

T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ
ENSTİTÜSÜ

ÇOK ÇEKİRDEKLİ İŞLEMCİLER İÇİN
MOLEKÜLER DİNAMİK SİMÜLASYONU

Umut DEMİRTAŞ
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI

GEBZE
2011

T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ
ENSTİTÜSÜ

ÇOK ÇEKİRDEKLİ İŞLEMCİLER İÇİN
MOLEKÜLER DİNAMİK SİMÜLASYONU

Umut DEMİRTAŞ
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI
TEZ DANIŞMANI
Yrd. Doç. Dr. Fatih Erdoğan SEVİLGİN

GEBZE
2011

ÖZET

TEZİN BAŞLIĞI: ÇOK ÇEKİRDEKLİ İŞLEMCİLER İÇİN MOLEKÜLER DİNAMİK SİMÜLASYONU

YAZAR ADI : Umut Demirtaş

Olay Temelli Moleküler Dinamik simülasyonunun paralelleştirilmesi problemi son yıllarda gelişen çok çekirdekli işlemcilerin sağladığı avantajları kullanma adına önemli bir problem haline gelmiştir. Önerilen çok az sayıdaki algoritmanın yetersiz kaldığı görülmüş ve bu konudaki eksiğin giderilmesi adına asenkronize, kilit mekanizmalarından mümkün olduğu kadar arındırılmış yeni bir algoritma tasarlanmıştır.

Tasarlanan algoritma, yönetici-işçi (Master-Worker) modeli ile üretici-tüketici (Producer-Consumer) modelinden yola çıkılarak asenkronize ve kilitsiz (lock-free) veri yapıları yardımıyla gerçekleştirilmiştir. Yapılan çalışmalar neticesinde deneylerde şuana kadar literatürdeki en iyi performans elde edilmiş, gelecek çalışmalar adına umut verici sonuçlar ortaya çıkmıştır.

SUMMARY

THESIS TITLE: PARALLEL EDMD SIMULATION FOR MULTI-CORE ARCHITECTURE

AUTHOR : UMUT DEMİRTAŞ

In recent years, in the behalf of using the advantages of the improved multi-core processors collimate the Event-Based Molecular Dynamics simulation problem has become an important problem.

A few algorithms have been proposed. The algorithms are examined and it shown that all algorithms are insufficient in many cases. Therefore a new asynchronies lock-free parallel algorithm is proposed and the simulation performance has been improved as much as possible.

The designed algorithm is implemented on the basis of Master-Worker model and Producer-Consumer model with the aid of asynchronous and lock-free data structures. As a result of the studies, the best performance of the literature is obtained in the experiments and the obtained results are promising for the future studies.

TEŞEKKÜR

Benden desteğini esirgemeyen ailem Salim Demirtaş, Fatma Demirtaş, Oğuz Demirtaş, Işıl Geçim ve Yonca Erdem'e, hocam Fatih Erdoğan Sevilgen'e ve arkadaşlarıma teşekkür ederim.

İÇİNDEKİLER DİZİNİ

	<u>Sayfa</u>
ÖZET	iv
SUMMARY	v
TEŞEKKÜR	vi
İÇİNDEKİLER DİZİNİ	vii
SİMGELER VE KISALTMALAR DİZİNİ	ix
ŞEKİLLER DİZİNİ	x
TABLolar DİZİNİ	12
1 GİRİŞ	13
1.1 MD Simülasyonu	13
1.2 MD Simülasyon Teknikleri	13
1.2.1 Zaman Adımlı Yaklaşım	14
1.2.2 Olay Tabanlı Yaklaşım	14
1.3 Paralel EDMD	15
1.4 Tezin Literatüre Katkısı	16
2 OLAY TABANLI MD SİMÜLASYONU	17
2.1 Genel Yapısı	17
2.2 Hücre Bölümlendirme	18
2.3 Molekül Bazlı Lokal Zaman Kullanımı	21
2.4 Öncelik Kuyruğu Kullanımı	22
2.5 Kullanılan Seri Algoritma Özellikleri	23
3 PARALEL EDMD ÇALIŞMALARI	25
3.1 Ortam Bölümlendirme	26
3.2 Olay Bazlı Bölümlendirme	29
3.2.1 Sebep Sonuç Problemi (Causality)	32

3.2.2	Uyumluluk, Tutarlılık Problemi (Coherence)	32
3.2.3	Tutarlılık ve sebep sonuç problemlerinin birbirlerini tetiklemesi (Both Causality and Coherence)	32
3.3	Varsayımsal Olay İşleme	33
4	EDMD İÇİN ASENKRON PARALEL ALGORİTMA	35
4.1	Ara Bellek Özellikleri	40
4.2	Koşut Zamanlı Kuyruk Yapısı Uygulaması	41
5	PROGRAMLAMA DETAYLARI	45
5.1	Sınıf Diyagramı	45
5.2	İklendirme	54
6	DENEYLER	56
7	SONUÇLAR	59
	KAYNAKLAR DİZİNİ	60

SİMGELER VE KISALTMALAR DİZİNİ

EDMD: Event Driven Molecular Dynamics (Olay Tabanlı Moleküler Dinamik)

MD: Moleküler Dinamik

ŞEKİLLER DİZİNİ

<u>Sekil</u>	<u>Sayfa</u>
2.1 EDMD'nin temel işleyiş süreci.	17
2.2 EDMD Simülasyonu İçin Temel Algoritma.	18
2.3 Molekülün Çevresinin Hücre Yapısıyla Tanımlanması.	19
2.4 Hücre Büyüklüğü İle Molekül Çapı Arasındaki İlişki.	20
2.5 EDMD Simülasyonunda Kullanılan Seri Algoritma.	24
3.1 2 boyutlu örnek parçalama.	27
3.2 Ortam Bölümlendirme Yöntemi ile Paralleleştirme.	28
3.3 Olay Bazlı Bölümlendirme Yöntemi ile Paralleleştirme.	29
3.4 Tek bir olayın işlenme döngüsü.	30
3.5 Paralel olay işleme döngüsü	31
3.6 Varsayımsal Olay İşleme Algoritmasına Bir Örnek	34
4.1 Geliştirilen Yeni Algoritma İçin Şema	36
4.2 İşçiler için Algoritma	37
4.3 Yönetici için Algoritma	39
4.4 Koşut Erişime Uygun Kuyruk Uygulaması	43
4.5 Kutularda olabilececek Olası Bir Durum	44
5.1 Sınıf Diyagramı	46
5.2 Molekül Sınıfı	47
5.3 Vektör Yapısı	48
5.4 Olay Sınıfı	48
5.5 Hücre Sınıfı	49
5.6 Zaman Ölçümleri İçin Timetest Sınıfı	49
5.7 Genel Değişkenler İçin Context Sınıfı	50
5.8 Kuyruk Yapısı	51
5.9 Duvar Sınıfı	51
5.10 Ara Bellek İçin Geliştirilen Sınıf Yapısı	52
5.11 Simülasyon Sınıfı	53
6.1 Arabellek Boyutu 4 Olduğu Zaman Hızlanma	58
6.2 Arabellek Boyutu 8 Olduğu Zaman Hızlanma	58

6.3 Arabellek Boyutu 12 Olduđu Zaman Hızlanma	58
6.4 Arabellek Boyutu 24 Olduđu Zaman Hızlanma	58

TABLÖLAR DİZİNİ

Tablo

Sayfa

Tablo 6.1 Arabellek Boyutu 4 Olduđu Zaman Çalışma Zamanı	57
Tablo 6.2 Arabellek Boyutu 8 Olduđu Zaman Çalışma Zamanı	57
Tablo 6.3 Arabellek Boyutu 12 Olduđu Zaman Çalışma Zamanı	57
Tablo 6.4 Arabellek Boyutu 24 Olduđu Zaman Çalışma Zamanı	57

1 GİRİŞ

Bilgisayar bilimlerinin son yüzyılda hızlı bir şekilde ilerlemesi bilimsel hesaplamaların bilgisayarlar tarafından daha hızlı ve yüksek doğrulukla yapılmasını sağlamıştır. Bir çok bilimsel çalışmada teorik ve deneysel hesaplamaların dışında simülasyon çalışmalarına da ihtiyaç duyulmaktadır (Rapaport, The art of molecular dynamics simulation 2004) (Pöschel and Schwager 2005). Bu simülasyonlarda, belirli özellikteki ortamların bilgisayarlar tarafından betimlenmesi amaçlanmaktadır. Moleküler Dinamik simülasyonu bu anlamda birçok bilimsel çalışmada etkin bir biçimde kullanılan bir simülasyondur.

Günümüzde işlemci mimarisi fiziksel anlamda karşılaştığı sınırlardan dolayı çok çekirdekli yapılara yönelmiştir. Çok çekirdekli işlemcilerde çekirdek sayısı ve çekirdeklerin kapasitesi arttıkça, birçok uygulamada olduğu gibi MD simülasyonlarında da paralelleştirme çalışmalarına ihtiyaç duyulmuştur.

Bu tez çalışmasında, belirlenen ihtiyaca yönelik olarak, literatürdeki algoritmalara alternatif bir paralel algoritma tasarlanmıştır. Tasarlanan algoritmanın daha önce önerilenlerden farkı, performansı ve literatüre katkısı incelenmiştir.

1.1 MD Simülasyonu

Moleküler Dinamik simülasyonu atomik boyuttaki etkileşimleri bilgisayar yardımıyla incelemektedir. Belirli bir ortamda bulunan atomların, parçacıkların ya da moleküllerin birbirleriyle etkileşimi bilgisayar yardımıyla betimlenerek gerçek hayatta olan olaylar hakkında bilgi sahibi olmak istenmektedir.

1.2 MD Simülasyon Teknikleri

Gerçek hayatta moleküller arası çekim kuvvetleri olduğu kabul edilmiştir. Bu kuvvetler moleküllerin hareketlerini lineer olmayan bir doğrultuda yapması anlamına gelmektedir. Molekülleri hareket ettirme prensibine göre işleyen MD simülasyonlarının birçoğu bu kuvvetleri görmezden gelmektedir. Bu kuvvetlerin yok

sayılması, olası olayları kolay hesaplanabilir kılmaktadır. Hesaplanabilen bu olaylar moleküllerin zaman ile ilerlemesi ya da çarpışmaları olabilir.

Molekülleri hareket ettirmeye dayanan iki farklı simülasyon yaklaşımı ortaya atılmıştır. Bunlar, “zaman adımlı” ya da “olay tabanlı” yaklaşımlardır.

1.2.1 Zaman Adımlı Yaklaşım

İlk yaklaşım, belirli bir zaman aralığında molekülleri hareket ettirme prensibine dayanmaktadır. Bu zaman aralığı içerisinde eğer moleküller arası çarpışmalar varsa bunlar gerçekleştirilir ve bu şekilde simülasyonda ilerleme sağlanmaktadır. Zaman adımının seçimi performansı belirleyen en önemli faktördür. Ayrıca zaman adımı seçimi simülasyonun tutarlılığını belirler. Belirli bir değerden büyük seçilmesi durumunda çarpışma kaçırılması çok büyük olasılıktır. Bu nedenle oldukça küçük seçilmelidir. Ancak, küçük seçilmesi de performansı oldukça düşürmektedir.

1.2.2 Olay Tabanlı Yaklaşım

Kolayca anlaşılabilceği gibi yeterince küçük seçilen bir zaman adımında moleküller birçok kez çarpışma olmadığı halde güncellenmektedir. Bilindiği gibi kuvvetler ihmal edildiğinde, bir molekül bir çarpışmaya karışmadığı sürece aynı doğrultuda gitmektedir. Eğer sistemde gerçekleşecek ilk olay bilinmiş olsaydı bütün moleküller direk bu zamana ilerletilebilirlerdi ve simülasyonda ilerleme bu şekilde sağlanabilirdi. Bu şekilde yapılan simülasyon zaman adımlı değil de olay tabanlı olarak düşünülebilir.

Olay tabanlı simülasyonda, moleküller sistemdeki olası çarpışmalardan ilkinin zamanına ilerletilir ve bu çarpışma gerçekleştirilir. Bu şekilde simülasyonda ilerleme sağlanmış olur. Bir sonraki adımda olası çarpışmaları hesaplarken önceki adımda hesaplanmış olan olaylardan faydalanılabilir, çünkü birçok olay hala geçerliliğini korumaktadır. Sadece, sistemde gerçekleşen ilk çarpışma sistemdeki olası bazı çarpışmaları ortadan kaldırabilir ve yenilerini ekleyebilir.

Olası çarpışmaların zamana göre sıralanması, geçerliliklerini kaybedenlerin çıkarılması ve yerine yenilerinin eklenmesi bir öncelik kuyruğu yapısı ile gerçekleştirilebilir. Literatürde birkaç farklı öncelik kuyruğu algoritması önerilmiş ve bu algoritmaların sahip oldukları farklı özellikler kullanım açısından farklılıklar getirmiştir.

1.3 Paralel EDMD

Olay tabanlı MD simülasyonları, alan ve molekül sayısının büyüklüğü, hesaplama karmaşıklığı ve gözlemlenmek istenen olayların oluşması için gereken simülasyon süresi gibi bir çok etkenden dolayı çok uzun zaman almaktadır. Bir maddenin belirli bir ortamdaki bir tek etkileşimini incelemek bile günler hatta haftalar sürebilmektedir. Bu nedenle simülasyonu koşturarak işlemek gelişen süper bilgisayarlar göz önüne alındığında elzem hale gelmiştir.

EDMD için şuana kadar önerilen paralel algoritmalar incelendiğinde; alan ayrıştırma (domain decomposition) ve varsayımsal olay işleme (speculative execution) teknikleri kullanılarak koştur işlem sağlanmaya çalışılmıştır. Alan ayrıştırma yöntemi (Miller and Luding 2003) çalışmada gerçekleştirilmiştir. Varsayımsal işleme yönetimi ile olay bazlı ayrıştırma (Event based decomposition), ve yönetici-işçi (Master-Worker strategy) teknikleri kullanılarak 3 farklı çalışmada koştur işlenir hale getirilmek istendiği görülmüştür. Alan ayrıştırma yönteminde simülasyon ortamı işlemcilerle olabildiğince eşit yük dağılımı hedeflenerek bölünmüş ve koştur işleme sağlanmıştır. Olay bazlı ayrıştırma, varsayımsal olarak olası olayları ele alarak, her bir olayı farklı bir iş parçacığına (thread) paylaştırılmak suretiyle simülasyonu koştur işlemeyi sağlamaktadır (Herbordt, Khan and Dean 2009). Ayrıca Varsayımsal işleme yöntemi, yönetici-işçi mekanizması üzerine kurularak koştur işlem sağlanmış ve hızlanma bu şekilde elde edilmeye çalışılmıştır (Su, Jiang ve Zhang 2007). Son olarak yapılan başka bir çalışmada hem olay bazlı ayrıştırma, hemde yönetici-işçi yöntemleri birlikte kullanılarak koştur işlem sağlanmıştır (Khan ve Herbordt 2011). Her 3 yöntemde de belirli miktarlarda hızlanma elde edilmekte fakat hızlanmanın çoğu zaman yetersiz kaldığı veya sürdürülebilir olmadığı görülmektedir.

EDMD’de olaylar arasında yakın bir ilişki vardır. Bu ilişki olayları bağımsız bir şekilde düşünmemizi engeller. Hem zaman hem de konum açısından var olan bu ilişki, bilinen paralelleştirme metotlarının kolayca uygulanmasını engeller. Bu sebeple, literatürde EDMD için önerilen çok az sayıda paralelleştirme çalışması vardır.

1.4 Tezin Literatüre Katkısı

Bu çalışmada literatürdeki paralel algoritmalara alternatif bir paralel algoritma tasarlanmış ve bu yeni algoritmanın verimliliği hakkında deneyler yapılmıştır. Literatürdeki algoritmaların karşılaştığı problemler ve eksik kaldığı noktalar incelenmiş, yeni tasarlanan algoritma ile bu problemlerin etkin biçimde minimuma indirilmesi sağlanmıştır. Kısaca, tasarlanan algoritma literatürdeki algoritmalarda karşılaşılan kilit mekanizmalarını azaltmış, yöneticinin iş yükünü azaltmış ve ayrıca ara bellek kullanımı ile tutarlılık problemlerinin çözümünü kolaylaştırmıştır. Yapılan deneylerde belirli miktarlarda hızlanma elde edilmiş ve algoritmanın etkinliği ortaya konulmuştur.

