T.C.
AKDENİZ ÜNİVERSİTESİ

ÇELİK ÇERÇEVE SİSTEMLERİN GPU TABANLI OPTİMİZASYONU

Tevfik Oğuz ÖRMECİOĞLU

FEN BİLİMLERİ ENSTİTÜSÜ

İNŞAAT MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

ARALIK 2019

ANTALYA

T.C.
AKDENİZ ÜNİVERSİTESİ

ÇELİK ÇERÇEVE SİSTEMLERİN GPU TABANLI OPTİMİZASYONU

Tevfik Oğuz ÖRMECİOĞLU

FEN BİLİMLERİ ENSTİTÜSÜ

İNŞAAT MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

ARALIK 2019

ANTALYA

T.C.

AKDENİZ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ÇELİK ÇERÇEVE SİSTEMLERİN GPU TABANLI OPTİMİZASYONU

Tevfik Oğuz ÖRMECİOĞLU

İNŞAAT MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

Bu tez 30/12/2019 tarihinde aşağıdaki jüri tarafından Oybirliği ile kabul edilmiştir.

Doç. Dr. İbrahim AYDOĞDU

Doç. Dr. Erkan DOĞAN

Doç. Dr. Ferhat ERDAL

ÖZET

ÇELİK ÇERÇEVE SİSTEMLERİN GPU TABANLI OPTİMİZASYONU

Tevfik Oğuz ÖRMECİOĞLU

Yüksek Lisans Tezi, İnşaat Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. İbrahim AYDOĞDU

Aralık 2019; 54 sayfa

Çerçeve yapı sistemleri, doğrusal yapı elemanlarından oluşan ve inşaat mühendisliğinde en sık kullanılan taşıyıcı sistemlerden birisidir. Son yıllarda bilgisayar ve optimizasyon yöntemlerindeki gelişmeler sayesinde bu yapı sistemlerinin optimizasyonu alanındaki çalışmalar hızla artmaktadır. Çerçeve sistemlerin optimizasyonu problemi karmaşık ve düzensiz yapıda olması sebebiyle bu tür problemlerin çözümünde meta-sezgisel teknikler sıklıkla tercih edilmektedir. Meta-sezgisel yöntemler bu tür karmaşık problemlerin çözümüne imkân sunsa da çok tekrarlı işlemler barındırmaktadır. Bu nedenle çözüm süresi uzayabilmektedir. Buna ek olarak, çerçeve yapı sistemlerin analizinde sıklıkla kullanılan Sonlu Elemanlar Metodunun çözüm kabiliyeti ve çeşitliliği fazla olsa da standart bilgisayarlar ile özellikle büyük ölçekli yapıların çözümünde ciddi hesaplama sürelerine ihtiyaç duyulmaktadır. Bu iki durum birlikte değerlendirildiğinde büyük ölçekli çerçeve sistemlerin optimizasyonu için oldukça fazla süreye ihtiyaç duyulmaktadır. Bu sürelerin makul seviyelere indirilmesi standart bilgisayarlarda da bulunabilen GPU (Graphical Processing Unit) işlemcilerinin paralel programlama ile birlikte çalıştırılmasıyla mümkündür. Bu tez çalışmasında GPU mimarisine uygun şekilde çelik çerçeve sistemleri analiz eden bir tasarım programı geliştirilmiştir. Optimizasyon yöntemi olarak BioCografya Tabanlı Optimizasyon metodu kullanılmıştır. Sonlu elemanlar yöntemi kullanılarak yapılan tüm işlemler GPU paralel programlama yapısı ile oluşturulmuştur. Geliştirilen programının tasarım ve optimizasyon kısımları CPU (Central Processing Unit) mimarisine göre yazılmıştır. Daha sonra, sonuçları karşılaştırmak amacıyla da programın tamamen CPU mimarisinde, paralel işlem barındırmayan ayrı bir sürümü hazırlanmıştır. Her iki sürüme göre iki farklı tasarım örneği oluşturulmuş, bu örnekler değişik yük kombinasyonu ve bilgisayar özelliklerine göre toplam 48 defa optimize edilmiştir. Elde edilen sonuçlara göre FEM işlemlerinde GPU işlemcilerinin CPU işlemcilerine göre daha başarılı oldukları, optimizasyon algoritmasının yeterliliği ortaya konulmuştur.

ANAHTAR KELİMELER: Optimizasyon, Optimum Tasarım, Biyo-Cografya Optimizasyonu, Meta-Sezgisel, FEM, Paralel işlem, Çerçeve Yapılar, GPU, CPU

JÜRİ: Doç. Dr. Öğr. Üyesi İbrahim AYDOĞDU

Doç. Dr. Erkan DOĞAN

Doç. Dr. Ferhat ERDAL

ABSTRACT

GPU BASED OPTIMIZATION OF STEEL FRAME STRUCTURE SYSTEMS

Tevfik Oğuz ÖRMECİOĞLU

MSc Thesis in Civil Engineering

Supervisor: Assoc. Prof. Dr. İbrahim AYDOĞDU

December 2019; 54 pages

Frame construction systems are one of the most frequently used structural systems in structural engineering. In recent years, thanks to advances in computer and optimization methods, studies on optimization of these structural systems are increasing rapidly. Since the optimization of frame systems is complex and irregular, meta-heuristic techniques are often preferred to solve such problems. Meta-heuristic methods allow for the solution of such complex problems, but they contain many repetitive processes . Therefore, the solution time can be extended. In addition, although Finite Element Method (FEM), which is frequently used in the analysis of frame structure systems, has a wide range of solution capability and diversity , serious computing times are required especially for the solution of large scale structures. When these two situations are evaluated together, it takes a lot of time for the optimization of large-scale frame systems. Reducing these times to reasonable levels is possible by running GPU (Graphical Processing Unit) processors, which can be found on standard computers, in parallel with programming. In this thesis, an optimum design program which solves steel frame systems in accordance with GPU architecture is developed. As the optimization method, Biogeography Based Optimization (BBO) method was used. GPU parallel programming structure is used in all FEM method. The design and optimization parts of the developed program are written according to the CPU (Central Processing Unit) architecture. In order to compare, a separate version of the program, which does not include parallel processing in CPU architecture, has been prepared. Two different design examples were created according to both versions, which were optimized 48 times for different load combinations and computer characteristics. According to the results obtained, it is demonstrated that GPU processors are more successful in FEM operations than CPU processors and the efficiency of optimization algorithm is revealed.

KEY WORDS: Optimization, Optimum Design, Meta-heuristic, Bio-Geography Optimization, FEM, Parallel Processing, Frame Structure, GPU, CPU

COMMITEE: Assoc. Prof. Dr. İbrahim AYDOĞDU

Assoc. Prof. Dr. Erkan DOĞAN

Assoc. Prof. Dr. Ferhat ERDAL

ÖNSÖZ

Eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, beni optimizasyon mantığıyla tanıştıran, araştırmanın tüm aşamalarında yardımlarını esirgemeyen, değerli danışman hocam Doç. Dr. İbrahim AYDOĞDU'ya, şükranlarımı sunarım.

Bu tez çalışması esnasında komitede bulunan bütün değerli hocalarıma emekleri için çok teşekkür ederim. Ayrıca CUDA, GCC ve CuBLAS başta olmak üzere uzun yılların emeğini içinde barındıran projelerini karşılıksız olarak akademisyenlerin kullanımına açan bütün yazılım geliştiricilerine ve destekleriyle her zaman yanımda olan aileme teşekkürlerimi sunarım.

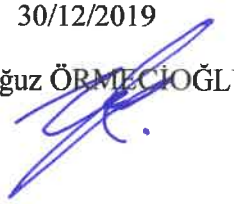
İÇİNDEKİLER

AKADEMİK BEYAN

Yüksek Lisans Tezi olarak sunduğum "Çelik Çerçeve Sistemlerin Gpu Tabanlı Optimizasyonu" adlı bu çalışmanın, akademik kurallar ve etik değerlere uygun olarak bulunduğunu belirtir, bu tez çalışmasında bana ait olmayan tüm bilgilerin kaynağını gösterdiğimi beyan ederim.

