: 1–8.

3. Barnes GH, Brown RM, Kato M, Kuck DJ, Slotnick DL, Stokes RA. The illiac iv computer. Computers, IEEE Transactions on. 1968; 100:746–757.

4. Sterling T, Becker DJ, Savarese D, Dorband JE, Ranawake UA, Packer CV. Beowulf: A Parallel Workstation For Scientific Computation. In: In Proceedings of the 24th International Conference on Parallel Processing. CRC Press 1995: 11–14.

5. Murzin AG, Brenner SE, Hubbard T, Chothia C. SCOP: A structural classification of proteins database for the investigation of sequences and structures. Journal of Molecular Biology. 1995; 247:536–540.

6. Kolodny R, Pereyaslavets L, Samson AO, Levitt M. On the Universe of Protein Folds. Annual Review of Biophysics. 2013; 42:559–582.

7. Kaya H, Chan HS. Contact order dependent protein folding rates: kinetic consequences of a cooperative interplay between favorable nonlocal interactions and local conformational preferences. Proteins: Structure, Function, and Bioinformatics. 2003; 52:524–533.

8. Kaya H, Chan HS. Solvation Effects and Driving Forces for Protein Thermodynamic and Kinetic Cooperativity: How Adequate is Native-centric Topological Modeling? Journal of Molecular Biology. 2003; 326:911–931.

9. Kaya H, Chan HS. Simple two-state protein folding kinetics requires near-levinthal thermodynamic cooperativity. Proteins: Structure, Function, and Bioinformatics. 2003; 52:510–523.

10. Kaya H, Uzunoğlu Z, Chan HS. Spatial ranges of driving forces are a key determinant of protein folding cooperativity and rate diversity. Physical Review E. 2013; 88. doi:10.1103/PhysRevE.88.044701

11. Ferguson N, Sharpe TD, Schartau PJ, Sato S, Allen MD, Johnson CM, Rutherford TJ, Fersht AR. Ultra-fast Barrier-limited Folding in the Peripheral Subunit-binding Domain Family. Journal of Molecular Biology. 2005; 353:427–446.

12. Sato S, Religa TL, Daggett V, Fersht AR. Testing protein-folding simulations by experiment: B domain of protein A. Proceedings of the National Academy of Sciences of the United States of America. 2004; 101:6952–6956.

13. Furukawa K, Oda M, Nakamura H. A small engineered protein lacks structural uniqueness by increasing the side-chain conformational entropy. Proceedings of the National Academy of Sciences of the United States of America. 1996; 93:13583–13588.

14. Hanaoka S, Nagadoi A, Yoshimura S, Aimoto S, Li B, de Lange T, Nishimura Y. NMR structure of the hrap1 myb motif reveals a canonical three-helix bundle lacking the positive surface charge typical of myb DNA-binding domains. Journal of Molecular Biology. 2001; 312:167–175.

15. Macias MJ, Gervais V, Civera C, Oschkinat H. Structural analysis of WW domains and design of a WW prototype. Nature Structural & Molecular Biology. 2000; 7:375–379.

16. Fazi B, Cope MJTV, Douangamath A, Ferracuti S, Schirwitz K, Zucconi A, Drubin DG, Wilmanns M, Cesareni G, Castagnoli L. Unusual Binding Properties of the SH3 Domain of the Yeast Actin-binding Protein Abp1 STRUCTURAL AND FUNCTIONAL ANALYSIS. Journal of Biological Chemistry. 2002; 277:5290–5298.

17. Pires JR, Taha-Nejad F, Toepert F, Ast T, Hoffmüller U, Schneider-Mergener J, Kühne R, Macias MJ, Oschkinat H. Solution structures of the YAP65 WW domain and the variant L30 K in complex with the peptides GTPPPPYTVG, N-(n-octyl)-GPPPY and PLPPY and the application of peptide libraries reveal a minimal binding epitope. Journal of Molecular Biology. 2001; 314:1147–1156.

18. Fong S, Hamill SJ, Proctor M, Freund SMV, Benian GM, Chothia C, Bycroft M, Clarke J. Structure and Stability of an Immunoglobulin Superfamily Domain from Twitchin, a Muscle Protein of the NematodeCaenorhabditis elegans. Journal of Molecular Biology. 1996; 264:624–639.

19. Luisi DL, Kuhlman B, Sideras K, Evans PA, Raleigh DP. Effects of varying the local propensity to form secondary structure on the stability and folding kinetics of a rapid folding mixed α/β protein: characterization of a truncation mutant of the N-terminal domain of the ribosomal protein L9. Journal of Molecular Biology. 1999; 289:167–174.

20. Jiménez MA, Villegas V, Santoro J, Serrano L, Vendrell J, Avilés FX, Rico M. NMR solution structure of the activation domain of human procarboxypeptidase A2. Protein Science. 2003; 12:296–305.

21. Gallagher T, Alexander P, Bryan P, Gilliland GL. Two crystal structures of the B1 immunoglobulin-binding domain of streptococcal protein G and comparison with NMR. Biochemistry. 1994; 33:4721–4729.

22. McPhalen CA, James MN. Crystal and molecular structure of the serine proteinase inhibitor CI-2 from barley seeds. Biochemistry. 1987; 26:261–269.

23. Wikström M, Drakenberg T, Forsén S, Sjöbring U, Björck L. Three-dimensional solution structure of an immunoglobulin light chain-binding domain of protein L. Comparison with the IgG-binding domains of protein G. Biochemistry. 1994; 33:14011–14017.

24. Chan HS, Shimizu S, Kaya H. Cooperativity principles in protein folding. Methods in enzymology. 2004; 380:350–379.

ÖZGEÇMİŞ

1988 yılının aralık ayında doğan Gökhan Selamet, ilkokul ve liseyi Darüşşafaka Eğitim Kurumlarında tamamladı. Gaziantep Üniversitesi Fizik mühendisliği bölümünden mezun oldu. Şu anda biyoenformatik alanında protein katlanması üzerine çalışmalar yapmaktadır.