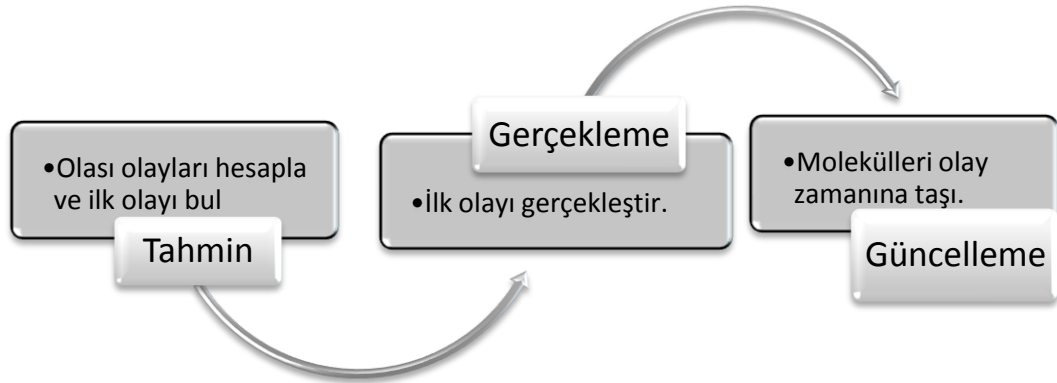
Tezin ilerleyen bölümlerinde sırasıyla, önce Olay tabanlı Moleküler Dinamik ile ilgili bilgiler verilecek ve koşut işlem öncesi seri algoritmanın özellikleri incelenecektir. Daha sonra koşut işlem algoritmaları incelenecek ve değerlendirilecektir. Ardından problem için önerilen yeni algoritma tanıtılacak ve bu algoritmanın özellikleri irdelenecektir. Deney sonuçları ve yapılan karşılaştırmalardan sonra değerlendirmeler yapılacak ve gelecek adına yapılabilecek çalışmalardan bahsedilecektir.

2 OLAY TABANLI MD SİMÜLASYONU

MD simülasyonlarında betimlenen ortam, moleküller ve bir çok etken simülasyon performansını etkilemektedir. Betimlenen sistemlerde moleküller tek atomlu ya da çok atomlu olabilmektedir. Sıcaklık basınç gibi birçok faktör moleküllerin yapılarını ve davranışlarını etkilemektedir. Hatta kimyasal reaksiyonlar bile gerçekleşebilir ve molekül sayıları ve yapıları değişebilir.

2.1 Genel Yapısı

MD simülasyonlarında zaman adimli yaklaşıma nazaran olay tabanlı yaklaşım büyük bir yenilik getirmiştir. Simülasyon çalışma süreleri zaman adimli yaklaşıma göre oldukça kısaltılmıştır (Pöschel and Schwager 2005). Olay tabanlı yaklaşımın temel işleyişi Şekil 2.1 EDMD'nin temel işleyiş süreci.'de anlatılabilir.



Şekil 2.1 EDMD'nin temel işleyiş süreci.

Simülasyonda ilerleme 3 ana işlem ile sağlanır. Bunlar tahmin, gerçekleştirme ve güncelleme işlemleridir. Tahmin aşamasında olası bütün olaylar hesaplanır (ilerleyen bölümlerde, olası olayların hepsinin hesaplanmasına gerek olmadığı gösterilecektir.) ve bu olaylardan en önce olacak olan olay seçilir. Gerçekleme aşamasında seçilen olayın molekülleri olay zamanına güncellenir ve moleküllerin yeni doğrultuları hesaplanır. Bu şekilde olay gerçekleştirilmiş olur. Son olarak güncelleme aşamasında bütün moleküller olay zamanına ilerletilir. Bu şekilde simülasyonda bir adım işlenmiş olur. Simülasyon periyodu tamamlanıncaya kadar bu adımlar tekrarlanır.

Algoritma karmaşıklığını incelemek için temel bir algoritma verecek olursak Şekil 2.2 EDMD Simülasyonu İçin Temel Algoritma. algoritma anlamlı olacaktır.

Algoritma 1 EDMD Simülasyonu için temel algoritma
Simülasyon periyodu süresince aşağıdakileri adımları tekrarla.
Olası olayları hesapla ve ilk olayı bul.
İlk olayı gerçekleştir.
Molekülleri olay zamanına taşı.

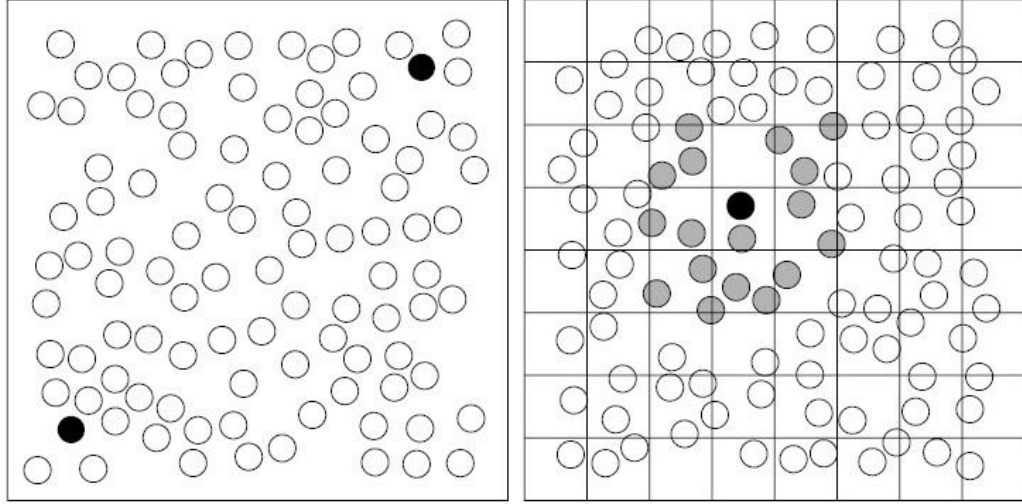
Şekil 2.2 EDMD Simülasyonu İçin Temel Algoritma.

Verilen en temel algoritmanın problem karmaşıklığı hakkında bir şeyler söylemek gerekirse; öncelikle ortamdaki ilk olayı hesaplamak için bütün moleküllerin olası çarpışmalarından ilki bulunmalıdır. Bunun anlamı, bir molekül için $N - 1$ molekülün kontrol edilmesi demektir. N molekül düşünüldüğünde bu olay $O(N^2)$ zaman almaktadır. Bir olayı gerçekleştirmek, çarpışan moleküllerin hızları ve pozisyonları esas alındığında momentumun korunduğu düşünülürse $O(1)$ zamanda bir iki hesaplama ile çok kolay yapılabilir. Molekülleri olay zamanına taşımak bütün moleküller için konum ve zaman bilgilerini güncellemek demektir ve bu iş $O(N)$ zaman karmaşıklığına sahiptir. Görüldüğü üzere verilen en temel algoritma $O(N^2)$ zaman karmaşıklığındadır. Bu algoritmanın üzerinden birçok iyileştirme yapılmış ve karmaşıklık olabildiğince azaltılmıştır.

2.2 Hücre Bölümlendirme

EDMD için verilen en temel algoritmanın zaman karmaşıklığı düşünüldüğünde ortamdaki ilk olayı hesaplama belirleyici olan bölümü oluşturmaktadır. Çünkü N molekül için $O(N^2)$ işlem gerekmektedir. Bu gereklilik bir molekülün olası çarpışma partneri olarak bütün moleküllerin hesaba katılması gereksiniminden kaynaklanmaktadır. Aslında, kolayca düşünülebilir ki ortamda birbirlerinden uzak olan iki molekülün çarpışma olasılığı yok denecek kadar azdır. Bir molekül eğer bir çarpışma yapacaksa yakın çevresinde bulunan moleküllerle çarpışma yapmalıdır. Bu durumda her molekül için çevre tanımı yapılmalı ve tanımlanan çevre her adımda güncellenmelidir.

Literatürde çevre tanımı için kullanılan 2 farklı yaklaşım vardır. Bunlardan ilki komşu listesi ya da komşuluk listesi yöntemidir. Bu yöntemde her molekül yakın çevresindeki moleküllerin listesini tutar ve günceller.

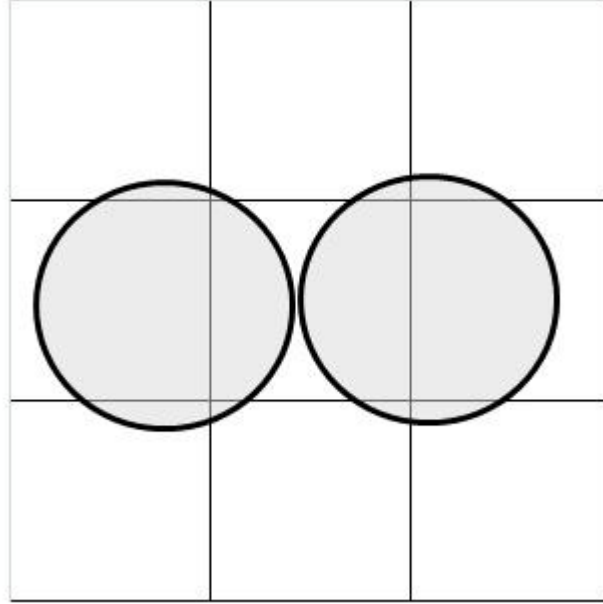


Şekil 2.3 Molekülün Çevresinin Hücre Yapısıyla Tanımlanması.

Diğer bir yaklaşım ise ortamı hücelere bölmektir (Şekil 2.3 Molekülün Çevresinin Hücre Yapısıyla Tanımlanması.). Bu yaklaşımda ortam alt bölgelere bölünür ve bu alt bölgelere hücre adı verilir. Hücreler 2 boyutta kare, 3 boyutta ise küpler olarak düşünülebilir. İki hücre eğer en az bir ortak noktaya sahipse komşu hücreler olarak adlandırılırlar. Bir hücrenin 2 boyutta 8 komşusu, 3 boyutta ise 26 komşusu bulunur. Bir hücrede bulunan bir molekül sadece bu hücredeki ve komşu hücrelerdeki moleküllerle çarpışabilir; daha uzakta bulunan bir hücrenin molekülleriyle çarpışamaz (bunun için hücre boyutu molekül çapıyla orantılı seçilir). Bunun için öncelikle bulunduğu hücreden çıkması ve en azından o hücrenin bir komşusuna geçmesi gerekir. Hücre yaklaşımı hücre geçişi olarak tanımlayabileceğimiz yeni bir olay tanımlamamızı gerektirir. Başka bir deyişle, molekülün bir hücreden başka bir hücreye geçişi de olay olarak tanımlanmaktadır.

Bir molekül bulunduğu hücreyi bilir. Bir hücre ise komşu hücrelerini ve varsa duvarlarını bilir. Bu sayede molekül olası çarpışmalarını sadece bulunduğu hücrede ve komşu hücrelerde arar. Bunun sonucunda bir molekülün olası çarpışmalarını

aramak için $N - 1$ moleküle de bakma zorunluluğu ortadan kalkar. Sadece çevresinde bulunan molekül sayısı kadar moleküle bakmak gereklidir.



Şekil 2.4 Hücre Büyüklüğü İle Molekül Çapı Arasındaki İlişki.

Bir hücrede bulunan molekül sayısı hücre büyüklüğü ile doğru orantılıdır. Hücre büyüklüğü bir molekülün çapından küçük olamaz. Aksi halde, bir molekül komşuluğunda bulunmayan bir molekül ile çarpışabilir. Örneğin Şekil 2.4 Hücre Büyüklüğü İle Molekül Çapı Arasındaki İlişki.'te merkezleri komşu olmayan iki farklı hücrede bulunan molekül iki molekül, eğer hücre büyüklüğü molekül çapından küçük seçilirse çarpışma partneri olabilirler. Hücrelerin çok büyük seçimi ise molekülün daha çok çarpışma partnerine sahip olabilmesi anlamına gelir. Küçük seçimi ise daha çok fazla hücre geçiş olayı gerçekleşmesine sebep olmaktadır. En uygun hücre boyutu seçimi molekül yoğunluğuna bağlı olarak değişmektedir.

Yapılan çalışmalarda, esas alınan unsur hücre sayısını belirlemektir. Çünkü hücre sayısı, ortalama olarak bir hücreye kaç molekül düşeceğini belirlemektedir. Bu durum doğrudan problem karmaşıklığını belirleyecek husustur. Bu nedenle hücre boyutunu belirlemek hücre sayısını belirlemek adına önemlidir. Hücre boyutlarından yola çıkarak bir iki hesaplama yardımıyla hücre sayısı hesaplanabilir.

Molekül Sayısının belirli bir oranında hücre sayısı belirlemek, yoğunluğun eşit dağıldığı bir sistemde molekülün olası çarpışma partnerlerinin sayısını sabit tutacaktır. Bir Molekül için $O(1)$ olan hesaplama süresi N molekül için $O(N)$ olacaktır. Bu şekilde $O(N^2)$ olan tahmin evresi $O(N)$ zamana indirilmektedir. Aynı şekilde, problem karmaşıklığı da $O(N)$ 'e düşürülmüş olur.

2.3 Molekül Bazlı Lokal Zaman Kullanımı

EDMD simülasyonu üzerinde yapılabilecek bir diğer iyileştirme ise her bir molekül için yerel zaman kullanılmasıdır. Eğer moleküllere yerel zaman ataması yapılırsa, işlenen bir olayın ardından molekülü olay zamanına güncellemek zorunda kalınmaz. Başka bir deyişle, olay bazlı simülasyon yapıldığını düşünürsek, molekülün son olay zamanına kadar hiçbir olaya karışmamaktadır. Bu durumda da güncellenmek zorunda değildir.

En son işlenen olay zamanına “*Global Zaman*” ya da “*Simülasyon Zamanı*” denilir. Bir molekülün global zamandan ne kadar öncede bulunduğu bilinir ve olası çarpışmalarını hesaplamak için global zamandaki pozisyonunun göz önüne alınması yeterlidir. Bunun dışında molekülün yerel zamanı ile global zaman arasında bir ilişki bulunmamaktadır.

Hücre geçiş olaylarının sayısının çokluğu göz önüne alındığında molekülü güncellememenin ne kadar önemli bir iyileştirme olduğu çok daha iyi anlaşılabilir. Çünkü çarpışmalarla hücre geçişlerinin oranına bakıldığında hücre geçişi sayısı çarpışmalardan kat kat fazladır. Bu şekilde yerel zaman kullanımı ile molekülün pozisyonları sadece bir olaya katıldıklarında güncellenmektedir.

Pozisyon ve zaman bilgilerinin hassasiyeti mümkün olduğunca yüksek doğrulukta tutulmak istenmektedir. Bu bilgilerin sürekli güncellenmemesi hesaplamalarda meydana gelen bağıl hatayı da azaltmaktadır. Bir molekülün bilgilerinin her güncellenmesinden sonra belirlenen hassasiyetten sonraki basamaklarda meydana gelebilecek bağıl hata çoğalmaktadır.

N molekül için $O(N)$ olan güncelleme adımı, çok az hafıza işgal edilerek gereksiz hale getirilmiştir. Unutulmamalıdır ki bu değişiklik tahmin aşamasına ekstra bir yük getirmiştir ancak kazanımın çok daha fazla olduğu aşikârdır. Sonuç olarak yapılan bu iyileştirme problemin çözümü için verilen temel algoritmayı hızlandırmış ancak problem karmaşıklığını $O(N)$ 'den aşağı düşürmemiştir.