30/12/2019

Tevfik Oğuz ÖRMECİOĞLU



SİMGELER VE KISALTMALAR

Simgeler

F_k	:	Düğüm noktaları üzerinde oluşan yüklerin toplamı
k_i	:	i elemanının lokal eksendeki rijitlik matrisi
u_i	:	Lokal eksendeki deplasman vektörü
U_i	:	Global eksendeki deplasman vektörü
f_i	:	Lokal eksende iç tesir vektörleri
F_i	:	Global eksende iç tesir vektörleri
B_i	:	Transformasyon matrisi
K_{sys}	:	Sistem rijitlik matrisi
P_{sys}	:	Sistem yük vektörü
d_{sys}	:	Sistem deplasman vektörü
W	:	Çerçeve sistemin toplam ağırlığı
m_r	:	Her bir yapı elemanının birim ağırlığı
n_y	:	Tüm çerçeve sistemindeki toplam grup sayısı
l_s	:	s elemanının uzunluğu
t_r	:	r grubu içerisindeki toplam eleman sayısı
M_{nx}	:	x-ekseni boyunca ki nominal eğilme dayanımı
M_{ny}	:	y-ekseni boyunca ki nominal eğilme dayanımı
M_{ux}	:	x-ekseni üzerindeki gerekli eğilme dayanımı
M_{uy}	:	y-ekseni üzerindeki gerekli eğilme dayanımı
P_n	:	Nominal eksen dayanımı
P_u	:	Gerekli eksenel dayanım
H	:	Yapı yüksekliği
M_{nx}	:	x-ekseni boyunca ki nominal eğilme dayanımı
$g_{d,j}$	:	Sehim sınırlandırıcı katsayısı
δ_{jl}	:	j numaralı elemanın l yükleme durumu altındaki maksimum sehimi
δ_j^u	:	Çelik çubuğun alabileceği sehim üst limiti
n_{sm}	:	Sehim limitinin uygulandığı elemanların tamamı
n_{lc}	:	Yükleme sayısı
g_{tdj}	:	En üst kat öteleme sınırlandırıcısı
$(\Delta_{top})_{jl}$	:	En üst kattaki j numaralı düğüm noktasının l numaralı yükleme durumu
n_{jtop}	:	En üst kattaki düğüm noktalarının sayısı
g_{idj}	:	Yatay deplasman sınırlandırıcısı
n_{st}	:	Kat sayısı
$(\Delta_{oh})_{jl}$	:	j numaralı kattaki ötelemenin l numaralı yükleme durumundaki değeri
g_{cdi},g_{cmi}	:	Kolon-kolon uyuşum sınırlandırıcısı
n_{cc}	:	n_{cc} çerçeve sistemindeki kolon-kolon birleşimlerinin sayısı
$W_{ic(\text{üst})}$	:	Üst kolonun kesitinin birim ağırlığı
$W_{ic(alt)}$	:	Alt kolonun kesitinin birim ağırlığı

$d_{ic(\text{üst})}$	:	Üst kolonun W kesitinin derinliği
$d_{ic(alt)}$	:	Alt kolonun W kesitinin derinliği
g_{bci}, g_{bbi}	:	Kiriş-kolon uyuşum sınırlandırıcısı
b_{fbk}, b'_{fb}, b_{fc}	:	Kirişe ait flanş ölçüleri
d_{cl}	:	Kolonun derinliği
t_{fl}	:	Flanşın kalınlığı
f_{cost}	:	Maliyet-performans değeri
p_c	:	Yapı maliyet katsayısı
W_y	:	Bütün yapının ağırlığı

Kısaltmalar

BBO	:	Biogeography-Based Optimization (BioCografya Tabanlı Optimizasyon)
HSI	:	Habitat suitability index
SIV	:	Suitability index Variables
CPU	:	Central Processing Unit
GPU	:	Graphical Processing Unit
FEM	:	Finite Element Method
HPC	:	High Performance Computing
BBO	:	Biogeography Based Optimization
SM	:	Streaming Multiprocessor
GP	:	Graphical Pipeline
TSE	:	Türk Standartları Enstütüsü
MP	:	Massively Parallel
Gflops	:	Giga Floating Points Per Second
Tflops	:	Tetra Floating Points Per Second
CUDA	:	Compute Unified Device Architecture
GPGPU	:	General Purpose GPU
SP	:	Single Precision
DP	:	Double Precision
LRFD-AISC	:	Load and Resistance factor design of American Institute of Steel Construction
RC	:	Reinforced Concrete
CFS	:	Cold Formed Steel
LF yada LFS	:	Levy Flights Search (Levy Uçuş Araması)
PSO	:	Partical Swarm Optimization (Parçacık Sürü Optimizasyonu)
NMA	:	Nelder-Mead Algorithm
MFO	:	Moth-Flame Optimization
BA	:	Bat Algorithm
ES	:	Evolution Strategy
ALC yada ALCO	:	Aging Leader and Challengers Optimization
MDM	:	Modified Dolphin Monitoring
RBDO	:	Reliability-Based Design Optimization
GA	:	Genetic Algorithm

GGA	:	Guided Genetic Algorithm
RSM	:	Response Surface Methodology
HS	:	Harmony Search
TLBO	:	Teaching-Learning-Based Optimization
ETE	:	First Explores Then Exploits
BB-BC	:	Big Bang-Big Crunch
FWA	:	FireWorks Algorithm
IFWA	:	Improved FireWorks Algorithm
TSA	:	Tree Seeds Algorithm
GSO	:	Grey Wolf Optimization

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Çelik ülkemiz genelinde özellikle ticari binalar için sıklıkla tercih edilen yapı türlerinden biridir. Malzeme yönünden bakıldığında çelik yapıların betonarme yapılara göre prefabrik olarak kolay üretilebilmesi, taşınması ve birleştirilip yapı haline getirilebilmesi mümkündür. Ayrıca geri dönüşüm konusunda çelik yapılar, betonarme yapılara göre çok daha kolay ve yüksek verimliliğe sahiptirler. Bu sebeplerle çelik yapıların sürdürülebilirliği betonarme yapılara göre daha yüksektir. Aynı zamanda çelik yapılar, homojen olmaları sebebi ile diğer kompozit malzemeler ile üretilmiş yapılara göre, yapı analiz hesaplamalarını daha kolay gerçekleştirebilme avantajı sunmaktadır.

Öte yandan, çelik yapıların eleman sayıları ve eleman grup sayıları artıkça bu yapıların tasarımları ve analizleri oldukça zorlaşmakta; yapı karmaşıklaştıkça bu yapılar için gereken çözüm kombinasyonlarının sayısı da artmaktadır. Bu yapı kombinasyonları arasından, verilen yükleme koşulları altındaki şartları sağlayacak en hafif ve dolayısıyla en ekonomik yapıyı tecrübe ve deneyim yoluyla ya da deneme yanılma metoduyla bulmak oldukça zor; neredeyse imkânsız bir görevdir. Bu nedenle, böyle bir görevin çözümünü bulmak için kullanılabilecek en etkin metotlardan biri optimizasyondur.

Tanım olarak optimizasyon, amacı maksimizasyon ya da minimizasyon olan bir problemin, tanımlı şartlar altında mevcut çözümlerinin arasından en uygun çözümü bulma işlemidir. Literatürdeki optimizasyon tekniklerini birçok çeşitte kategorize etmek mümkündür. Bunlar arasında en sık kullanılanları, türev tabanlı ve meta-sezgisel tabanlı optimizasyon teknikleridir. Türev tabanlı optimizasyon teknikleri genellikle sürekli değişken içeren fonksiyonlarda iyi performans gösterse de; özellikle mühendislik problemlerinde tanımlanan ayrık değişkenli problemlerin çözümlerinde çok başarılı değillerdir. Bu tarz problemlerin çözümünde meta-sezgisel tabanlı optimizasyon teknikleri oldukça etkin performans göstermektedir. Çelik yapılardaki profiller ayrık değişken olarak tanımlanabildiği için meta-sezgisel optimizasyon teknikleri çelik yapıların optimizasyonu için çok daha uygun tekniklerdir.