2.4 Öncelik Kuyruğu Kullanımı

Öncelik kuyruğu veri yapısının en önemli özelliği, adından da anlaşılacağı gibi içerdiği önceliği yüksek olan öğeye hızlı bir şekilde erişim imkânı sağlamasıdır. Diğer öğeler farklı kuyruklarda farklı sıralarda bulunabilirler. Öncelik kuyruklarında aranan bir diğer önemli özellik ise hızlı bir şekilde ekleme çıkarma işlemlerinin yapılabilmesidir. Literatürde birçok farklı kuyruk yapısı önerilmiştir. En çok bilinen kuyruk yapıları İkili Arama Ağacı (Binary Search Tree), yığın (Heap) olarak söylenebilir.

Problem karmaşıklığını azaltmak adına inceleyeceğimiz son iyileştirme çalışması öncelik kuyruğu veri yapısının EDMD simülasyonlarında kullanılmasıdır. Simülasyonun bir adımında olası çarpışmalar hesaplanıp, ilki seçilir ve bu olay işlenir. Bir sonraki adımda ise bir önceki adımda hesaplanan birçok olası çarpışma tekrar hesaplanmaktadır. Eğer bir önceki adımda hesaplanan olası çarpışmalar bir yerde saklansaydı; bunların içinden geçerli olmayanlar elenip yerlerine yenileri eklenerek yeni adımdaki olası çarpışmalar hesaplanabilirdi. Geçerliliğini yitiren çarpışmalar araştırıldığında ise görülebilir ki, sadece gerçekleşen olayın molekülleri ile ilgili olabilecek çarpışmalar geçerliliklerini yitirebilirler. Bu olaylar ise kolayca bulunabilir (ya da bu olayları aramaya hiç gerek kalmayabilir). Sistemdeki olası olaylardan en erken olanının belirli bir yerde bulunacağını garanti eden ve ekleme çıkarma operasyonlarının kolayca yapılmasını sağlayan bir veri yapısı EDMD için kullanılabilecek bir öncelik kuyruğu yapısı şeklinde ifade edilebilir.

Literatürde EDMD için kullanılan öncelik kuyruğu yapıları araştırıldığında en iyi bilinenler İkili Arama Ağacı üzerinde dairesel listeler yapısı (Rapaport 1980), Tam İkili Ağaç (Complete Binary Tree) (Marin and Cordero 1995) , Yığın ve ayrıca

son olarak (Paul 2007) tarafından önerilen hibrit kuyruk yapılarıdır. Bu yapılardan bazıları her molekülün olası bütün çarpışmalarını tutmaktadır, bazıları ise her molekül için molekülün olası çarpışmalarından sadece ilkinin tutmaktadır. Bu şekilde kuyruktaki olay sayısı sınırlandırılabilir. Bu yapı aynı zamanda hafıza kullanımı açısından da kullanışlıdır.

Simülasyonda işlenen bir olaydan sonra geçerliliğini yitiren olayları araştırmaya gerek kalmayabilir. Eğer bir olay işlenmeden önce geçerliliği kontrol edilirse, geçerliliğini yitiren olaylar bu aşamada ortaya çıkacaktır.

Kuyrukta her molekül için bir adet olay tutulduğunu varsayalım (bizim kullandığımız kuyruk yapısını göz önüne aldığımızda). Denilebilir ki her bir olay bir molekülü temsil etmektedir. Bilindiği gibi, bir olay işlendiğinde olaya karışan moleküllere yeni olaylar atanmaktadır. İşlenen olayın molekülünün yeni hesaplanan olayı kuyruğa eklenmelidir ancak, (varsa) çarpışma partnerinin yeni hesaplanan olayı eklenmemelidir. Bunun yerine sistemde geçerliliğini yitirmiş eski olayı ile değiştirilmelidir (bir anlamda geçerliliğini yitirmiş olan olay güncellenmelidir).

Güncelleme işlemine rağmen geçerliliğini yitirmiş olabilecek olaylar hala kuyrukta bulunabilir. Çünkü bir molekülün çarpışacağını tahmin ettiği başka bir molekül, bu olaydan önce başka bir olaya karışabilir ve sonuç olarak bu olayın geçerliliğini ortadan kaldırabilir. Geçerliliği yitirilen bu olay kuyrukta güncellenmeden bulunabilir. Akıllara bu olayların güncellenmemesi durumunda problem olup olmayacağı sorusu gelebilir. Ancak, geçerliliğini yitiren bu tür bir olayın molekülü geçersiz olaydan önce bir olay tahmini yapamayacağı için güncelleme geçersiz olay zamanına kadar bekletilebilir (“Lazy Deletion” Geç Silme Stratejisi). Bu yüzden kuyruktan işlenmek için çekilen her bir olayın geçerliliği öncelikle kontrol edilmelidir

2.5 Kullanılan Seri Algoritma Özellikleri

Yukarıda bahsedilen iyileştirmeler neticesinde söylenebilir ki kuyruk yapısının problem karmaşıklığı EDMD simülasyonunun problem karmaşıklığını belirleyecek

kısımdır. Anlatılan geliřtirmeler neticesinde EDMD simülasyonu algoritmasının oldukça deęiřtięi düşünölebilir. Anlařılır olması ağıısından algoritmayı tekrar bir gözden geçirecek olursak, EDMD simülasyonunun iřleniři řekil 2.5 EDMD Simülasyonunda Kullanılan Seri Algoritma.'te ki algoritma ile gereklenmektedir.

Algoritma 2 EDMD Simölasyonunda kullanılan seri algoritma

Olası olayları hesapla ve kuyruęa ekle.
 Simölasyon periyodu süresince ařaęıdakileri adımları tekrarla.
 Kuyruktan ilk olayı çek.
 Olayın geçerlilięini kontrol et ve geçerliyse gerekleřtir.
 Molekül(ler)e yeni olay ata ve kuyruęa ekle.

řekil 2.5 EDMD Simölasyonunda Kullanılan Seri Algoritma.

Olası olayların hesaplanmasını ve kuyruęa eklenmesini ilklendirme olarak sayılabilir. Bu kısım simölasyonda en bařta ve bir kez yapılmalıdır. Bir anlamda moleküllerin oluřturulması, bilgilerinin okunması gibi olaylarda ilklendirme olarak kabul edilebilir (Yapılan deneylerde olaylar bir kere hesaplanıp dosyaya yazılmış ve moleküller gibi dosyadan okunmuřtur).

Problem karmařıklıęını tekrar deęerlendirecek olursak; bir olayın geçerlilięini kontrol etmek ve bir olayı gereklemek $O(1)$ zaman almaktadır. Kuyruktan bir olayı çekmek ve kuyruęa yeni bir olay eklemek veri yapısına göre deęiřiklik göstermektedir. Bu durumda algoritma karmařıklıęını kuyruk yapısı belirlemektedir.

Yaptıęımız alıřmalarda uygulanabilirlięi ve idare etmesi kolay olduęunu düşöndüğümüz için “Yığın” (Heap) veri yapısını kullandık. Hem karmařıklık ağıısından $O(\log n)$ zamanda ekleme, silme iřlemlerinin yapılabilmesi hem de bir dizi üzerinde gereklenerek kolayca idare edilebilmesi yığın veri yapısını seçmek için bizi motive eden nedenlerdir. Biraz önce anlatılanlar ışıęında yığın veri yapısı kullanımıyla seri algoritmanın karmařıklıęı $O(\log n)$ olarak kabul edilebilir.

3 PARALEL EDMD ÇALIŞMALARI

Paralel EDMD simülasyonları için literatür araştırıldığında görülmüştür ki çok az sayıda çalışma bulunmaktadır. EDMD simülasyonunun paralelleştirilmesinin zorluğu, olaylar arasındaki sıkı ilişkilerden kaynaklanmaktadır. Bahsedilen bu ilişki olayların zaman ve konum açısından birbirlerine yakınlığından kaynaklanmaktadır. Çok yakın aralıklı zamanlarda ve konumda bulunan iki olay neticesinde biri diğerinin geçerliliğini ortadan kaldırabilir. Ayrıca geçerliliğini ortadan kaldırmaya bile sonrasında moleküllerin yapacağı yeni olayların araştırılması yanlış hesaplanabilir.

İki olayın konum olarak birbirlerine yakın olmalarından kasıt bu olayların bulundukları hücrelerin en az bir ortak komşu hücresinin bulunmasıdır. Örneğin A ve B olaylarını ele alalım ve bu iki olayın gerçekleştikleri hücrelerin bir ortak komşuları olsun. Bu olayların hücre geçiş olayları olduğunu ve her iki olayında ortak olan komşuya geçen moleküllerin olayları olduğunu varsayalım. Bu durumda A'nın molekülüne atanacak yeni olay ile B'nin molekülüne atanacak yeni olayın birbirlerini etkileme olasılıkları vardır. Bu olaylar bağımsız olarak düşünülemezler. Anlaşılabileceği gibi olaylar arasındaki ilişki sadece olay geçerliliğini değil olaylardan sonraki tahmin aşamasını da etkilemektedir.

Kısaca anlatılmak istenen şudur ki: simülasyon paralelleştirilirken klasik bir yaklaşımdaki gibi zamanlarda dilimleri ayrı olarak düşünülemez ve bağımsız olarak paralel işlenemez. Aynı şekilde farklı konumlara sahip olan olaylar da ayrı olarak düşünülemez. Buna rağmen bu tür problemleri aşmak için birkaç yaklaşım önerilmiş ve ortaya çıkabilecek durumlar incelenerek ortaya çıkan çalışmalar neticesinde bir iyileştirme yapılmıştır. Önerilen yaklaşımları inceleyecek olursak, sırasıyla aşağıdaki başlıklarda toplanabilir.

- Ortam Bölümlendirme
- Olay Bazlı Bölümlendirme
- Yönetici-İşçi yaklaşımı
- Hem olay bazlı bölümlendirme hemde yönetici-işçi kullanımı

Burada bahsedilen yaklaşımlar literatürde bulunan sayılı yaklaşımlardır. Her bir yaklaşımın kendi içerisinde karşılaştığı problemler ve sınırlar mevcuttur. Algoritma performansları incelendiğinde görülmüştür ki, ilk üç yaklaşımda etkili bir performans artışı sağlanamamıştır; sadece son bahsedilen çalışma ile hızlanma elde edilmiştir. Algoritmaları daha yakından incelersek performansları ve sınırları hakkında daha çok şey söyleyebiliriz.

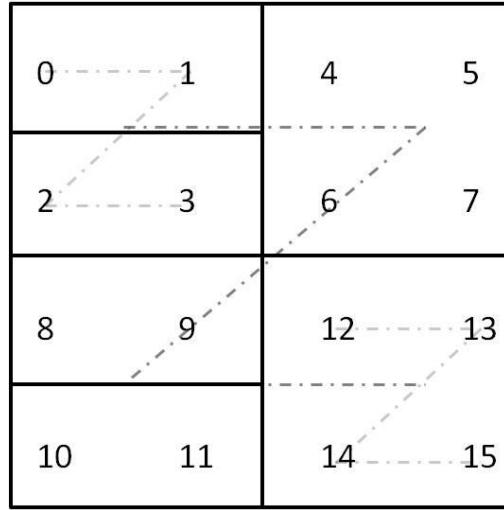
3.1 Ortam Bölümlendirme

İlk olarak inceleyeceğimiz algoritma, ortam bölümlendirme yaklaşımının EDMD simülasyonu için bir uygulaması olacaktır. Algoritmanın temel prensibi ortamı farklı hesaplama düğümlerine paylaştırarak performans artışı sağlanmasıdır. (Miller and Luding 2003)

Bilindiği gibi simülasyon ortamı daha önce çok sayıda hücreye bölünmüştür. Aynı yaklaşımla P adet işlemcimiz olduğu bir durumda, simülasyon ortamı P parçaya bölünmesi düşünülmüştür. Parçalama rastgele bir şekilde değil de “İkili Parçalama Stratejisi” (Binary Partitioning Strategy) kullanılarak yapılmıştır. Bu sayede yük dağılımının olabildiğince eşitlenmesi sağlanırken işlemciler arasındaki haberleşme ihtiyacının en küçüklenmesi amaçlanmıştır. Molekül yoğunluğu, simülasyon devam ettikçe simülasyon ortamının farklı bölümlerinde değişebilir. Bu nedenle belirlenen zamanlarda bölümlendirmenin tekrarlanması sağlanmıştır. Burada amaç iş yoğunluğunun olabildiğince adil olmasını sağlamaktır.

Bölümlendirme yaparken hücre numaraları oldukça kullanışlı olmuştur. Hücre numaraları ağaç yapısına benzer bir yapıda verilmiştir. Hücrelerin numaralarına ikili gösterim (Binary representation) mantığına göre, her bir koordinata 1 ile k arasında serpiştirme (Interleaving) kullanılarak sayılar verilerek, hücre numaraları $x[1], y[1], z[1] \dots x[k], y[k], z[k]$ arasında elde edilmiştir. Sonuç olarak hücre sayısının D boyut sayısı olmak üzere $2^{kD} - 1$, olarak alınması düşünülmüştür. Bu sayede ardı ardına bulunan 2^m adet hücre bloğu bir dikdörtgen prizması halinde düşünülebilir. 2 boyutlu olarak anlatmak için Şekil 3.1 2 boyutlu örnek

parçalama.'te verilen örnek açıklayıcı olacaktır. Örnekte 6 işlemci için ortaya çıkabilecek bir bölümlendirme verilmiştir.



Şekil 3.1 2 boyutlu örnek parçalama.

İkili parçalama stratejisi şu şekilde uygulanmaktadır. N molekül için N olay ele alınırsa (her bir molekül için kuyrukta bir olay tutulduğundan dolayı) ve ortam P adet işlemciye bölünmek istenirse her bir işlemciye ortalama N/P yük düşeceği düşünülebilir. Bu en iyi olabilecek yük dağılımıdır. Hücre numaralarını kullanarak baştan başlayıp ardışık m adet hücre seçilir (her bir işlemci için m sayısı değişebilir). Seçilen m hücre içerisinde yaklaşık N/P olay olması aranır. Uygun olarak bulunan m sayısı kadar hücre bir işlemciye atanır. Bu şekilde bütün işlemcilere simülasyon ortamının bir alt parçası atanmış olur.

Bölümlendirilen ortamlar arasında hücre geçişleri (ve hatta çarpışmalar) olabileceğinden dolayı bu bölümler birbirlerinden ayrık olarak düşünülemezler. Bu durumda işlemciler eş zamanlı hareket etmeli ve birbirleri arasında haberleşerek bu olayların tutarlı bir şekilde işlenmesini sağlamalıdır. Eş zamanlı hareket etmek EDMD'nin zamandan bağımsız olan yapısından dolayı oldukça zordur. Bunun yerine optimistik bir yaklaşım ve geriye-alma (roll-back) protokolü kullanılmıştır. Bunun için belirli aralıklarla sistem kaydedilmelidir. Ayrıca tek bir kayıt yeterli olmayabilir. Belirlenen kayıt sayısı kadar sistemin yedeklerinin alınması hafıza kullanımını aynı sayıda arttırmaktadır.

İşlemciler belirlenen zaman aralıklarında birbirlerinden bağımsız bir şekilde olayları işlerler. Tutarlılığı bozacak bir durum olması halinde geriye-alma uygulanır. Yazarın kullandığı yedek kayıt sayısı 2'dir ve yazar 2 kaydın yeterli olacağını söylemektedir (Miller and Luding 2003).

Paralel algoritmanın işleyişini vermeden önce belirtmemiz gereken son şey “*Sınır Hattı*” (border zone) tanımıdır. Her işlemciye atanan ortamın diğer işlemcilerin ortamlarıyla komşu olduğu hücreler sınır hattı olarak adlandırılmıştır. Bu hücreler komşu oldukları işlemcilere bildirilerek çarpışma partneri olarak bu hücrelerdeki moleküllerinde hesaba katılması sağlanmıştır. Aksi halde tutarsızlık olabilecek durumlar oluşabilir. Bir işlemci komşusunun sınır hattındaki bir hücrede bulunan bir moleküle olay atayamaz ancak onu çarpışma partneri olarak araştırabilir. Sınır hattında bulunan hücreler komşu işlemciye bir nevi sanal hücre olarak atanır denilebilir.