Seksenli yılların sonlarından itibaren genetik algoritmanın geliştirilmesi ile başlayan meta-sezgisel algoritma tekniklerin kullanımı günümüze kadar gelişimine devam etmiş olup, bugüne kadar birçok meta-sezgisel teknik geliştirilmiştir. Bunlardan bazıları genetik algoritma, parçacık küme optimizasyonu, karınca kolonisi optimizasyonu, yapay arı kolonisi optimizasyonu, çekirge kolonisi optimizasyonu, harmoni arama optimizasyonu, ateş böceği optimizasyonu ve benzeridir.

Meta-sezgisel optimizasyon tekniklerine, en iyi sonucu bulma işlevini genelde doğadan ilham alarak yerine getirmeleri sebebiyle doğadan ilham alan (Natural Inspired) teknikler de denir. Bu tekniklerin arasında en etkin olanlarından birisi de BioCografya tekniğidir. BioCografya optimizasyon tekniği arama mantığı olarak, adalar içerisinde yaşayan doğal habitlardaki türlerin, birbirleri arasında göç alıp verme ve mutasyon hareketlerinden esinlenerek geliştirilmiş bir optimizasyon tekniğidir.

Literatür araştırmalarından da anlaşılmaktadır ki BioCografya optimizasyon tekniği yaklaşık 10 senelik bir geçmişi olmasına rağmen birçok mühendislik probleminin çözümünde başarılı bir performans göstermiştir. Aynı zamanda birçok kafes, çerçeve ve çelik yapıların optimizasyonunda kullanılmış olup kendini ispat etmiş bir tekniktir. Bu sebeple, bu tez çalışmasında da en iyi çözümün bulunması için BioCografya optimizasyon tekniği kullanılmıştır.

Her ne kadar meta-sezgisel teknikler çelik yapıların optimizasyonunda başarılı sonuçlar verse de, meta-sezgisel teknikler aynı zaman da çok tekrarlı işlem barındıran optimizasyon teknikleridir. Bu nedenle çözüm süreleri oldukça uzun olabilmektedir. Bunlara ek olarak, çelik yapıların analizinde çok sık kullanılan Sonlu Elemanlar Yönteminde yapıların eleman sayısı ve düğüm sayısı bakımından büyüklükleri artıkça yapı matrislerinin de boyutları artmaktadır. Örnek vermek gerekirse 500 düğüm noktalı bir yapının çözüm matrisi 9 milyon eleman barındırmaktadır. Bu matrisin Gauss Eliminasyon Metodu ile çözümü ise milyarlarca işlem gerektirmektedir. Bir de bu işlemlerin, meta-sezgisel optimizasyon teknikleri ile binlerce kere daha tekrar edildiği düşünüldüğü zaman çözüme ulaşmak oldukça uzun sürmektedir. Bu da optimizasyon işleminin kullanılabilirliğini düşürmektedir.

Bu süreyi azaltmak için birçok stratejiler kullanılmaktadır. Hesaplama işlemlerinin kısaltılması, yapı matrisinin küçültülmesi, daha iyi kodlama teknikleri kullanılması ya da birçok bilgisayarın birbirleri ile bağlanması yoluyla hız kazanılmaya çalışılması ve paralel çalışan işlemciler kullanılması bunlara örnek verilebilir. Bu sayılan stratejilerden en etkili olan strateji paralel programlama olmuştur. Paralel programlama mantığı 90'lı yılların başlarında gerçek manada işlemci içerisinde doğmuş olsa da, bundan önceki dönemlerde zamanının birçok süper bilgisayarı birden fazla bilgisayarın birbirlerine bir nevi çalışma ağı altyapısı oluşturacak şekilde paralel programlama mantığı ile bağlanmasıyla oluşturulmuştur. Bu sistemde birimler temelde bağımsız bilgisayarlar olmalarına karşın, sonucun tamamıyla ilgilenmeyerek sadece onlara verilen problemin belli parçalarını çözen ve sonucu toplamakla görevli bilgisayara gönderen; böylece kümülatif bir birikimle büyük problemlerin çözülebildiği sistemler olmuşlardır. Bu sistemde bir iş, birçok iş hattına ayrıldıktan sonra her biri bir bilgisayara gönderilip çözümlenerek bütüne ulaşılırken; gerçek paralel yapıya sahip olan modern işlemciler tek bir iş hattı üzerinde, içerilerinde ki birden çok işlemciyi görevlendirerek, aynı anda tüm iş hattını tek sefer de çözüme ulaştırabilmektedir.

İlk üretilen paralel işlemciler, hem sadece belirli işlemleri yapabilme kabiliyetleri ve barındırdıkları işlemci sayılarının günümüze göre çok az olması, hem de programlanmalarının çok karmaşık ve pahalı olmalarından ötürü yaygınlaşamamış; bellek erişim süreleri ve özel donanım ihtiyaçları da bu işlemcileri deneysel işlemciler olmaktan öteye geçirememiştir. 2000'li yıllarda seri üretime geçen ve ilk başlarda sadece CPU işlemcilerine ağır yük bindiren 3B efektlerin çizilmeleri için tasarlan GPU işlemcilerinin yaygınlaşmasıyla gerçek manada paralel işlemcilerinin doğuşu da başlamıştır. Bu işlemciler imal edildikleri yıllarda sadece grafiksel çözümlemeler için amaçlanmış olsalar da bazı bilim insanlarının onları amaçları dışında karmaşık matematiksel matrislerin çözümünde de kullanabileceklerini keşfetmeleri sonucu bu işlemciler hakkındaki çalışmalar hızlanmış, bu olay sadece GPU işlemciler ile çalışan CUDA ve HIPS gibi programlama dillerinin gelişmesine sebep olmuştur.

Günümüzde sırf bilimsel çalışmalar için üretilen ve grafik işleme kabiliyetleri olmayan GPU işlemcileri dahi geliştirilmiştir. Hatta bu durum yeni bir terim olan Genel Amaçlı GPU (GeneralPurposeGPU) teriminin doğmasına sebep olmuştur. GPU işlemcilerini ilk üretilen paralel işlemcilerden ayıran bir özellik de bu işlemcileri günlük amaçlarda kullandığımız ofis bilgisayarı, dizüstü bilgisayarları gibi her bilgisayarın içinde bulunabilir, ucuz bir işlemci olmasıdır. GPU işlemcilerine cep telefonları gibi taşınabilir elektronik aletlerin içinde bile ulaşabilmek mümkündür. GPU işlemcileri sayesinde paralel işlem kabiliyetlerine erişmek için özel yapım süper bilgisayarlara gereksinim kalmamıştır. Böylece, işlem gücü yüksek olan ve sadece belirli kişilerin yararlanabildiği kaynaklarla çözülebilecek problemlerin çözümü artık tüm bilim insanları için mümkün hale gelmiştir.

Literatür incelendiğinde CPU işlemcisi tabanlı çelik çerçeve optimizasyonları çalışmalarına ve birden çok bilgisayarı iş ağına bağlayarak denenen paralel çözüm çalışmalarına rastlansa da GPU kullanılarak tamamen paralel işlem mantığıyla yapılan bir çalışma görülmemiştir. GPU işlemcisinin çelik çerçeve analizinde büyük bir avantaj sağlayacağı tahmin edilmektedir. Bu nedenle bu tez çalışmasında GPU işlemcisi kullanılarak toplam analiz süresi azaltılmaya çalışılmıştır.

Bunun yanı sıra çelik çerçevelerin optimizasyon problemindeki sınırlayıcı fonksiyonlar, LRFD-ASIC şartnamesinde belirtilen mukavemet sınırlayıcıları, yatay öteleme ve katlar arası öteleme sınırlayıcıları ile kolon-kolon ve kolon-kiriş sınırlayıcılarından oluşmuştur.

GPU işlemcisinin performansını ölçmek amacıyla tüm işlemlerin sadece CPU işlemcisiyle yapıldığı ve ayrıca GPU işlemcisi kullanılarak yapıldığı iki ayrı program yazılmıştır. Bu programlar iki farklı büyüklükteki çelik çerçeve yapıyla ve farklı yükleme koşulları altında test edilmiş, daha sonra süreler ölçülüp, karşılaştırılmıştır.