Algoritma 3 Ortam Bölümlendirme Yöntemi ile Paralleleştirme

Sınır Hattı olaylarının en erkenini bul.
 Planlanmış olayların en erkenini bul.
 Zaman adımını bu değerlerin en küçüğü olarak seç.
 Zaman adımına kadar olayları paralel olarak işle.
 Molekül bilgilerinin haberleşmelerini gerçekleştir.
 Hata kontrolü yap.
 Eğer hata varsa
 geriye-alma gerçekleştir.
 Zamanlanmış görev varsa gerçekleştir.

Şekil 3.2 Ortam Bölümlendirme Yöntemi ile Paralleleştirme.

Algoritma ilk olarak dinamik bir şekilde senkronizasyon zamanını belirler. Bunun için sınır hatlarında olabilecek en küçük olay ile bölümlendirme veya yedekleme gibi daha önce olabilecek görevleri karşılaştırır. Bunlardan en küçük olanı senkronizasyon zamanı olarak seçilir. Bu zamana kadar işlemciler kendi bölgelerini paralel bir şekilde işlerler. Birbirlerine sınır hatlarındaki molekül bilgilerini gönderirler ve hata kontrolü yapılır. Zaman adımına kadar başka bir molekül bir başka işlemcinin bölgesine geçmiş ve tutarlılığı bozmuş olabilir. Hata kontrolü

sonrası gerekirse geriye-alma işlemi yapılır. Gereken yedekleme, tekrar bölümlendirme gibi tasarlanmış görevler gerçekleştirilir ve simülasyonda ilerleme gerçekleştirilmiş olur.

Probleme getirilen çözüme hızlı bir değerlendirme yapacak olursak; işlemciler arası molekül geçişleri senkronizasyonun zorunlu ve sık yapılmasını gerektirmektedir. Birçok defa geriye-almalar kaçınılmazdır. Ortamı eşit yük dağılımı sağlayacak ve minimum sayıda geçiş olayı olacak şekilde bölmek arasında bir değiş tokuş vardır.

Yöntemin çalışması birçok kritik noktaya bağlıdır. İşlemciler arası haberleşme, geriye-alma, senkronizasyon ve işlemcilerin boşa geçen zamanlarından dolayı yapılan deneylerde en fazla $\sqrt{P}$ kat hızlanma sağlanabilmiştir.

3.2 Olay Bazlı Bölümlendirme

Bir diğer çalışmada ise ortam bölümlendirmesi yerine olay bazında bir bölümlendirme düşünülmüştür (Herbordt, Khan and Dean 2009). Algoritma dağıtık bir mimariden ziyade çok çekirdekli bir mimari üzerinde EDMD'nin paralel olarak işlenmesini amaçlamaktadır. Olaylar, iş parçacıkları (Threads) tarafından bağımsız olarak işlenir ve tutarlılığı sağlayacak şekilde sistemde güncellenir. Hafıza kullanımı dağıtık değil paylaşımlıdır. Bu sayede iş parçacıkları arası haberleşme, ortak hafıza üzerinde çok daha az zamanda yapılmaktadır. Olay bazlı bölümlendirme çalışmasında kullanılan algoritmayı inceleyecek olursak:

Algoritma 4 Olay Bazlı Bölümlendirme Yöntemi ile Paralleleştirme

Her iş parçacığı için
 En küçük olayı seç.
 Olayı gerçekleştir.
 Moleküllere yeni olay ata.
 Sırası geldiğinde sistemi güncelleştir.

Şekil 3.3 Olay Bazlı Bölümlendirme Yöntemi ile Paralleleştirme.

Sistem dışarıdan bakıldığında bir “*Boru Hattı*” (Pipeline) modeli olarak düşünülebilir. İş parçacıkları yerine işçiler diyecek olursak, işçiler her ne kadar bağımsız çalışsalar da güncelleme aşaması seri yapılarak tutarlılık sağlanmak istenmiştir. Bahsedilen olaylar arasındaki yakın ilişki, bu tip bir yaklaşımda çok fazla problemler ortaya çıkarmaktadır. Problemlerden bir kısmı kilit mekanizmaları ile çözülebilir ancak olayları paralel işlemenin doğasından gelen nedensellik (Causality) ve tutarlılık (Cohenrence) problemleri daha farklı şekilde çözülmelidir. Bu problemleri ve çözümlerini algoritmayı anlattıktan sonra inceleyeceğiz.

Öncelikle algoritmanın seri halini anlatmak gerekirse; Bir olay işlenirken sırasıyla önce kuyruk işlemleri yapılır ve en erken olay çekilir. Ardından olay gerçekleşir (ancak bilgiler sistemde güncellenmez). Daha sonra moleküllere yeni olaylar atanır. Son olarak sistemdeki güncelleştirmeler yapılarak ilerleme sağlanmış olur.



Şekil 3.4 Tek bir olayın işlenme döngüsü.

Seri algoritma üzerinden yola çıkarak olay bazlı bölümlendirmeyi ortaya atan paralel algoritmada ise işçiler sırayla şu şekilde çalışmaktadırlar. Öncelikle kuyruktan bir olay çekerler. Kuyruktan olay çekme işlemi kilit mekanizması ile kontrol edilir ve her seferinde sadece bir işçi kuyruktan olay çeker. İşçiler olayları paralel olarak işlerler ve sadece güncellemeler zaman sırasına göre seri olarak yapılır. Zaman sırasına göre zamanı gelen olay sistemde güncellenirken eğer işini bitiren bir işçi varsa beklemeye geçer. Her hangi bir sebeple bir olay sırası gelinceye kadar geçerliliğini yitirirse bu olay baştan tekrar işlenir. Bu süre zarfında zaman olarak daha sonraki olaylara sahip olan işçiler boşa beklemek zorunda kalırlar. Algoritmanın işleyişini daha iyi anlamak için aşağıdaki şeklin incelenmesi anlamlı olabilir.



Şekil 3.5 Paralel olay işleme döngüsü

Şekilden de anlaşılabileceği gibi sistem bir boru hattı gibi çalışmaktadır. İşçi sayısı arttıkça ilişkili olayları işleme olasılığı artar. Bu sebeple işçi sayısı çok yüksek sayılara ulaşamaz. Arttıkça da performans açısından iyileştirme oranı işçi sayısını arttırmaya göre çok düşük kalacaktır. Bu sebeple olay bazlı bölümlendirme çok çekirdekli sistemlere daha uygundur.

Yöntem, (Paul 2007) tarafından önerilen öncelik kuyruğu yapısını kullanmaktadır. Bahsedilen kuyruk yapısında ekleme silme gibi işlemler $O(1)$ zamanda yapılmaktadır. Bu anlamda kuyruk yapısı bizim kullandığımız yığın yapısından daha düşük çalışma karmaşıklığına sahiptir. Ayrıca paralel olarak işlenmeye daha uygun bir yapı olduğu düşünülmektedir.

Sistemde birçok kilit mekanizması bulunmaktadır. Bir olay işlenirken olaya karışan moleküller kilitlenir. Ayrıca olayın bulunduğu hücre de kilitlenir. Kuyruk ekleme ve silme işlemleri esnasında bir tutarsızlığa karşı kilitlenir. Kilit mekanizmalarıyla hatalardan kaçınılabilir ancak yine de senkronizasyonu sağlamak zordur. Olaylar birbirlerini etkileyebilirler. Senkronizasyon için gerekli bütün durumlar düşünüldüğünde, dikkat edilmesi gereken birkaç husus ortaya çıkmıştır. Bu problemleri ve problemlere getirilen çözümleri daha yakından inceleyecek olursak;

- Sebep Sonuç Problemi (Causality)
- Uyumluluk, tutarlılık Problemi (Coherence)
- Tutarlılık ve sebep sonuç problemlerinin birbirlerini tetiklemesi (Both Causality and Coherence)

3.2.1 Sebep Sonuç Problemi (Causality)

Önceki kısımlarda da belirtildiği gibi iki olay bağımsız bir şekilde işlendiğinde biri diğerrinin geçersiz olmasına yol açabilir. Örneğin A ve B olaylarını göz ününe alalım ve A olayı B'den önce olsun. A, m_1 ve m_2 moleküllerinin çarpışma olayı ve B, m_3 ile m_2 moleküllerinin çarpışma olayı olsun. Bu durumda A'nın sonucunda B geçersiz duruma düşecektir.

3.2.2 Uyumluluk, Tutarlılık Problemi (Coherence)

İki olay birbirlerini direk olarak etkilemeseler bile sonuçlarında tahmin edilecek olan olaylar yanlış hesaplanabilir. Aynı şekilde A ve B olaylarının bağımsız işlendiğini düşünelim ve bu olaylar konum olarak birbirlerine yeteri kadar yakın olsunlar. İki olayında tahmin aşamasında olduklarını ve henüz moleküllerinin sistemde güncellenmediğini varsayalım. Bu durumda olayların molekülleri sistemdeki pozisyon ve hızlarını korumaktadır. B'nin moleküllerinden bir tanesi tahmin aşamasında, A'nın moleküllerinin kendisinden önce değişeceğini bilmeden yanlışlıkla kendisine çarpışma partneri olarak seçebilir. Ancak bu durum A'nın sonucunda B'nin iptal edilmesine yol açmadığı halde en az birinin tahmin aşamasının tutarsız hale gelmesi anlamına gelir.

3.2.3 Tutarlılık ve sebep sonuç problemlerinin birbirlerini tetiklemesi (Both Causality and Coherence)

İki olay herhangi bir tutarlılık ya da sebep sonuç problemiyle karşılaşmadan paralel olarak işlenmiş olsalar bile aralarında hala bir problem oluşabilir. Yine A ile B olaylarının paralel işlendiklerini düşünelim. A'nın sonucunda yeni bir olay sisteme B'den önce olacak şekilde eklenebilir. Sisteme yeni eklenen olay ile B sebep sonuç problemine sahip olabilirler.

3.3 Varsayımsal Olay İşleme

İnceleyeceğimiz son çalışmada ise olay bazlı bölümlendirmeye benzer olarak varsayımsal olay işleme (Speculative Execution) algoritması önerilmiştir (Su, Jiang ve Zhang 2007). Yönetici-İşçi (Master-Worker) modeli kullanılarak, yönetici tarafından işçilere kuyrukta bulunan olası olaylar dağıtılır. Yönetici ayrıca kendisi de olay işlemeye devam eder. Sırası gelen bir olay eğer bir işçi tarafından işlenmiş ise hesaplanan veriler işçiden alınır ve kullanılır.

Olası olayların dağıtımı ve gerektiğinde bilgilerin toplanması, sadece yönetici tarafından yapılabildiğinden yöneticinin görevi olan simülasyonu ilerletme işlevi bir çok kez sekteye uğramaktadır.

Hücre bölümlendirmenin kullanıldığını düşünürsek, kuyrukta bulunan birçok olay hücre geçiş olayı olacaktır. Bu olayların dağıtılması ve bilgilerin toplanması tahmin edilebileceği gibi direk işlenmesinden çok fazla zamana alacaktır. Çünkü bizim simülasyonumuzda bir hücre geçiş olayı, molekülün bulunduğu hücre bilgisinin güncellenmesi ve yeni olayının hesaplanması anlamına gelmektedir. Bu tip bir olayın dağıtımı, tutarlılığının kontrolü ve toplanması bu bilginin güncellenmesinden çok daha zahmetli bir iştir. Bu durumda varsayımsal hesaplamalar birçok kez seriden fazla zaman alacaktır. Yazarların fikrini daha iyi anlatmak için Şekil 3.6 Varsayımsal Olay İşleme Algoritmasına Bir Örnek'te olası bir çalışma durumu örnek olarak anlatılmıştır.

Yönetici	İşçi 1	İşçi 2	İşçi 3
E0 Gerçek Hesaplama	E1 Varsayımsal Hesaplama	E2 Varsayımsal Hesaplama	E3 Varsayımsal Hesaplama
E1 Varsayımsal			
E2 Varsayımsal	E4 Varsayımsal Hesaplama	E5 Varsayımsal Hesaplama	E6 Varsayımsal Hesaplama
E3 Varsayımsal			
E4 Gerçek Hesaplama	E7 Varsayımsal Hesaplama	E8 Varsayımsal Hesaplama	E9 Varsayımsal Hesaplama
E5 Varsayımsal			
E6 Varsayımsal	E10 Varsayımsal Hesaplama	E11 Varsayımsal Hesaplama	E12 Varsayımsal Hesaplama
E7 Varsayımsal			

Şekil 3.6 Varsayımsal Olay İşleme Algoritmasına Bir Örnek

Sistemde bulunan en erken olay yönetici tarafından işlenir. Bu süre zarfında işçilerde olası olayları işlerler. İşini bitiren bir işçi yöneticiden kendisine yeni olay vermesini bekler ve bu durumda işçiler boşa çok fazla zaman harcamaktadırlar. Yöneticinin de aynı zamanda olay işlemesi onu bir anlamda darboğaz (bottleneck) haline getirmiştir. Olaylar sürekli senkron bir şekilde işlenir bu durumda bir çok tutarlılık kontrolü yönetici tarafından kontrol edilir. İşçiler olay işlemekten başka bir iş yapmazlar. Yönetici sırası geldiğinde işlenecek bir olay varsayımsal olarak işlenmiş ise geçerliliğini kontrol eder ve mümkünse işlenmiş hesaplamaları kullanır. Geçerliliğini yitiren hesaplama sonuçları yönetici tarafından tekrar hesaplanır. Örneğin şekilde E4 olayı yönetici tarafından tekrar hesaplanmaktadır.

Bir önceki algoritmada bahsedilen tutarlılık ve sebep-sonuç tipi problemler burada da ortaya çıkabilir. Bu durumlardan kaçınmak için yazarlar molekülleri, olayları ve hatta hücreleri kilitlenmeyi seçmişlerdir. Bu durum performans açısından en çok kaçınılması gereken durumdur.

Yazarlar performans değerlendirmesi yaparken seri ile paralel çalışma zamanlarını karşılaştırmamışlardır. Sadece paralel çalışma zamanları arasında bir karşılaştırma yapılmıştır. Bundan dolayı etkili bir hızlanma elde edip edemedikleri hususunda bir bilgiye ulaşmak mümkün değildir.

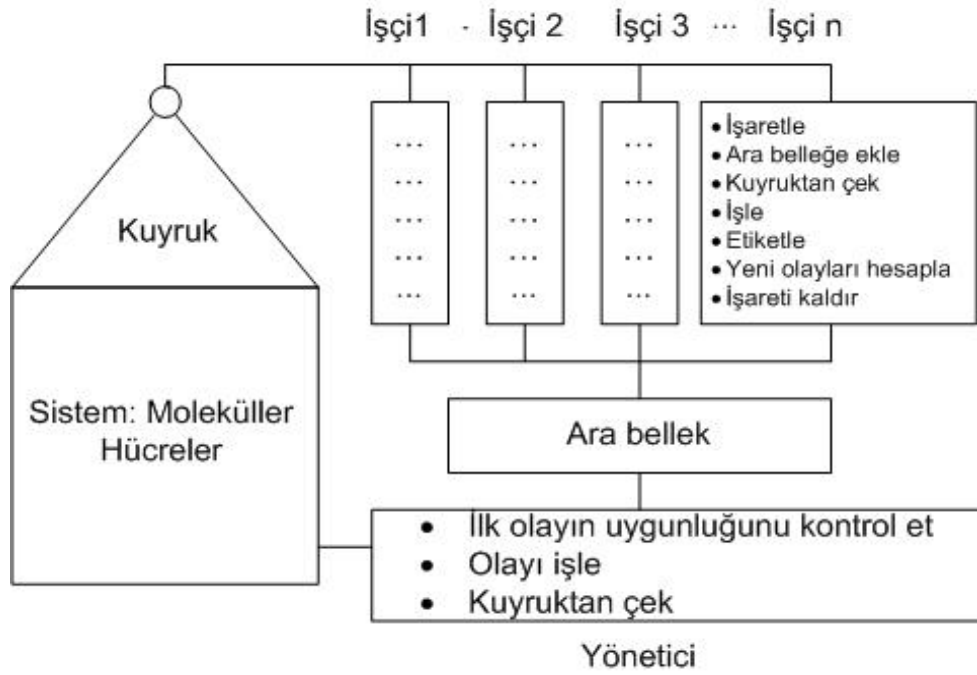
4 EDMD İÇİN ASENKRON PARALEL ALGORİTMA

Geliştirilen algoritmalar incelendiğinde olay bazlı işleme yöntemlerinin daha gelişmeye açık olduğu düşünülmüş ve bu yolda ilerlenmeye karar verilmiştir. Birbirlerine göre algoritmaların iyi ve kötü yönleri düşünüldüğünde, olayları kuyruktan bir yöneticinin çekip dağıtmasındansa işçilerin kendilerinin almasının daha az senkronizasyon yükü getireceği düşünülmüştür. Tasarladığımız algoritmada anlatılan algoritmalarından ikincisinde olduğu gibi işçiler kuyruk yapısına direk kendileri ulaşarak olası olayları alıp işleyebilmelidirler.