2. KAYNAK TARAMASI

Bu tez çalışmasının ilk aşaması olan literatür taramasında konu ile doğrudan ilgili ve konunun parçaları olan birçok çalışma bulunmuştur. Bulunan çalışmaların belli başlıları, yazılan bu teze ilham kaynağı olmuş, bilgi ve birikimlerinden yararlanılmıştır. Konu ile ilgili ulusal ve uluslararası çalışmalar incelendiğinde çerçeve sistemlerin FEM metodu ile analizi ile ilgili literatürde çok sayıda çalışmaya rastlanmıştır. Ayrıca bu çalışmaların içinde çerçeve yapıların analizinde farklı optimizasyon teknikleri kullanılarak yapılanları da bulunmuştur. Ancak, literatür taramasında çerçeve sistemlerin analizinde FEM metodu ile birlikte optimizasyon tekniklerinin kullanıldığı ve analizlerin GPU desteği ile paralel olarak çözümünün irdelendiği ya da sınandığı bir çalışmaya rastlanmamıştır.

Aydoğdu (2010) çalışmasında üç boyutlu çerçeve sistemlerin karınca kolonisi ve harmonik arama optimizasyon teknikleriyle beraber FEM metodu kullanılarak analiz edilmesine ve bu doğrultuda yazılımsal akış diyagramını hazırlayarak en iyi performans değerlerine ulaşmaya çalışmıştır. Bu tezin hazırlanışında özellikle bu çalışmadan yararlanılmıştır.

Literatür incelendiğinde, 2013 yılına kadar yazılmış çerçeve yapılarla ilgi FEM yönteminden yararlanarak optimizasyon çalışması yapılmasına ilişkin araştırmaların özeti Saka ve Geem (2013) çalışması ile derlenmiştir. Bu yıldan sonra yapılmış çalışmalar içerisinde, Camp ve Huq (2013) betonarme yapıların optimizasyon yardımıyla maliyet ve CO2 emisyonlarının düşürülmesi üzerine bir çalışma yapmışlardır. Kazemzadeh Azad ve Hasancebi (2013) çelik çerçeve yapıların üst sınır stratejisi tekniğini kullanılarak analizinin diğer optimizasyon algoritmalarına göre karşılaştırılmasını, zaman tüketimi çerçevesinde değerlendirilmiştir. Kociecki ve Adeli (2013) , iki aşamalı genetik algoritma vasıtasıyla çelik çerçeve çatı sisteminin analizinin, standart analiz paket programı ile karşılaştırmasını yapmış ve sonuçlarını irdelemişlerdir. Saka ve Geem (2013) çalışmalarında optimizasyon formül yapısının düğüm noktaları koordinatlarının değiştirilmesi sonucu oluşan işlem çözümleri konusuna değinmişlerdir. Erdal ve Saka (2013) ise çalışmalarında 12 ayrı kompozit olmayan kirişin FEM metodu kullanılarak maksimum yük taşıma kapasitesinin bulunması üzerine çalışmışlardır.

Xu, Lu, Guan ve Ren (2014) çalışmalarında kemer köprü ile yüksek katlı bir yapının analiz sonuçlarının GPU tabanlı paralel programlar vasıtasıyla daha başarılı ve hızlı bir biçimde çizilmesi konusunu incelemişlerdir. Hasancebi ve Carbas (2014) çalışmalarında yarasa algoritması kullanarak çözdükleri çelik çerçeve yapının, farklı algoritmalar kullanarak yaptıkları çözümler ile karşılaştırmalarını yapıp performanslarını irdelerken; Dogan (2014) çalışmasında hücresel, moment dayanımlı ve kaynaklı kirişleri avlanma algoritması kullanmış ve optimum tasarımlarının diğer yöntemlere daha başarılı şekil bulunduğu sonucuna ulaşmıştır.

Kociecki ve Adeli (2015) çalışmalarında iki aşamalı Genetik Algoritma kullanarak Kanada'nın Ottawa eyaletindeki iki tren istasyonunun çatısının şekil optimizasyonunu yapıp, en ideal çatı şeklini bulmuşlardır. Turan ve Doğan (2015) ise çalışmalarında kavramsal hidrolojik modellerin kalibrasyonunu av arama, yapay arı kolonisi ve ateş böceği optimizasyon algoritmaları yardımı ile incelemişlerdir.

Aydogdu, Akın ve Saka (2016) çalışmalarında arı kolonisi ve LF dağılımı tekniğini birleştirerek, bu kombinasyonun performansını çelik çerçeveli yapılar üzerinde deneyip, sonuçları sadece arı kolonisi kullanılarak üretilen sonuçlar ile karşılaştırmış ve daha başarılı olduğunu bulmuşlardır. Afzali, Darabi ve Niazkar (2016) çalışmalarında arıların eşleşme ritüelleri üzerine geliştirilen optimizasyon metodu ile çelik çerçeve yapıların minimum ağırlığının bulunması konusunu irdelemişlerdir. Artar (2016) ise deprem yükü etkisi altında çelik yapıların ağırlık optimizasyonu üzerine bir çalışma yapmıştır. Carbas (2016) makalesinde çelik çerçeve yapıları kendi geliştirdiği bir ateş böceği algoritmasıyla çözüp, en hafif yapının bulunması üzerine araştırma yapmışdır. Hadidi, Jasour ve Rafiee (2016) çalışmalarında çelik çerçeve sistemlerde aşamalı çökmeye karşı LRFD-AISC sınırlamaları içerisinde yer alan limitler çerçevesinde yapıyı analiz eden bir genetik algoritma optimizasyon çalışması gerçekleştirirken, Jalili, Hosseinzadeh ve Taghizadieh (2016) ise taşıyıcı sistemlerin analizini kendilerinin modifiye ettiği BBO algoritmasında denemiş ve sonuçlarını yayımlamıştır. Li, Li ve Teng (2016) betonarme bir yapının analizini optimizasyon yapmadan FEM metodu ile GPU üzerinde paralel işlem gücünü kullanarak çözümlemeye çalışmışlar; Maheri, Askarian ve Shojaee (2016) makalelerinde genetik algoritma ve PSO algoritmalarını birleştirerek yapı üzerindeki kirişlerin şekil optimizasyonu üzerinde incelemişlerdir.

Yine 2016 yılı içerisinde Oh, Park, Choi ve Park (2016) tarafından yapılan çalışmada beton dolgulu çelik tüp yapı sistemleri beton dolgulu kare profil tip yapı sistemleri ile optimizasyon kullanarak karşılaştırılmıştır. Pan, Yu, Chen, Luo ve Liu (2016) çalışmalarında yapısal hasar tespiti için ateş böceği ve NMA algoritmalarını birleştirerek optimizasyon denemesi yapmışlardır. Papavasileiou ve Charmpis (2016) ise sismik yükler altında betonarme bir yapının analizinin optimizasyon yardımıyla çözülmesi üzerine çalışmışlardır. Prendes-Gero, Alvarez-Fernandez, Lopez-Gayarre, Drouet ve Junco (2016) araştırmalarında eugenic teorisi kapsamında genetik algoritma kullanarak yapı maliyet optimizasyonu üzerine çalışmışlarken; Rahmzadeh, Ghassemieh, Park ve Abolmaali (2016) çelik yapılarda yatay yüklere dayanıklılık çerçevesinde kullanılan kesmeye dayanım panellerinin FEM metoduyla hesaplamasını yapılarak daha ekonomik hale getirilmesi üzerine çalışmışlardır. Saka, Carbas, Aydogdu ve Akın (2016) makalelerinde çelik uzay kafes sistemlerin optimum tasarımlarını beş farklı optimizasyon tekniğiyle analiz edip, metotlar arasındaki verimliliğin ölçülmesi üzerine bir çalışmışlardır. Wrzesien, Phan, Lim, Lau, Hajirasouliha ve Tan (2016) ise portal çerçeve yapıların optimum tasarım üzerindeki etkisini minimum maliyet optimizasyonu yaparak değerlendirmişlerdir. Yazdi (2016) çalışmasında genetik algoritma ve fuzzy logic tekniğini kullanarak ağırlık optimizasyonu gerçekleştirirken; Ye, Hajirasouliha, Becque ve Pilakoutas (2016) ise sürü algoritması kullanarak CFS kirişlerinde ağırlık optimizasyonunu denemişlerdir.