Sistem güncelleştirme işleri için bir yönetici kullanımı, işçilerin her ne olursa olsun birbirlerini beklemelerini ortadan kaldıracak görülmüştür. Bu durumda işçiler ile yönetici arasındaki ilişki, üretici tüketici problemine dönüşmektedir. İşçiler olası olayları işler ve hiç beklemeden yöneticiye verirler, yönetici ise bu olayları düzgün bir şekilde sistemde güncelleme işlerini üstlenir. Bu yapıda, anlatılan üçüncü algoritmadan farklı olarak yönetici olay işlememektedir. İşçiler ise sistem güncelleştirme işlemleriyle uğraşmadıklarından birbirlerini beklemek zorunda kalmamaktadır. İşçiler işledikleri olayları yöneticiye iletirken birbirlerini beklememek için bir arabellek kullanımı düşünülmüştür. Bir işçi bir olayı işledikten sonra ara belleğe hesaplamaları yazıp, hiç beklemeden kuyruktan yeni bir olay çekerek işine devam edebilir.

Ara bellek kullanımı olaylar arasındaki ilişkileri bulup tutarsızlıkları ortadan kaldırabilmeyi sağlamaktadır. Bir işçi bir olayı ara belleğe ekledikten sonra ara bellekte bulunan diğer olaylarla işlediği olay arasındaki olası tutarsızlıkları araştırarak tutarsızlık problemleri ortadan kaldırılabilir. Bu problemlerin ortadan kaldırılması için ayrıca yöneticinin de yapması gereken işler vardır.

Algoritmanın çalışma şeklini daha iyi anlatmadan önce Şekil 4.1 Geliştirilen Yeni Algoritma İçin Şema’i incelemek uygun olabilir.



Şekil 4.1 Geliştirilen Yeni Algoritma İçin Şema

Şekilde sol tarafta kuyruk yapısı, moleküller ve hücreler betimlenmiştir. Üst tarafta işçilerin asenkron çalışmaları gösterilmiş, aşağıda ise ara bellek ile yönetici gösterilerek yapının genel bir şekli anlatılmak istenmektedir.

Tutarlılık ve uyumluluk problemlerini ortadan kaldırmak için gereken en önemli unsur aynı anda bütün olayların sistem içerisinde ve erişilebilir olmasıdır. Bunun için bizim geliştirdiğimiz yapıda bir olay kuyruktan çekilmeden önce ara belleğe eklenir. Aynı şekilde ara bellekte bulunan işlenmiş bir olay sisteme güncellenmeden ara bellekten çekilmemektedir.

Burada kontrol edilmesi gereken birkaç durum ortaya çıkmaktadır. Öncelikle bir olay sadece bir işçi tarafından çekilebilmelidir. Bunu sağlamak için olaylara atomik bir işaret eklenmiş ve bu sayede bir olayın aynı anda birden fazla işçi tarafından işlenmesi önlenmiştir. Bir işçi kuyruktan olay çekmeden önce kuyruktaki en erken olayı işaretlemek istemektedir. Eğer olay daha önce bir başka işçi tarafından işaretlenmiş ise işaretleme işlemi sonucu doğru döndürülmez ve işçi istediği olayın başka bir işçi tarafından daha önce işaretlendiğini anlamaktadır. İşçi çok kısa bir süre bekleyerek tekrar kuyruktan olay çekmeye çalışır. Eğer olay henüz hiçbir işçi

tarafından işaretlenmemiş ise işaretlenir ve doğru cevabı işçiye geri döndürülür. Bu sayede olayın kendisine ayrıldığını anlayan işçi olayı kendisine kopyalar, ara belleğe ekler ve kuyruktan çeker. İşaretleme işlemi atomik bir şekilde yapıldığından aynı anda iki işçi aynı olayı işaretleyemezler. İşaretleme işlemleri esnasında her hangi bir kilitleme işlemi kullanılmaz. Bu işaret aynı zamanda yöneticiye olayın hala işlendiğini belirtmek içinde kullanılır. Ara belleğe eklenip, kuyruktan çekilen olay hesaplamaları bitene kadar işçi tarafından hala işaretli durumda tutularak yöneticiye olayın hala hesaplama aşamasında olduğu anlatılır. İşçi hesaplamaları bitirip olayın üzerindeki işareti kaldırarak yöneticiye olayın artık hesaplamalarının bittiğini bildirmiş olur. Bu sayede ara bellekte yönetici tarafından henüz hesaplamaları bitmemiş bir olayın işlenmesi engellenmiş olur.

Algoritmayı ayrıntılı olarak incelemek için yönetici ile işçilerin çalışma adımları ayrılarak verilmiştir. Şekil 4.2 İşçiler için Algoritma'de bir işçinin sırasıyla gerçekleştirdiği adımlar aşağıda verilmiştir.

Algoritma 3(a) İşçiler için Algoritma
Simülasyon periyodu süresince aşağıdakileri adımları tekrarla. Kuyruktaki ilk olayı işaretle Olayın kopyasını al, arabelleğe ekle ve kuyruktan çek. Kopya olayı gerçekleştir ve sonuçları arabelleğe yaz. Komşuluk kontrolü yap ve gerekliyse olayı etiketle. Eğer olay etiketlenmediyse yeni olay(lar)ı hesapla İşareti kaldır ve yöneticiye işlemek için izin ver.

Şekil 4.2 İşçiler için Algoritma

Bir olay ara belleğe eklenip kuyruktan çekildikten sonra işçi tarafından işlenmeye başlanır. İşçi zamanı henüz gelmeyen bir olayı işlerken eğer bilgileri sisteme güncellerse hata yapılmış olur. Bunun için molekülün bu bilgilerinin başka bir yere yazılması gerekir. Aynı şekilde molekülün bilgileri işçi tarafından kullanılmaya başlandığı sırada yönetici tarafından sistemde güncelleniyor olabilir, bu sebeple işçi olayı işlemeye başlamadan önce bir kopyasını oluşturur. Hesaplanan bilgilerde bu kopya üzerine yazılır. Bir anlamda, işçi olayın kopyasının işler ve

sonuçları buraya yazar. Orjinal olay hala olası problemler için gerekmektedir. Tutarsızlık veya geçerlilik problemleri bunlara örnek olarak verilebilir.

İşaret mekanizmasının dışında ara bellekte bulunan olaylar için bir etiket mekanizması geliştirilmiştir. Tutarlılık ve sebep sonuç problemlerinin çözümü için olaylar ara belleğe eklendiğinde işçiler ekledikleri olaylara bir etiket vererek yöneticiye bu durumları bildirirler. Bu sayede olası problemler ortadan kaldırılmış olur. Etiketleme mekanizması, varsayımsal işleme algoritmasında anlatılan yöneticinin tutarsızlık araştırması işlerini işçilere vermek için tasarlanmıştır. Bu sayede yöneticinin bir darboğaz olması azaltılmak istenmiştir.

Bir olay işlenirken ortaya çıkabilecek olası problemler incelendiğinde, sadece zaman ve konum açısından belirli yakınlıkta olan olaylar arasında problemler çıkabileceğini görülmüştür. O halde bu olayların etiketlenmesi yeterli olacaktır. Ara bellekte bulunan olaylar zamandan bağımsız, olay bazlı işlendiklerinden dolayı konum olarak ilişkili olanların hepsi işaretlenmelidir. İki olayın birbiriyle ilişkili olması için ikisinin de olay sonrasında en az bir ortak komşu hücreye sahip olması gerektiğini biliyoruz. Bu durumda işlenen iki olayın bulundukları hücrelere bakılır ve bu hücreler arasındaki uzaklık iki komşu hücrenin arasında olabilecek en büyük uzaklıkla karşılaştırılır. Bu mesafe bir hücrenin en uzun köşegeni düşünülürse bu köşegen uzunluğunun 2 katı olacaktır. Hücrelerimiz küp şeklinde olduğundan bir kenar uzunluğu N olan hücre için bu mesafe $2N\sqrt{3}$ olarak bulunmuştur.

Etiketleme işleminde mantıksal değer (boolean value) yerine tam sayı kullanımı olaylar arasındaki ilişkileri daha iyi betimlememizi sağlamaktadır. Etiketin varsayılan değeri “0” dır. Bunun anlamı bu iki olay ilişkisiz demektir. İki olay eğer bu mesafeden küçük olan olaylar ise ilişkili olabilir diyerek “1” etiketi ile işaretlenir. Bunun anlamı moleküllerin konum zaman hesaplamaları doğru ancak gelecek olası olay hesaplamaları yanlış olabilir demektir. Aynı zamanda mesafelerinde bakılmaksızın eğer olaylara karışan moleküllerden herhangi biri ortak ise daha geç olan olay “2” ile etiketlenir. Bu ise moleküllerin konum zaman bilgileri de hatalı olabilir demektir. Son olarak ikiz olaylar denilebilecek olaylardan daha geç olanı “3” ile etiketlenir. Bu olaylar A ile B moleküllerinin çarpışması olayıyla, bunun eşleniği olan B ile A moleküllerinin çarpışması olayı denebilir.

Etiketleme işlemi sonrasında yeni olayların hesaplanması aşamasına geçilir. Etiketlenmemiş bir olayın yeni olayları hesaplanır. Ancak etiketlenmiş bir olayın yeni olayları hesaplanmaz. Çünkü bu hesaplamalar büyük olasılıkla yanlış olacaktır. Bu hesaplamalar yöneticiye bırakılmıştır. Bu aşamadan sonra işçi olay üzerindeki işareti kaldırılır ve bir olayın işlenmesi tamamlanmış olur. İşçi yeniden kuyruktan bir olay çekmeye çalışır.

Olabildiğince bütün hesaplamalar işçilere verilmek istenmiştir. Bu sayede yöneticinin işleri azaltılmak istenmiştir. Yöneticiye sadece sistem güncelleme işleri bırakılmak istensen de etiketli olaylardan dolayı başka işler de yüklenebilmektedir. Yöneticinin çalışma aşamaları incelenmeden Şekil 4.3 Yönetici için Algoritma'teki algoritma verilebilir.

Algoritma 3(b) Yönetici için Algoritma
Simülasyon periyodu süresince aşağıdakileri adımları tekrarla. Arabellekteki olayın sistemdeki ilk olay olup olmadığını kontrol et (Eğer değilse bekle). Arabellekteki olayın işaretinin kalkmasını bekle (tamamlanıp tamamlanmadığını kontrol et ve tamamlanmamışsa bekle). Olayın geçerliliğini kontrol et (Olay işlendikten sonra geçerliliğini yitirmiş olabilir). Eğer geçerliyse hesaplanmış verileri sistemde güncelle. Olayı arabellekten kaldır.

Şekil 4.3 Yönetici için Algoritma

Yöneticinin esas işi ara bellekte olan olayları sistemde güncellemektir. Bunun için öncelikle belleğin boş olup olmadığını kontrol eder. Boş ise bir olay ekleninceye kadar bekler. Ardından ara bellekteki ilk olayın kuyruktaki ilk olaydan daha erken olup olmadığını kontrol eder. Eğer daha erken ise son olarak işaretli olup olmadığını kontrol eder. İşaretli ise halen işleniyor anlamındadır ve işaretinin kalkmasını bekler. İşareti kalkan olayın sistemdeki ilk olay olduğu artık aşîkârdır.

Olay bilgileri sistemde güncellenmeden önce orjinal olayın geçerliliği son bir kez kontrol edilerek moleküllerin başka bir çarpışma yapıp yapmadıklarına bakılır. Eğer olay geçerli ise önceden hesaplanan değerler sistemde güncellenir. Güncelleme

işlemi sırasında molekül bilgilerini kopyalamak isteyen bir işçi olabileceği için okuyucu-yazıcı kilitleri (Reader-Writer Lock) ile molekül bir anlık kilitlenerek bilgileri güncellenir. Molekül bilgilerinden sonra yeni hesaplanan olaylar kuyrukta güncellenmelidir. Olayın sahibi olan molekülün yeni olayı kuyruğa eklenir ancak çarpışma partnerinin yeni hesaplanan olayı ile kuyrukta var olan eski olayı değiştirilerek güncellenmelidir. Güncelleme işlemlerinden sonra yönetici ara bellekten bu kaydı siler ve yeni bir olay işlemek için tekrar ara belleğe bakmaya başlar.

4.1 Ara Bellek Özellikleri

Ara belleğin tasarımı düşünüldüğünde sahip olması beklenen özellikler öncelikle sıralanmalıdır. Tasarım bu özelliklere göre zaten bir düzeyde şekillenecektir. Aranılan en önemli özellik ara belleğin eşzamanlı (concurrent) erişime izin vermesi gerekmektedir. Birçok işçi aynı anda olaylar eklemek isteyebilirler. Eş zamanlı silme işlemine henüz gerek yoktur; çünkü sadece bir yönetici vardır ve sadece o silme işlemi gerçekleştirebilir. Bunun dışında eklenecek her olay zaman sırasına göre doğru sıraya eklenmelidir (insertion). Bu sayede ara bellek tekrar bir sıralama işlemine gerek kalmadan kullanılabilir. Bu aynı zamanda yöneticinin de işini kolaylaştırmaktadır; çünkü her seferinde sadece bellekteki ilk olayı ele alarak işlem yapabilmektedir. Ayrıca hızlı çalışmalıdır ki düşünülen sistem için bir darboğaz haline gelmeden kullanılabilmelidir.

Anlatılan bütün özellikleri sağlayacak bir veri yapısı düşündüğümüzde karşımıza atomik, kilitsiz ve sıralı bir liste (atomic lock-free sorted list) yapısı çıkmaktadır. Bu listeyi soyut bir veri tipine göre gerçekleyerek istediğimiz veri tipinde kullanabilmeyi amaçladık. Listenin boyutu konusunda öncelikle bir sınır belirlenmesi düşünülmeyse de, yönetici işçilere göre yavaş kaldığından dolayı listenin hızlı bir şekilde büyüdüğü görüldüğünden, listeye belirli bir sınır verilmesi uygun görülmüştür.

Listeye limit konulması sonrasında işçiler bir olay işlemeye başlamadan önce listenin boyutunu kontrol ederler ve liste müsait ise olay işlemeye başlarlar. Bu

durumda listenin boyutu bazen aşılabilir ancak genel olarak belirli bir düzeyde tutulmuş olur. Aynı şekilde listenin boyutu yeni bir olay işlemeye uygun olamasa da kuyrukta olabilecek daha erken bir olaydan dolayı yönetici yeni bir olayı işleyemez ise kilitlenme (deadlock) durumu söz konusu olabilir. Bunu için eğer kuyrukta bulunan bir olay ara bellekteki olaylardan daha önce ise listenin boyuna bakılmaksızın bir işçi tarafından alınır işlenir ve ara belleğe eklenir. Listenin boyutu ile işçi sayısı arasında oluşacak ilişki gözden kaçmamalıdır. Listenin boyutunun çok küçük olması işçilerin iş yapamayacak şekilde sıkışmaları anlamına gelmektedir. Listenin boyunun çok büyük olması ise birçok geçersiz olayın işlenmesi ve tutarlılık kontrolünün belirgin şekilde zaman alması anlamına gelmektedir.

4.2 Koşut Zamanlı Kuyruk Yapısı Uygulaması

Yöneticinin işçilere göre daha fazla zaman almasının en önemli nedeni kuyruk işlemlerinin, yığın veri yapısında kilit mekanizmalarıyla kontrol edilmesidir. Yönetici molekül bilgilerini çok kısa bir sürede güncelledikten sonra kuyruk üzerinde değişiklik yapmak istediği zaman kuyruğa ait olan mutex'i bekler. Yapılan ölçümlerde yöneticinin en çok bu bekleme için vakit harcadığı görülmüştür.

Yığın kuyruk yapısında eş zamanlılık (concurrency) sağlanmadığından kuyrukta yapılan bütün işlemler kilit mekanizmasıyla kontrol edilerek yapılmaktadır. Bu bağlamda yöneticinin işlerinin azaltılması kuyruk yapısının değiştirilmesiyle sağlanabileceği görülmüştür.

Araştırmalar sonucunda, EDMD için önerilen kuyruk yapıları içerisinde önemli görülenler sırasıyla Rapaport, Marin, ve G. Paul tarafından önerilen yapılardır. Rapaport, kullandığı simülasyonda, her molekül için hesaplanan olası bütün olayları bir ikili ağaç yapısına eklemiştir. En erken olay ağacın en solunda yer alan olaydır. Bir olay işlendiğinde, bu olayın moleküllerinin kuyrukta bulunan diğer olayları bulunup silinmeli ve yerlerine yenileri eklenmelidir. Bu işlemi kolaylaştırmak için her molekülün kuyrukta bulunan olayları arasında liste yapıları oluşturulmuştur. Bu liste yapıları sayesinde ekleme çıkarma işlemleri hızlandırılmıştır. Buna rağmen zaman karmaşıklığı $O(\log n)$ olarak kalmaktadır.

Marin'in önerdiği kuyruk yapısı her molekülün sadece olası ilk olayını kuyrukta tutmaktadır. Bu anlamda hafıza kullanımı açısından Rapaport'un önerdiği yapıdan daha iyidir. Diğer taraftan hesaplanan olası birçok olay tekrar hesaplanmak zorunda kalabilir. Bu ise Marin'in yapısı için bir dezavantajdır. Marin, bir tam ikili ağaç yapısını kuyruk yapısı olay tasarlamıştır. Bu yapıda ağacın sadece yaprakları olayları tutmaktadır. Ayrıca her bir düğüm çocuklarının olaylarından en erken olanını göstermektedir. Bu şekilde kök düğümünün sistemdeki en erken olayı göstermesi sağlanmıştır. Marin, her ne kadar kullandığı simülasyonda Rapaport'un veri yapısının performansını iyileştirdiğini söylese de, zaman karmaşıklığı hala $O(\log n)$ olarak hesaplanmıştır.

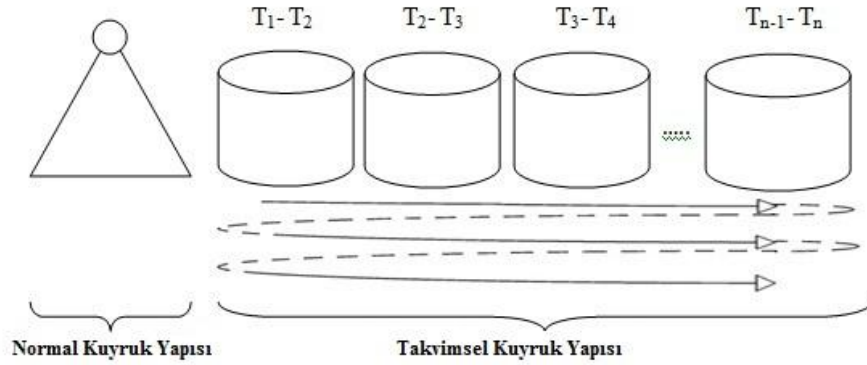
Son olarak literatüre yakın zamanda giren G. Paul'un önerdiği veri yapısı incelendiğinde, hibrit bir yaklaşımın zaman karmaşıklığını $O(1)$ zamana indirdiği gösterilmiştir. G. Paul, bir normal kuyruk yapısı ile bir sınırlandırılmış kuyruk yapısını (bounded priority queue) bir araya getirmiştir. Normal kuyruk yapısı olması beklenen en yakın olaylardan ortalama olarak (çok büyük olmayan) belirli bir sayıda olaydan oluşturulmaktadır. Bu sayede en erken olayın kuyruktan çekilmesi belirli sayıda eleman arasından yapılarak $O(1)$ zamanda yapılabilir. Ayrıca sınırlandırılmış kuyruk yapısı şu şekilde çalışmaktadır. Belirli bir sayıda liste oluşturulur ve bu listelere zaman aralığı verilir. Bu listeler bir kutu (Bucket) gibi düşünülebilir. Bu sayede her bir listede ortalama belirli sayıda eleman bulunması sağlanır. Zaman aralıkları belirli olduğu için yeni hesaplanan bir olayın, eklenmesi gereken doğru liste $O(1)$ zamanda bulunabilir. Listeye ekleme işleminde bir sıralama aranmaz ve listenin sonuna eklenir.

Bu tez çalışmasında, görülen ihtiyaç üzerine literatürde EDMD için önerilen kuyruk yapıları incelenmiş ve G. Paul tarafından önerilen kuyruk yapısı koşut zamanlı erişilebilir şekilde gerçekleştirilmiştir.

Gerçekleştirilen kuyruk uygulamasında, Paul'un önerdiği yapıda bulunan sınırlandırılmış öncelik kuyruğu yerine "Takvimsel Kuyruk" (Calendar Queue) kullanmak hafıza açısından daha verimli olacağı düşünülmüş ve kullanılmıştır. Paul'un sınırlandırılmış kuyruk yapısında her bir kutu (Bucket) normal bir liste olarak tasarlanmışken, takvimsel kuyruk yapısında bu listeler ara bellek için

gerçeklenen kilitsiz sıralı liste yapısı olarak gerçekleştirilmiştir. Bu sayede her bir kutu kendi içerisinde koşut erişime açılmıştır. Normal kuyruk yapısı çok az sayıda eleman barındıracağından yapısal olarak aynı ara bellekte olduğu gibi kilitsiz sıralı liste yapısı kullanılmasında bir sakınca görülmemiştir.

Kutuların zaman aralıkları simülasyon başlangıcında belirlenir. Zaman aralığı değerinin seçimi, normal kuyruğa ortalama olarak ne kadar olayın geleceğini belirleyeceğinden simülasyon performansını direk olarak etkilemektedir. Şekil 4.4 Koşut Erişime Uygun Kuyruk Uygulaması, geliştirilen yapıyı genel olarak anlatmaktadır.



Şekil 4.4 Koşut Erişime Uygun Kuyruk Uygulaması

Bir olayı kuyrukta güncellenmesi sırasıyla (gerekliyse) silme ve ekleme işlemleriyle yapılır. Kutularda bulunan kilitsiz sıralı liste yapısında bir olayın silinmesi o liste üzerinde o elemanın silindi olarak işaretlenmesi anlamına gelir. Bu sayede listeden silme işlemine o an zaman ayrılmaz.

Kutulardan bir tanesi güncel kutu olarak işaretlenir. Bu kutu üzerindeki olaylar normal kuyruktan sonra ilk olabilecek olayları barındırmaktadır. Normal kuyruktaki olaylar işlendiğinde güncel kutuda bulunan olaylardan doğru zaman aralığında bulunan olaylar normal kuyruğa eklenir ve güncel kutu bilgisi bir ilerletilir.

Takvimsel kuyruk yapısının en önemli özelliği takvimde bulunan zaman aralığının bir sonu olmamasıdır. En son kutudan sonra zaman aralığı tekrar ilk

kutudan başlanılarak devam ettirilir. Bir olayın eklenmesi gereken kutu hesaplanırken atlanması gereken zaman adımı sayısının kutunun, kutu miktarına göre modu alınarak (dairese olarak) ekleneceği kutu bulunur. Bu sayede eklenecek olan olayın zaman aralığına ait özel bir kutu olması gerekmez. Bir kutu birden fazla zaman aralığının olayını barındırır. Örnek olarak, kutu sayısının 5 olduğu bir durumda ve zaman adımının 0.1 olduğu bir durumda kutularda olabilecek olası bir durum gösterilmiştir.

1 . Kutu :	13,41	13,44	13,47	13,94				
2 . Kutu :	13,53	13,58		14,02				
3 . Kutu :				14,15	14,17			
4 . Kutu :	13,72	13,79		14,21	14,25	14,28		
5 . Kutu :	13,86	13,89					14,71	14,78

Şekil 4.5 Kutularda olabilecek Olası Bir Durum

Bir olayın kuyruğa eklenmesi için öncelikle normal kuyruğa mı yoksa takvimsel kuyruğa mı ekleneceğine kararı verilir. Takvimsel kuyruk yapısında bir olayın hangi kutuya ekleneceği $O(1)$ zamanda bulunur. Bir olayın bir kutuya eklenme işleminin performansı ortalama olarak bir kutuya düşecek eleman sayısı ile ilişkilidir. Bunun anlamı kutu sayısı ve zaman aralığı seçimi kuyruk yapısının performansını doğrudan belirler.

Geliştirilen kuyruk yapısında, olay ekleme operasyonları için herhangi bir kilit mekanizması konulmamıştır. Ancak en erken olayın normal listeden çekilmesi esnasında eş zamanlı erişime izin verilmemiştir. Bu durum az da olsa bir performans kaybına yol açmaktadır.

Yapılan deneylerde kuyruk yapısının koşut işleme uygun bir şekilde değiştirilmesinin performansı iyileştirdiği görülmüştür.

5 PROGRAMLAMA DETAYLARI

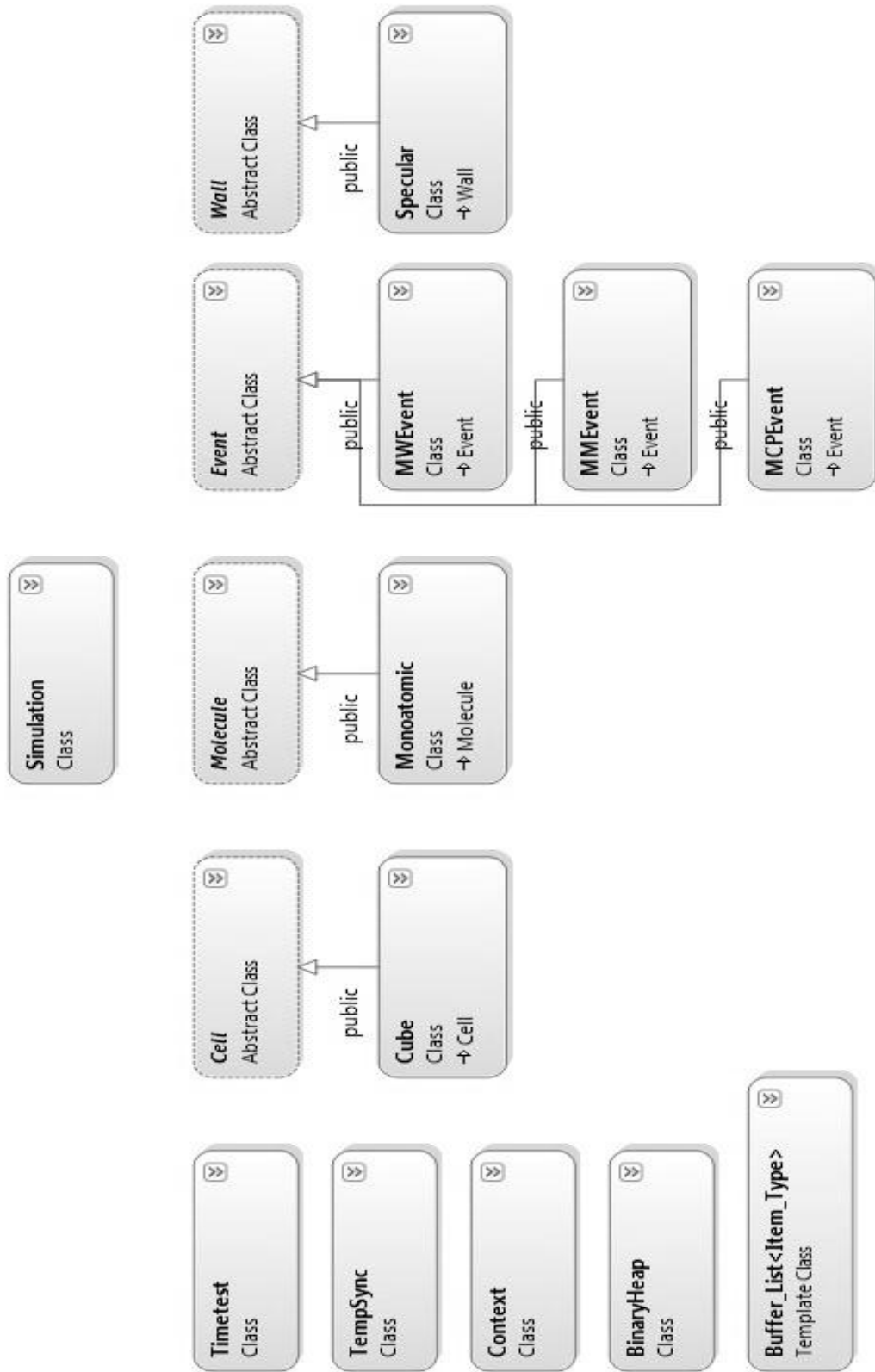
Simülasyon, C++ programlama dili gerçekleştirilmiştir. Ayrıca “*Pthread*” kütüphanesi yardımıyla iş parçacıkları kullanılmıştır. Yüksek çözünürlükte zaman ölçme işlemleri “*PAPI*” kütüphanesi yardımıyla yapılmıştır. Bilinen birçok zaman ölçme kütüphanesinin çok çekirdekli işlemcilerden sorun yaşadığı görülmüş; bunu nedenle doğruluğu kabul edilmiş bir kütüphane kullanılmıştır. Ayrıca zaman ölçme işlemleri ayrı bir sınıfa taşınarak başka bir kütüphane ile de gerçekleştirilme imkânı tanınmıştır.

5.1 Sınıf Diyagramı

Bütün objeler bir simülasyon sınıfı içerisinde oluşturulmuş ve bu sınıfın objesi sayesinde simülasyon gerçekleştirilmiştir. Global olarak bütün sınıfların ihtiyacı olan bilgiler “*Context*” sınıfına konmuş; bu sayede nesnel yapı korunmak istenmiştir.

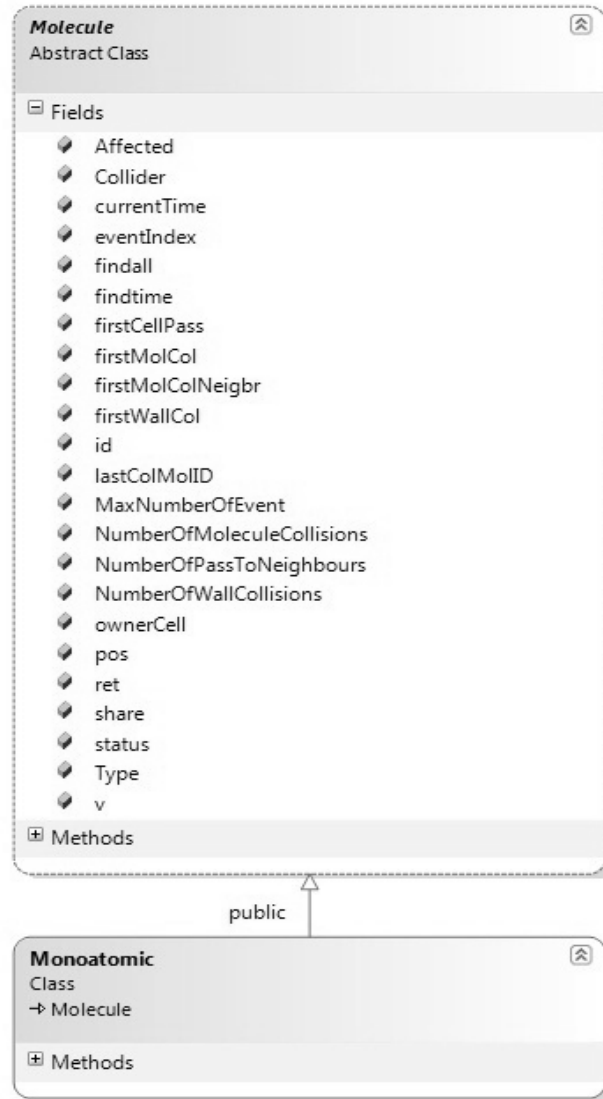
İçerisinde iki olay ve bir atomik değişken barındıran “*TempSync*” isminde bir sınıf oluşturularak, ara bellek bu sınıfın nesnelerinin tipinde bir liste olarak ortaya konmuştur. Bu yapının ilk olayı kuyruktan çekilen olay (orijinal olay), diğer olayı da üzerine işlemlerin yapılacağı kopya olay ve atomik değişkeni ise işaretçi olarak kullanılmaktadır.