2017 yılına gelinde optimizasyon konusu ile ilgili literatürde dikkat çekici bir artış görülmüştür. Akbari ve Ayubirad (2017) çalışmalarında zamana göre değişen yükler altında yapının optimum tasarımını GA metodu kullanarak tasarlayan bir uygulama geliştirip sonuçlarını paylaşırken; Amadio, Bedon, Fasan ve Pecce (2017) çalışmalarında sonlu elemanlar yöntemini kullanarak kompozit kirişlerdeki sismik yükler altındaki durumlarını değerlendirmişlerdir. Aydogdu (2017) makalesinde LF ve BBO algoritmalarını birleştirerek betonarme istinat duvarları üzerindeki sismik yükleri analiz etmiş ve sonuçlar içerisinde fiyat analizi yapmıştır. Aynı yıl gerçekleştirdikleri bir diğer

çalışmalarında Aydogdu, Carbas ve Akın (2017) çelik iskelet yapılarda 3 farklı optimizasyon tekniği denemişler ve bu optimizasyon teknikleri içerisinde LF metodunun olumlu etkilerini tespit etmişlerdir. Aydogdu, Efe, Yetkin ve Akın (2017) çalışmalarında çelik yapılarda SSO algoritması kullanarak tasarım optimizasyonu gerçekleştirilmiş ve sonuçlarını paylaşmışlardır. Azad (2017) çalışmasında çelik yapılarda ebat optimizasyonunu birçok optimizasyon metodunu birlikte kullanarak gerçekleştirmiş ve sonuçları karşılaştırmıştır. Çelik yapılar üzerine bir başka çalışmada Balogh ve Vigh (2017) çelik portal yapılarda yangın durumundaki en etkili tasarımın geliştirilmesi için GA kullanarak optimizasyon gerçekleştirmiş ve sonuçları yayımlamışlardır. Barraza, Bojorquez, Fernandez-Gonzalez ve Reyes-Salazar (2017) makalelerinde deprem yükü altında kalan çelik kolonların analizini GA ve PSO kullanarak yapmış ve optimum tasarımı bulmaya çalışmışlardır.

Aynı yıl içerisinde Bojorquez, Ruiz, Ellingwood, Reyes-Salazar ve Bojorquez (2017) yapmış oldukları çalışmalarında ölü yük, yayılı yük ve sismik yükler altında yapının durumunu nöral ağ kullanarak analiz etmişlerdir. Changizi ve Jalalpour (2017) ise çelik yapıların yükler altındaki şekil optimizasyonu üzerine bir çalışma yapmışlardır. Faghihmaleki, Nejati ve Masoumi (2017) çalışmalarında çelik yapılardaki kırılganlık tespiti için nöral ağ metodu kullanmışlar ve sonuçlarını yayımlamışlardır. Gholizadeh, Davoudi ve Fattahi (2017) çalışmalarında MFO algoritması kullanarak çelik yapılarda tasarım işlemi gerçekleştirmişler ve sonuçları raporlamışlardır. Gholizadeh ve Mohammadi (2017) ise sismik yükler altındaki çelik yapılarda BA ve PSO algoritmaları kullanarak tasarım optimizasyonu gerçekleştirmişlerdir. Hasancebi (2017) çalışmasında çok katlı çelik yapıların ES algoritması kullanarak optimizasyonunu gerçekleştirmiş ve sonuçlarını raporlamıştır. Kaveh ve Bolandgerami (2017) çalışmalarında çok elemanlı çelik yapılarda kademeli optimizasyon metotları kullanarak şekil optimizasyonu gerçekleştirmiş ve böylece çok elemanlı çelik yapılarda optimizasyon süresini kısalttığına dair sonuçlarını yayımlamıştır. Kaveh ve Ghazaan (2017) makalelerinde ALC ve PSO algoritmalarını birleştirerek kafes yapıların analizini gerçekleştirmiş ve performans değerlendirmesi yaparken; Kaveh, Vaez ve Hosseini (2017) çalışmalarında CBO algoritmasını MDM metoduyla birleştirerek çelik yapıların optimum tasarımını yeni yaptıkları metot ile tespit etmeye çalışmışlardır. Le, Bui-Vinh, Ho-Huu ve Nguyen-Thoi (2017) çalışmalarında çelik yapıların RBDO metodu kullanılarak optimizasyonunu gerçekleştirmiş ve sonuçları yayımlamışlardır.

Çelik yapılar üzerine 2017 yılı içerisinde gerçekleştirilen diğer bir çalışmada Lu ve Ye (2017) ise GGA algoritması kullanarak stabil olmayan bir zeminde çelik kubbe analizi gerçekleştirmişler ve sonuçları yayımlamışlardır. Maheri, Shokrian ve Narimani (2017) çalışmalarında EHBM algoritması kullanarak çelik çerçeve sistemlerde ağırlık optimizasyonu yaparken; Moradi ve Alam (2017) makalelerinde RSM metodu kullanarak çelik kolon ve kirişlerin bağlantı elemanlarındaki yüklerinin analizini yapıp sonuçları yayımlamışlardır. Pal, Banerjee, Chikermane ve Banerji (2017) çalışmalarında bulonlu çelik yapılarda gevşek bulonların potansiyel yerlerinin optimizasyon yöntemleri ile bulunması üzerine bir araştırma gerçekleştirmişlerdir. Pan, He, Tian, Su ve Zhang (2017) çalışmalarında çoklu optimizasyon teknikleri yerine bölgesel çoklu optimizasyon tekniklerini önermişler ve bu tekniğin üstünlükleri üzerine bir inceleme gerçekleştirmişlerdir. Phan, Lim, Tanyimboh ve Sha (2017) ise çelik çerçeve yapılarda iki çelik kirişin birleşim yerlerinin tam rijit ya da yarı rijit olması durumları altındaki

optimum tasarımlarını GA kullanarak gerçekleştirmişlerdir. Qiao, Han, Zhou ve Li (2017) çalışmalarında sismik yükler altındaki yüksek katlı yapıların şekil optimizasyonu yardımıyla çözümlenmesini incelerken; Truong ve Kim (2017) çelik çerçeve sistemlerin RBDO algoritması ile optimizasyonu üzerine bir çalışma gerçekleştirmişlerdir. Pholdee, Bureerat ve Yildiz (2017) çalışmalarını popülasyon tabanlı, artarak öğrenen, çok hedefli gelişim algoritması kullanarak araçların tavanlarının optimum şekil, doluluk ve genişlik hedefleri çerçevesinde yeniden şekillendirilmesi üzerine yapmışlardır. Erdal, Tunca ve Doğan (2017) oluklu kompozit kirişlerin optimum tasarımının avcı arama optimizasyon tekniği ile bulunması üzerine bir araştırma yapmışlardır. Erdal (2017) ise sinüzoidal açıklıkları olan kirişlerde ateş böceği algoritması kullanarak minimum ağırlık bulunması üzerine bir araştırma yapmıştır.

Daloglu, Artar, Ozgan ve Karakas (2018) çalışmalarında güçlendirilmiş uzay çerçeve sistemlerin optimum tasarımını HS ve TLBO algoritmaları kullanarak bulmuşlardır. Dede (2018) çalışmasında yeni bir optimizasyon tekniği olan Jaya algoritmasını kullanarak çelik ızgara yapıların optimizasyonu üzerine bir araştırma yapmıştır. Dogan, Seker, Saka ve Kozanoglu (2018) avcı arama algoritmasını ile çelik çerçevelerin kolon-kiriş birleşim noktalarını baz alarak minimum ağırlık optimizasyonu araştırması yapmışlardır. Fernandez-Caban ve Masters (2018) çalışmalarında çok elemanlı çelik çerçeve sistemlerin minimum ağırlık optimizasyonu için ETE ve PSO algoritması ile beraber BB-BC algoritmaları kombinasyonunu kullanarak sonuç kümesini küçültmüş ve bu tekniklerin performansı ile ilgili bir araştırma yayınlamışlardır. Gholizadeh ve Milany (2018) çalışmalarında FWA ve IFWA algoritmalarını kullanarak ayrık çelik çerçeve ve makas yapılarının boyut optimizasyonu gerçekleştirmişlerdir.