Tek tek sınıfların özelliklerini incelemeden önce bir sınıf diyagramına göz atacak olursak, simülasyon ortamı daha iyi anlaşılmış olacaktır.



Şekil 5.1 Sınıf Diyagramı

Molekül sınıfı soyut bir sınıf olarak tasarlanarak tek, çift veya çok atomlu molekül tiplerinin üretilmesine izin verilmiştir. Ayrıca Moleküller ile olaylar arasında çok yakın bir ilişki bulunduğundan birçok defa ikisinin birbirlerine göre durumları değiştirilse de son olarak her molekülün dört adet sabit olaya sahip olması kararlaştırılmıştır. Her molekül sahip olduğu olaylardan bir tanesini kuyrukta bulundurur. Bu sayede bir miktar hafıza kullanımı ile yeni bir olay oluşturma silme işlemlerinin önüne geçilmiş olur.



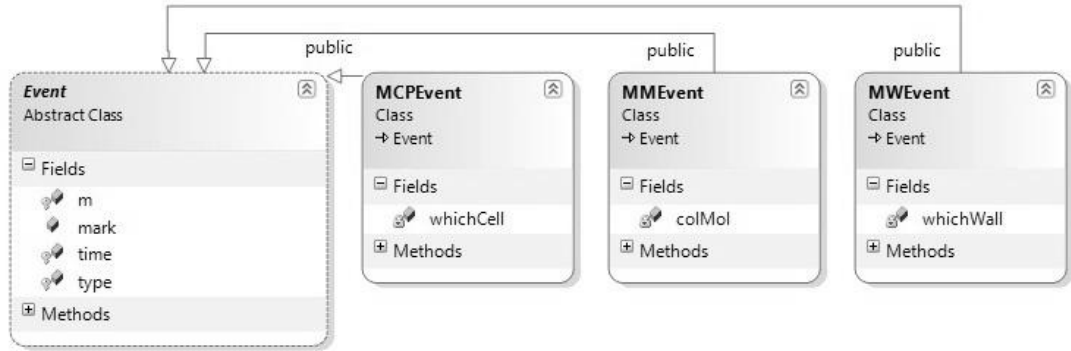
Şekil 5.2 Molekül Sınıfı

Moleküllerin konum ve hız bilgisi bir vektör yapısı oluşturularak tutulmuştur. Vektör yapısı çok basit bir şekilde her boyut için bir hassas sayı tutmaktadır.



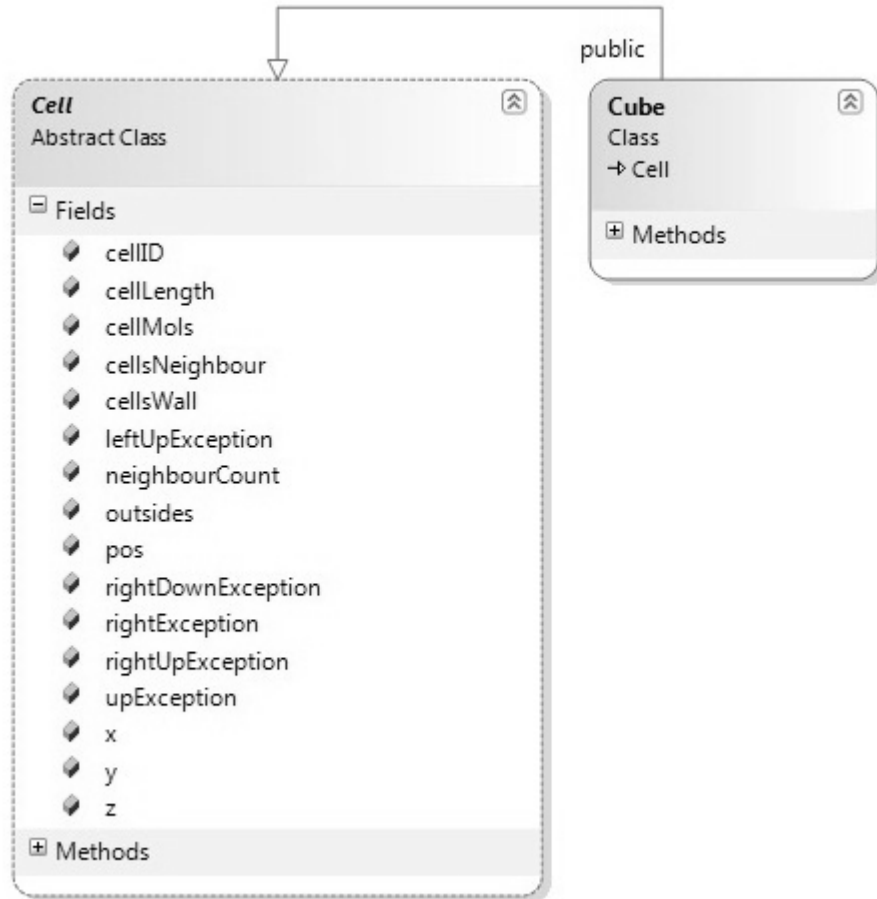
Şekil 5.3 Vektör Yapısı

Olay sınıfı da molekül sınıfı gibi soyut bir sınıftır. Hücre geçiş olayı, duvar çarpışması olayı ve molekül çarpışması olayı bu sınıftan türetilmiştir. Bahsedilen atomik işaret olay sınıfı içerisine eklenmiştir. Türetilen alt sınıflarda özel olarak duvar çarpışması için duvar no'su, hücre geçişi için hücre no'su, molekül çarpışması için molekül id'si bulunmaktadır.



Şekil 5.4 Olay Sınıfı

Hücre sınıfı da molekül ve olay sınıfı gibi soyut bir sınıf olarak tasarlanmıştır. Bu şekilde farklı hücre tüplerinin daha sonra türetilmesine izin verilmek istenmektedir.



Şekil 5.5 Hücre Sınıfı

Zaman ölçüm işlemleri “Timetest” isimli bir sınıfta toplanmış ve bu sayede farklı kütüphaneler kullanılmasına izin verilmiştir. Bu sınıfın en önemli iki metodu işlemci tiklerini sayan metot ile işlemci frekansını ölçen metottur. Bu iki metot sayesinde yüksek çözünürlükte zaman ölçümleri yapılır.



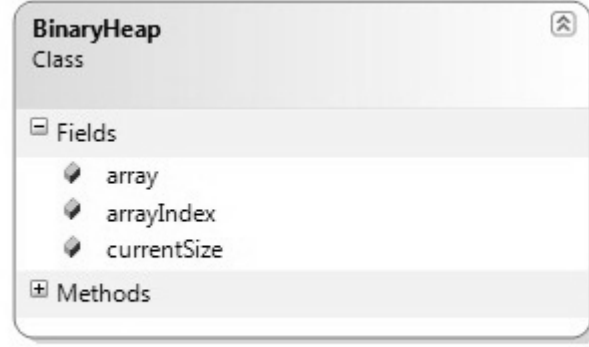
Şekil 5.6 Zaman Ölçümleri İçin Timetest Sınıfı

Bütün sınıflar tarafından erişilmek istenen bilgiler “context” sınıfında toplanmış ve bütün sınıfları başlık dosyalarına eklenmiştir. Bu bilgiler global simülasyon zamanı, toplan çarpışma sayıları gibi değişkenlerdir.



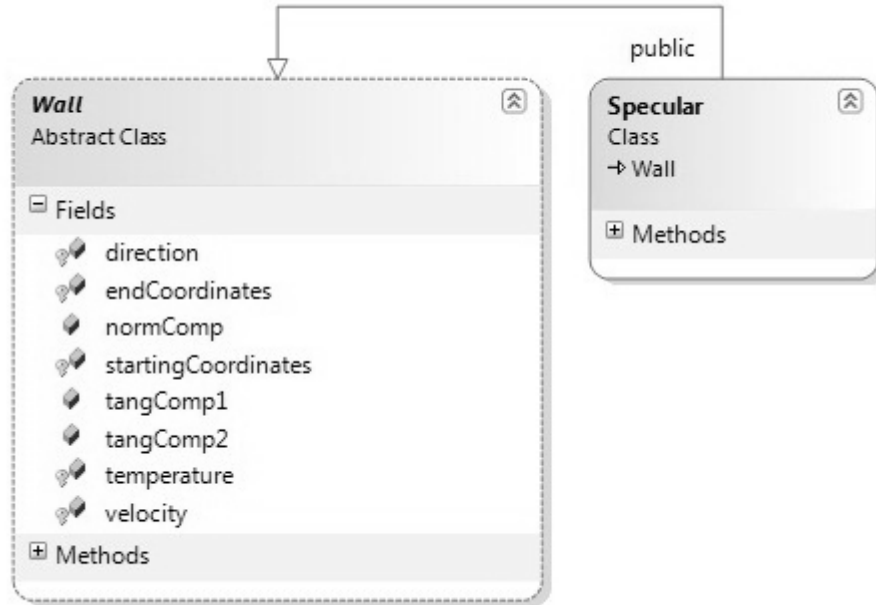
Şekil 5.7 Genel Değişkenler İçin Context Sınıfı

Kuyruk yapısı “BinaryHeap” sınıfında toplanmış bu sayede nesnesel programlama yapısı korunmuştur. Kuyruk yapısı olay göstergesinden (Pointer) oluşan standart vektör yapısındadır.



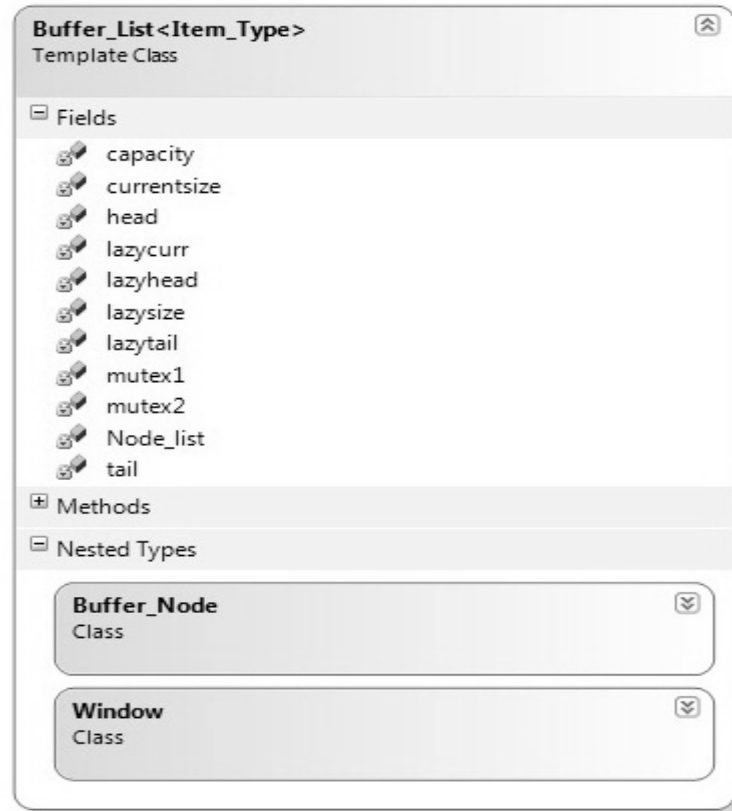
Şekil 5.8 Kuyruk Yapısı

Duvarlar yansıtıcı duvar olarak kabul edilmiştir; ancak molekül duvarı geliştirmesine uygun olması açısından duvar sınıfı soyutlaştırılmıştır.



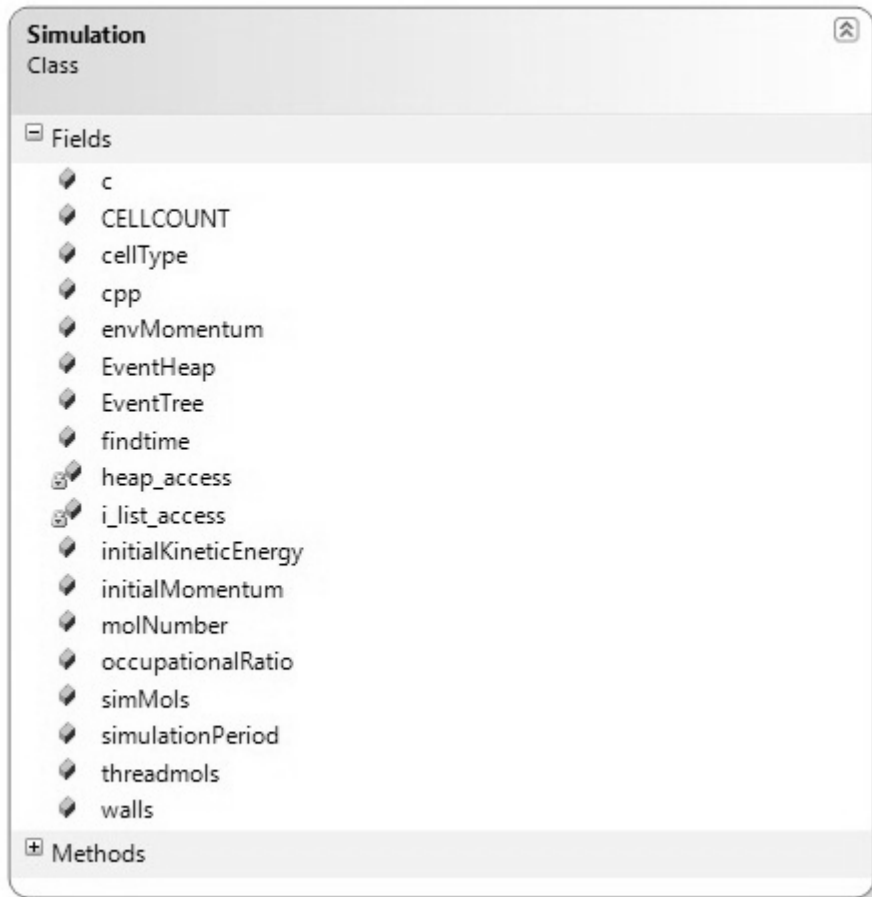
Şekil 5.9 Duvar Sınıfı

Geliştirilen ara bellek yapısının kilitsiz bir liste yapısı olması sağlanmıştır. Bu liste için her hangi bir veri türü belirlenmediğinden bu sınıf bir şablon sınıf şeklinde oluşturulmuştur. İstenildiği takdirde herhangi bir farklı veri tipi için liste oluşturulabilir. Bu sınıf kendi içerisinde iki iç sınıfa sahiptir.



Şekil 5.10 Ara Bellek İçin Geliştirilen Sınıf Yapısı

Son olarak bütün sınıflarımızı barındıran ve simülasyon ortamının oluşturulduğu “Simulation” sınıfı gösterilirse genel hatlarıyla sınıf yapıları anlatılmış olunur. Bu sınıf içerisinde moleküller, hücreler, kuyruk yapısı, duvarlar, liste yapısı ve diğer bütün değişkenler tutulur.



Şekil 5.11 Simülasyon Sınıfı

5.2 İklendirme

Betimlediğimiz ortamda moleküller tek veya çift atomlu olabilmektedir. Ancak, yapılan deneylerde tek atomlu moleküller kullanılmıştır. Ayrıca moleküller katı ve elastik olarak düşünülmüş, her hangi bir kimyasal reaksiyon olmayacağı kabul edilerek deneyler yapılmıştır. Toplam enerji sabit tutulup korunmuş, ortam özellikleri değiştirilmemiştir. Moleküllerin katı olduğu kabulü, çarpışmalar esnasında birbirlerinin içine girmelerini engellemiş ve elastik olmaları parçalanmamalarını sağlamıştır.

Veri setleri 8000 ile 128000 sayıda molekül dağıtılarak oluşturulmuş ve deneyler bu moleküller ile yapılmıştır. Her bir veri seti başta ortama uniform dağıtılmış ve molekül hızları sabit olarak rastgele doğrultularda verilmiştir. Ayrıca yoğunluklar her veri seti için sabit tutulmuştur. Veri setleri deneylerde kullanılmadan önce her molekül için yaklaşık 10 çarpışma yapacak şekilde (8000 molekül için 80000 çarpışma veya 128000 molekül için 1280000 çarpışma) simülasyona sokulmuş; elde edilen simülasyon ortamları kaydedilerek deneyler bu ortamlar üzerinden yapılmıştır.

Simülasyonun işleniş sürecinde birçok parametre mevcuttur. Bu parametreler başlangıç aşamasında bir dosyadan alınır. Simülasyon periyodu, molekül sayısı, hücre boyutu, molekül tipleri ve yarıçapları, simülasyon ortamı koordinatları gibi değerler parametre dosyasından alınırlar. Molekül bilgileri başka bir girdi dosyasından okunur. Moleküllerin pozisyon, hız ve zaman bilgileri bu dosyadan okunur. Ayrıca bir çok kez aynı girdi dosyaları ile çalışıldığından, simülasyon başlangıcında hesaplanan olası olaylardan oluşturulan kuyruk yapısı bir dosyaya kaydedilerek, simülasyonun her çalıştırılması esnasında bu işlemin tekrarı ortadan kaldırılmıştır. Bu sayede bu hesaplamaların tekrar tekrar yapılması engellenmiştir. Sonuç olarak iklendirme işlemi sadece dosyadan veri okuma işlemi haline dönüştürülmüştür.

Simülasyon ortamını oluştururken farklı büyüklükte molekül sayıları için veri setleri oluşturulmuştur. Örneğin, 8000 molekül için $80 \times 80 \times 80$ birim boyutlarında

bir küp oluşturulmuş ve ortam 10000 hücreye bölünmüştür. Bir hücrenin boyutu $8 \times 8 \times 8$ birimdir. Molekül yarıçapları 0,5 birim olarak alınmıştır. Ortam yoğunluğu yaklaşık 0,008 birim olarak kabul edilebilir. Momentumun korunduğu, basınç ve sıcaklığın sabit olduğu varsayılmıştır. Moleküller ilk olarak 1,5 birim hızla, rastgele doğrultulara sahip olarak homojen bir şekilde dağıtılmış, ardından her molekül yaklaşık 10 çarpışma yapacak şekilde simülasyon ilerletilmiş (8000 molekül için 80000 çarpışma gerçekleşmiş), molekül ve olay bilgileri kaydedilerek deneyler kaydedilen veriler kullanılarak gerçekleştirilmiştir.

Seri simülasyonda oluşturulan veri setleri molekül sayısını, hücre sayısını ve ortam büyüklüğünü her seferinde 2 kat artırarak 128000 moleküle kadar veri setleri oluşturulup, aynı işlemler bütün veri setlerine uygulanmıştır. Büyütülen veri setlerinde ortam yoğunluğu ve hücre boyutu sabit tutulmuştur. Anlaşılması açısından 16000 molekül için $160 \times 80 \times 80$ boyutlarında bir ortam da $8 \times 8 \times 8$ boyutlarında 20000 hücre oluşturulmuştur. Denilebilir ki ortamın kare iken dikdörtgen hale gelmesi olay frekanslarını değiştirebilir ancak bu durum göz ardı edilebilecek kadar küçük ölçülmüş bir durumdur.

Her hücre içerisinde kalan moleküllerin hangileri olduğunu bilmektedir ve aynı şekilde her molekül hangi hücrede bulunduğunu bilmektedir.

6 DENEYLER

Deneyler 64-bit 2.66GHz Quad-Core işlemci üzerinde gerçekleştirilmiştir. Kodlamalar standart GCC ile standart optimizasyon ayarları kullanılarak derlenmiştir. Problem büyüklüğü 8000 molekülden 128000 moleküle kadar arttırılmıştır. Tek tip sert moleküller kullanılmış ve her ortamda yoğunluklar sabit tutulmuştur.

Bir çok parametre simülasyon performansını etkilemektedir. Bunlara ara bellek boyutu, işçi sayısı, hücre büyüklüğü, kuyruk yapısı örnek olarak gösterilebilir. Deneylerde özellikle ara bellek boyutu ile işçi sayısı arasındaki ilişki incelenmiştir. (İşçi sayısı 2, 3, 4, 8, 16 ve 24 olarak değiştirilmiş aynı anda ara bellek boyutu 4, 8, 16, 24 olarak değiştirilmiştir.) herhangi bir iyileştirme yapılmadan ilk elde edilen sonuçlar umut vericidir. En iyi hızlanma 2.32 ile 128000 molekülde 4 işçi ve 24 ara bellek boyutu ile elde edilmiştir. Problem büyüklüğü oranında işçi sayısı arttırıldığında performansın iyileştiği görülmüştür. Bunun bir sebebi de problem büyüklüğü arttıkça olaylar arasındaki ilişkilerinde azalmasıdır.

Ara bellek boyutu işçi sayısı ile birlikte belirli bir oranda arttırılmalıdır. Bunun anlamı bir işçinin olay işleyip ekleyebileceği alanın ara bellekte bulunması gerekmektedir. Yinede ara belleğin çok büyük olması çok fazla geçersiz olay işleneceğinden istenmemektedir. İşçi sayısı ile ara bellek arasındaki optimum sayı problem büyüklüğü arttıkça artmaktadır.

Tablo 6.1 Arabellek Boyutu 4 Olduğu Zaman Çalışma Zamanı

İşçi Sayısı	Molekül Sayısı				
	8000	16000	32000	64000	128000
Seri	47,00	97,81	213,92	454,48	965,73
2 İşçi	33,86	72,47	141,73	296,25	624,30
3 İşçi	32,56	63,64	129,36	265,32	531,82
4 İşçi	32,24	65,23	131,57	269,61	538,84
8 İşçi	34,95	69,59	137,66	280,00	569,86
16 İşçi	38,89	74,28	147,40	296,79	604,59

Tablo 6.2 Arabellek Boyutu 8 Olduğu Zaman Çalışma Zamanı

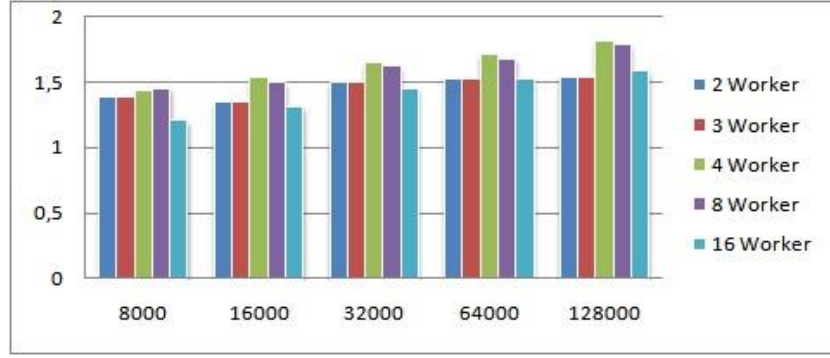
İşçi Sayısı	Molekül Sayısı				
	8000	16000	32000	64000	128000
Seri	47,00	97,81	213,92	454,48	965,73
2 İşçi	31,35	60,64	131,40	284,56	598,01
3 İşçi	32,21	54,61	99,73	204,85	429,36
4 İşçi	32,66	56,16	104,35	211,17	441,50
8 İşçi	36,31	64,07	123,78	248,91	505,07
16 İşçi	40,20	71,94	133,99	273,94	571,92

Tablo 6.3 Arabellek Boyutu 12 Olduğu Zaman Çalışma Zamanı

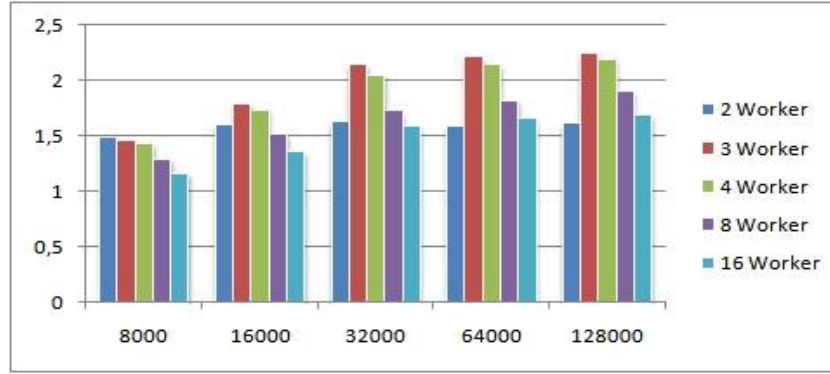
İşçi Sayısı	Molekül Sayısı				
	8000	16000	32000	64000	128000
Seri	47,00	97,81	213,92	454,48	965,73
2 İşçi	42,61	70,56	128,55	283,75	599,68
3 İşçi	43,23	74,56	107,38	196,11	424,83
4 İşçi	43,42	76,01	112,04	202,79	429,02
8 İşçi	44,84	80,82	130,19	234,40	452,35
16 İşçi	47,10	88,13	148,42	262,48	513,02

Tablo 6.4 Arabellek Boyutu 24 Olduğu Zaman Çalışma Zamanı

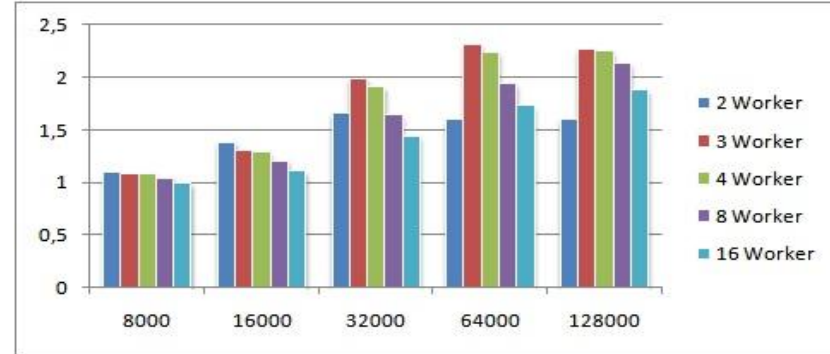
İşçi Sayısı	Molekül Sayısı				
	8000	16000	32000	64000	128000
Seri	47,00	97,81	213,92	454,48	965,73
2 İşçi	47,02	88,60	134,61	289,56	609,55
3 İşçi	47,03	89,91	143,48	198,66	426,53
4 İşçi	47,16	90,93	146,09	216,26	419,41
8 İşçi	48,34	94,02	158,09	253,95	440,48
16 İşçi	50,29	99,57	175,70	292,03	517,98



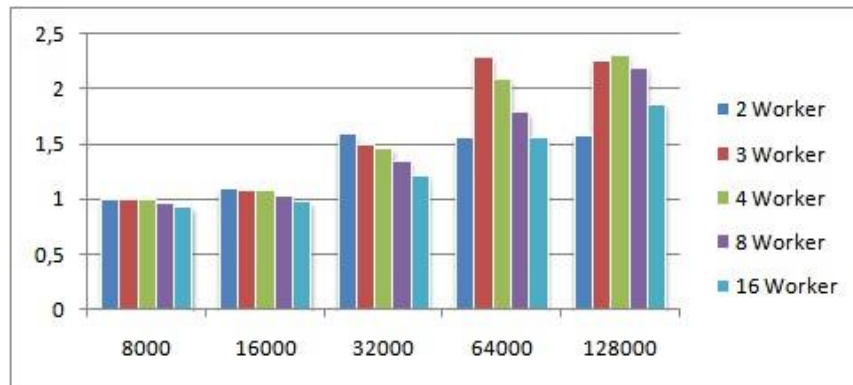
Şekil 6.1 Arabellek Boyutu 4 Olduğu Zaman Hızlanma



Şekil 6.2 Arabellek Boyutu 8 Olduğu Zaman Hızlanma



Şekil 6.3 Arabellek Boyutu 12 Olduğu Zaman Hızlanma



Şekil 6.4 Arabellek Boyutu 24 Olduğu Zaman Hızlanma

7 SONUÇLAR

Bu çalışmada olay bazlı bölümlendirmeye dayanan yeni bir paralel EDMD simülasyonu algoritması önerilmiştir. Algoritmanın genel anlayışı asenkronize olay işleme mekanizması ile kilitsiz ara bellek kullanımı üzerine kurulmuştur.

Yapılan incelemelerde görülmüştür ki, algoritmanın en çok zaman alan kısmı yöneticinin çalışmasıdır. Yığın yapısı kullanıldığında yönetici en çok kuyruk yapısında bulunan kilit mekanizmaları dolayısıyla beklemektedir. Bu nedenle kuyruk yapısı, ara bellek gibi kilitsiz bir yapıya kavuşturulması düşünülmüştür. Alternatif olarak literatürdeki yapılar incelendiğinde G. Paul tarafından önerilen kuyruk yapısının çok uygun olacağı düşünülmüştür (Paul 2007). Bu kuyruk yapısı ekleme çıkarma gibi işlemleri $O(1)$ zaman karmaşıklığında gerçekleştirmektedir. Bu çalışmada gerçekleştirilen kilitsiz liste yapısının kullanımı ile Paul'un kuyruk yapısı koşut işlem ile uyumlu bir hale getirilmiş eş zamanlı erişime izin vermesi sağlanmıştır.

Hücre boyutu ile tipinin paralel algoritma üzerindeki etkileri incelenmelidir. Bu anlamda yakın gelecekte farklı hücre yapıları ile paralel algoritmanın performansı test edilecektir.

Ayrıca alan bölümlendirme yöntemi ile olay bazlı bölümlendirme yaklaşımı hibrit bir yapı ile kullanılabilir. Bu sayede hem dağıtık sistemlerde hem de çok çekirdekli mimarilerde performans sağlayacak bir yapı oluşturulabilir.

Bu çalışma da verilen algoritma ile EDMD simülasyonun performansı iyileştirilmiş ve uygulamada kilit mekanizmalarından kaçınılabileceği gösterilmiştir.

KAYNAKLAR DİZİNİ

1. Herbordt, Martin Christopher, Md. Ashfaquzzaman Khan, and Tony Dean. "Parallel Discrete Event Simulation of Molecular Dynamics Through Event-Based Decomposition." *Proceedings of the 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. Washington DC.: IEEE Computer Society, 2009. 129-136.
2. Khan, Md. Ashfaquzzaman, ve Martin C. Herbordt. «Parallel discrete molecular dynamics simulation with speculation and in-order commitment.» *Journal of Computational Physics* 230, no. 17 (2011): 6563-6582.
3. Marin, Mauricio, and Patricio Cordero. "An empirical assessment of priority queues in event-driven molecular dynamics simulation." *Computer Physics Communications*, 1995: 214-224.
4. Miller, S., and S. Luding. "Event-driven molecular dynamics in parallel." *Journal of Computational Physics*, 2003: 306-316.
5. Paul, Greald. "A Complexity $O(1)$ priority queue for event driven molecular dynamics simulations." *Journal of Computational Physics*, 2007: 615-625.
6. Pöschel, T., and T. Schwager. *Computational granular dynamics: models and algorithms*. Springer, 2005.
7. Rapaport, D. C. *The art of molecular dynamics simulation*. Cambridge Univ Pr., 2004.
8. Rapaport, D. C. "The event scheduling problem in molecular dynamic simulation." *Journal of Computational Physics*, 1980: 184-201.
9. Su, Hung-Chi, Hai Jiang, ve Bin Zhang. «Synchronization on Speculative Parallelization of Many-Particle Collision Simulation.» *World Congress on Engineering and Computer Science*. 2007.