2019 yılına gelindiğinde ise Bybordiani ve Azad (2019) çelik çerçeve sistemlerin optimum tasarım değerlerini yapı altındaki zemin yapısını da göz önünde bulundurarak tespit etmek amacıyla BB-BC optimizasyon algoritmasını kullanmış ve araştırma sonuçlarını paylaşmışlardır. Boscardin, Yepes ve Kripka (2019) çalışmalarında betonarme yapıların HS metodu ile boyut optimizasyonu gerçekleştirirken, aynı zamanda analiz içerisinde sadece kolonları gruplayarak işlemi tamamlamışlar ve bu gruplama tekniği sayesinde elde ettikleri sonuçları raporlamışlardır. Chen, Yousif, Najem, Abavisani, Pham, Wakil, Mohamad ve Khorami (2019) ise çerçeve ve betonarme plak yapıların maliyetini GA algoritması kullanarak optimize etmişlerdir. Ding, Li, Hao ve Lu (2019) çalışmalarında TSA algoritmasını kullanarak yapısal hasar tespitinin gerçekleştirilmesi üzerine araştırma yapmışlardır. Karimi ve Vaez (2019) çelik çerçeve sistemlerin sismik optimizasyonunu PSO ve GWO optimizasyon tekniklerini kullanarak analiz etmişlerdir. Shallan, Maaly, Sagiroglu ve Hamdy (2019) ise çalışmalarında BBO ve GA algoritmalarını birleştirerek sabit, yarı sabit ve menteşeli kolon-kiriş bağlantılı uzay çelik çerçeve sistemlerin de boyut optimizasyonu yapmışlardır. Shen, Nagai ve Gao (2019) yapılarda kolon boyutlarının seçimi üzerine GA kullanarak yapılması ile ilgili bir araştırma yapmışlardır.

İncelenen yayınlara bakıldığında konuyla ilgili çalışmaların başlangıcının 1988 yılında Goldberg ve Holland (1988) tarafından literatüre kazandırılan genetik algortima çalışmaları ile başladığı anlaşılmıştır. Sonraki yıllarda optimizasyon ile ilgili farklı bilim dallarında bir çok çalışma yapılmış olsa da en etkili olanlarından biri Eberhart ve Kennedy (1995) makaleleri ile detaylarını yayınladıkları parçacık sürü kolonisi çalışmaları

olmuştur. İkili, bu metot sayesinde ileride birçok yeni sürü tabanlı optimizasyon metotlarının da bulunmasına öncülük etmişlerdir.

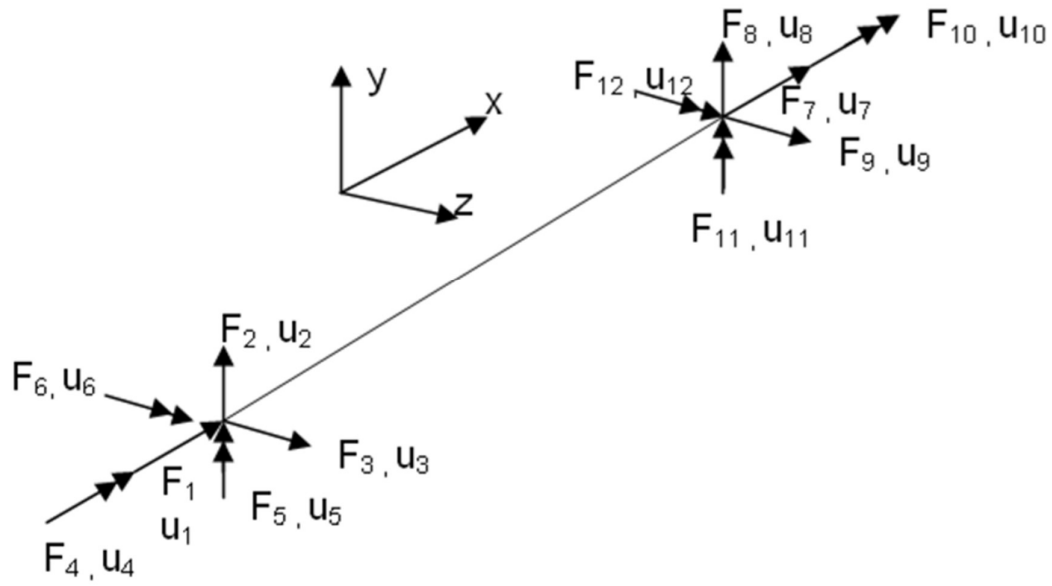
1990'lı yıllardan günümüze kadar geçen süre içerisinde bilgisayar teknolojilerinin gelişmesi ile beraber birçok yeni meta-sezgisel optimizasyon metodunun bulunduğu görülmüştür. Bu süre zarfı içerisinde bu metotların birleşimlerinin de çözümlerde uygulanmasının yaygınlaştığı görülmektedir. Ayrıca literatürde, optimizasyon metotları ile çözümlenen yapı tiplerine baktığımızda daha sıklıkla betonarme değil çelik yapı tiplerinin kullanıldığı tespit edilmiştir. Bunun yanı sıra çalışmalarda, genetik algoritma yerine parçacık sürü optimizasyon tekniklerinin daha çok kullanıldıkları görülmüştür. Ancak literatür taraması esnasında incelenen çalışmaların çoğunluğunun CPU işlemci tabanlı bilgisayarlarda paralel iş gücü kullanılmadan yapılabilecek şekilde dizayn edildiği görülmüştür; bu nedenle söz konusu çalışmaların analiz sürelerinin uzun olduğu anlaşılmıştır. Tezin konusunu oluşturan paralel işgücü alanında ise yeterli çalışmaya rastlanamamıştır.

3. MATERYAL VE METOT

3.1. Çerçeve Sistemlerinin Analizi

Çerçeve sistemler kolon ve kiriş gibi tek boyutlu yapı elemanlarından oluşan inşaat mühendisliğinde en sık kullanılan taşıyıcı sistemlerden biridir. Çerçeve sistemlerin temelde birden çok kiriş ve kolondan oluşması, bu taşıyıcı sisteminin analizi esnasında dış etki ve deplasman arasındaki ilişkinin doğrusal denklem sistemi şeklinde tanımlanmasına imkân vermiştir. Bu tip yapılan çözüm metotlarının genel adı Sonlu Elemanlar Metodu, İngilizce terim adı ise FEM'dir (Finite Element Method). Sonlu elemanlar metodu ile analizler esnasında sıralı olarak birçok doğrusal denklemin çözümü gerekmektedir. Bu denklemler kendi içerisinde adımlardan oluşmakta ve bu adımlar da birçok aritmetiksel işlemlerden oluşmaktadır. Problemi sonlu sayıda ki birçok küçük parçaya bölüp çözme metodolojisinin sadece yapı mühendislerince değil, birçok farklı bilimde hatta ekonomi dalında bile kullanımları görülmektedir.

Çerçeve sistemlerde temel düşünce her bir elemana etki eden yükleri ve bu yükler doğrultusunda oluşan deplasmanları belirlemek ve parçadan bütüne ulaşarak sonuca varmaktır. Bu yaklaşımı incelediğimizde, sistemin tek bir elemanı üzerinde oluşan yük ve deplasmanlar 3 boyutlu eksende Şekil 3.1'deki gibi olmaktadır.



Şekil 3.1. Bir çubuk üzerindeki düğüm uçlarında oluşan kuvvetler, momentler ve deplasmanlar (Aydoğdu 2010)

Burada $\{F_k\} = \{F_1, F_2, F_3, F_4, F_5, F_6\}$ çubuğun ilk düğüm noktasında oluşan yüklerin vektörü, $\{u_k\} = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ ise bu düğüm noktasında oluşan deplasmanların vektörüdür. F_1, F_2 ve F_3 eksenel yükleri temsil ederken, F_4, F_5 ve F_6 ise

momentleri temsil etmektedir. Sistemdeki her bir elemanın uç noktalarındaki yükler ile yine uç noktalarındaki deplasmanlar arasındaki bağlantı aşağıdaki gibidir.

$$\{f_i\} = [k_i]\{u_i\} \tag{3.1}$$

Burada $[k_i]$ i elemanının lokal eksendeki rijitlik matrisini ifade eder. Rijitlik matrisi 3B (3 Boyutlu) sistemler için 12 kolon ve 12 satırdan oluşan bir matristir. Bu matrisin herhangi bir satırındaki elemanların değerleri, ilgili satıra bir birim deplasman verilmesiyle elde dilen kesit tesirlerine eşittir. Bu işlemin tüm tabloya uygulanmasıyla Şekil 3.2'deki rijitlik matrisi elde edilir (Aydogdu 2010).

Çerçeve sistemin çözümü için öncelikle her elemanın lokal rijitlik matrislerinin belirlenmesi gerekmektedir; daha sonrada bu değerler lokalden global'e dönüştürülmelidir. Böylece, sistemin tüm elemanlarının lokal yer değiştirmeleri ile global yer değiştirmeleri arasındaki bağlantılar yardımıyla dönüşüm matrisi elde edilir.

$$[k]=\begin{bmatrix}
\frac{EA}{\ell} & 0 & 0 & 0 & 0 & 0 & -\frac{EA}{\ell} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{12EI_z}{\ell^3} & 0 & 0 & 0 & \frac{6EI_z}{\ell^2} & 0 & -\frac{12EI_z}{\ell^3} & 0 & 0 & 0 & \frac{6EI_z}{\ell^2} \\
0 & 0 & \frac{12EI_y}{\ell^3} & 0 & -\frac{6EI_y}{\ell^2} & 0 & 0 & 0 & -\frac{12EI_y}{\ell^3} & 0 & -\frac{6EI_y}{\ell^2} & 0 \\
0 & 0 & 0 & \frac{GJ}{\ell} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{\ell} & 0 & 0 \\
0 & 0 & -\frac{6EI_y}{\ell^2} & 0 & \frac{4EI_y}{\ell} & 0 & 0 & 0 & \frac{6EI_y}{\ell^2} & 0 & \frac{2EI_y}{\ell} & 0 \\
0 & \frac{6EI_z}{\ell^2} & 0 & 0 & 0 & \frac{4EI_z}{\ell} & 0 & -\frac{6EI_z}{\ell^2} & 0 & 0 & 0 & \frac{2EI_z}{\ell} \\
-\frac{EA}{\ell} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{\ell} & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{12EI_z}{\ell^3} & 0 & 0 & 0 & -\frac{6EI_z}{\ell^2} & 0 & \frac{12EI_z}{\ell^3} & 0 & 0 & 0 & -\frac{6EI_z}{\ell^2} \\
0 & 0 & -\frac{12EI_y}{\ell^3} & 0 & \frac{6EI_y}{\ell^2} & 0 & 0 & 0 & \frac{12EI_y}{\ell^3} & 0 & \frac{6EI_y}{\ell^2} & 0 \\
0 & 0 & 0 & -\frac{GJ}{\ell} & 0 & 0 & 0 & 0 & 0 & \frac{GJ}{\ell} & 0 & 0 \\
0 & 0 & -\frac{6EI_y}{\ell^2} & 0 & \frac{2EI_y}{\ell} & 0 & 0 & 0 & \frac{6EI_y}{\ell^2} & 0 & \frac{4EI_y}{\ell} & 0 \\
0 & \frac{6EI_z}{\ell^2} & 0 & 0 & 0 & \frac{2EI_z}{\ell} & 0 & -\frac{6EI_z}{\ell^2} & 0 & 0 & 0 & \frac{4EI_z}{\ell}
\end{bmatrix}$$

Şekil 3.2. Lokal eksendeki rijitlik matrisi (Aydoğdu 2010)

Düğüm noktalarındaki lokal deplasman vektörü ile global deplasman vektörü arasındaki bağlantı aşağıdaki gibi gösterilebilir. Benzer şekilde lokal ve global eksendeki kesit tesirleri arasındaki ilişki de denklem (3.3)'teki gibi gösterildiği gibidir.

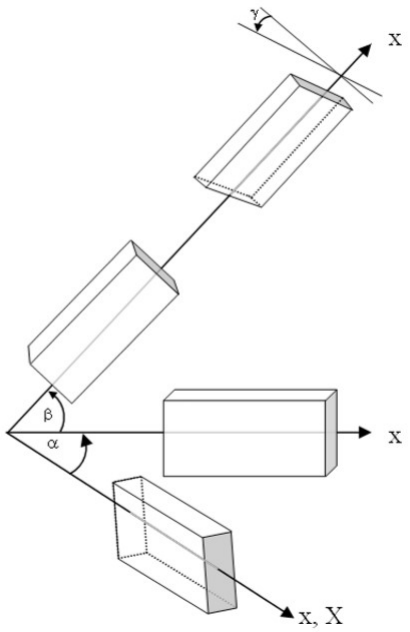
$$\{u_i\} = [B_i]\{U_i\} \tag{3.2}$$

$$\{f_i\} = [B_i]\{F_i\} \tag{3.3}$$

Burada, $\{u_i\}$ ve $\{U_i\}$ lokal ve global eksendeki deplasman vektörlerini; $\{f_i\}$ ve $\{F_i\}$ lokal ve global eksendeki iç kuvvet vektörlerini temsil etmektedir. $[B_i]$ ise global ve lokal eksenler arası dönüşüm matrisi diğer adıyla koordinat transformasyon matrisini temsil etmektedir. Her üç eksendeki koordinat transformasyon matrislerini bir arada çarpılması sonucu denklem (3.4)'de ki transformasyon matrisi elde edilir.

$$[B] = \begin{bmatrix} \cos\beta \cdot \cos\alpha & \sin\beta & -\cos\beta \cdot \sin\alpha \\ \sin\alpha \cdot \sin\gamma - \cos\alpha \cdot \sin\beta \cdot \cos\gamma & \cos\beta \cdot \cos\gamma & \cos\alpha \cdot \sin\gamma + \sin\alpha \cdot \sin\beta \cdot \cos\gamma \\ \sin\alpha \cdot \cos\gamma + \cos\alpha \cdot \sin\beta \cdot \sin\gamma & -\cos\beta \cdot \sin\gamma & \cos\alpha \cdot \cos\gamma - \sin\alpha \cdot \sin\beta \cdot \sin\gamma \end{bmatrix} \tag{3.4}$$

Bir düğüm noktasına ait transformasyon matrisi içerisindeki $\alpha, \beta\ ve\ \gamma$ açılarının görsel temsili Şekil 3.3 içerisinde gerçekleştirilmiştir.



Şekil 3.3. Lokal eksen ile global eksen arasındaki koordinat transformasyonu (Aydogdu 2010)

Bu noktaya kadar anlatılan koordinat transformasyon matrisi sadece çerçeve elemanının bir noktasını temsil etmektedir. Çerçeve elemanının her iki ucunda tüm serbestlik derecelerinde temsil eden koordinat transformasyon matrisi Şekil 3.4'de gösterilmişti.

$$[B] = \begin{bmatrix} b_{11} & b_{12} & b_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_{21} & b_{22} & b_{23} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_{31} & b_{32} & b_{33} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b_{11} & b_{12} & b_{13} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b_{21} & b_{22} & b_{23} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b_{31} & b_{32} & b_{33} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_{11} & b_{12} & b_{13} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_{21} & b_{22} & b_{23} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_{31} & b_{32} & b_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{11} & b_{12} & b_{13} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{21} & b_{22} & b_{23} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Şekil 3.4. Çerçeve elemanının her iki ucunu da tüm serbestlik derecelerinde temsil eden koordinat transformasyon matrisi (Aydoğdu 2010)

Buradaki $b_{i,j}$, $i = 1,2,3$ ve $j = 1,2,3$ Şekil 3.4'de gösterilen çerçeve elemanının koordinat transformasyon matrisi içerisindeki yerlerini temsil etmektedir. Koordinat transformasyon matrisi ortogonal bir matristir bu nedenle matrisin tersi transpozuna eşittir. (3.1)-(3.3) numaraları denklemler yardımıyla herhangi bir çerçeve elemanın rijitlik matrisi lokal eksenden global eksene aşağıdaki şekilde dönüştürülür.

$$(3.1)\rightarrow(3.3)\rightarrow[B_i]\{F_i\} = [k_i]\{u_i\} \tag{3.5}$$

$$[B_i]^T \cdot (3.5)\rightarrow\{F_i\} = [B_i]^T[\,k_i]\{u_i\} \tag{3.6}$$

$$\begin{aligned}(3.6)\rightarrow(3.2)\rightarrow\{F_i\} &= [B_i]^T[k_i][B_i]\{U_i\} \quad = [K_i]\{U_i\} \rightarrow [K_i] \\ &= [B_i]^T[\,k_i][B_i]\end{aligned} \tag{3.7}$$

Yukarıda elde edilen bağıntı ile tüm çerçeve elemanların rijitlik matrisleri lokal koordinatlardan global koordinatlara dönüştürülür. Bu aşamadan sonra eleman rijitlik matrisleri serbestlik derecelerine göre toplanarak sistem rijitlik matrisi (K_{sys}) aşağıdaki şekilde elde edilir.

$$\left[K_{sys}\right] = \sum_{i=1}^{m}([B_i]^T[k_i][B_i]) \tag{3.8}$$

Burada m, çerçeve sistemindeki toplam taşıyıcı eleman sayısını temsil etmektedir. Bu aşamadan sonra sistem yük vektörü (P_{sys}) elde edilir. P_{sys} yapı sistemine etki eden dış yüklerinin yapının serbestlik derecelerine karşılığıdır. Yük doğrudan yapının düğüm noktasına ekti ederse dış yük P_{sys} vektörüne eklenir. Eğer doğrudan düğüm noktasına etki etmeyen yük var ise bu yükler eleman uç noktalarına bir başka değişle düğüm noktalarına aktarılır. Bu işlem aşağıdaki ifade ile gösterilebilir.

$$\begin{aligned}&Eğer\ i \in Düğüm\ ise\ \left\{P_{sys}\right\} = \left\{P_{sys}\right\} + \{P_i\} \qquad i = 1,2,\ldots m \\ &Eğer\ i \notin Düğüm\ ise\ \{P_i\} \rightarrow \{YK_i\}; \left\{P_{sys}\right\} = \left\{P_{sys}\right\} + \{YK_i\}\end{aligned} \tag{3.9}$$

Bu aşamadan sonra rijitlik ve yük ifadeleri sistem uzayına aktarılmıştır. Denklem (3.1)'deki kuvvet ile deplasman arasındaki bağıntı lokal eksen, global eksen veya sistem düzeyinden bağımsızdır. Bu nedenle denklemi sistem düzeyine uyarlayarak aşağıdaki denklem elde edilir.

$$\{P_{sys}\} = [K_{sys}]\{d_{sys}\} \tag{3.10}$$

Yukarıdaki denklemde $\{d_{sys}\}$ yapı sisteminde her bir serbestlik derecesine gelen deplasman ve dönme değerlerini veren sistem deplasman vektörüdür. Denklem (3.10) ifadesi doğrusal denklem sistemini temsil etmektedir. Böylece Gauss Eliminasyon gibi sayısal analiz teknikleri ile denklem (3.10) çözülerek $\{d_{sys}\}$ vektörü elde edilir.

Sistem deplasmanları bulunduktan sonra $\{d_{sys}\}$ vektörü her bir çerçeve elemanına aktarılır ve her bir elemanın global koordinatlardaki deplasman vektörü elde edilir ($\{U_i\}$). Aktarma işlemi yine serbestlik dercesine göre yapılır ancak işlem yönü sistemden global eksene doğrudur.

Bu aşamadan sonra çerçeve elemanlarında oluşan kesit tesirleri hesaplanır. Her çerçeve elemanı birbirinden bağımsız şekilde tasarlandığı için kesit tesirleri lokal koordinatlarda ($\{f_i\}$) hesaplanır. $\{U_i\}$ ve $\{f_i\}$ arasındaki bağıntı rijitlik matrislerin dönüşüm bağıntısının elde edilmesinde kullanılan (3.1)-(3.3) numaraları denklemler yardımıyla aşağıdaki gibi elde edilir.

$$\{f_i\} = [k_i]\{u_i\} \rightarrow \{f_i\} = [k_{\ i}][B_i]\{U_i\} \tag{3.11}$$

Eğer düğüm noktasına etki etmeyen yükler var ise denklem (3.9)'daki bağıntıda elde edilen YK_i ifadesi lokal koordinatlara dönüştürülür (yk_i) çubuk iç tesiri değerlerinden çıkarılır.

$$Eğer\ i \notin Düğüm\ ise\ \{YK_i\} \rightarrow \{yk_i\}; \{f_i\} = \{f_i\} - \{yk_i\} \quad i = 1,2,\dots m \tag{3.12}$$

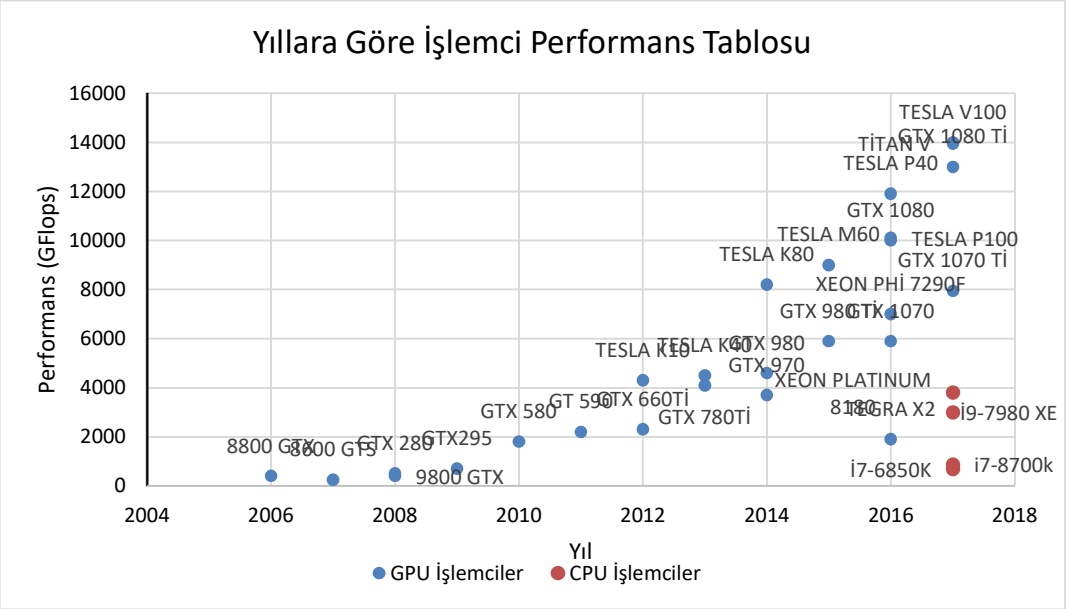
3.2. GPU Kavramı

GPU (Graphical Processing Unit) işlemcileri orijinalde sadece poligonlama, yansıma, aydınlatma ve gölgeleme gibi bir dizi seri grafiksel işlem diziliminin hesaplanması için özel olarak tasarlanmış işlemcilerdir. Bu şekilde sıralı olan ve sürekli tekrar eden bu işlem dizilimine GP (Graphics Pipeline), türkçesi Grafiksel İş Hattı denilmektedir. Bu işlemcilerin yıllar içerisindeki tasarımları daha esnek programlanabilme özelliklerine doğru evirilmiştir. Modern GPU işlemcileri, matris şeklinde yan yana dizilmiş birçok SM (Streaming Multiprocessor) işlemcilerinden, türkçesi çoklu akış işlemcilerinden oluşmaktadır. Bu mimari yaklaşım, GPU işlemcilerini ingilizcesi "High Performance Computing" (HPC) olan ve Yüksek Performanslı Hesaplama adıyla anılan ve çok yoğun tekrarlı ve çok veri içeren hesaplamalar için elverişli hale getirmiştir. Çizelge 1.1 de görülen Tesla P100 serisi özel GPU sunucu hesaplama işlemcisi, aynı aileden olan diğer işlemcilerdeki grafiksel işlem ünitelerinden arındırılmış ve özellikle HPC hesaplamaları için tasarlanmış bir işlemcidir.

Bu tezde kullanılan Geforce GTX 1080 Titan işlemcisi, içerisi içerisinde video çözümleme, OPENGL, Vulkan ve DriectX gibi grafiksel işlem dizilimleri olan normal bir masaüstü grafik işlemcisidir. Bu özelliklerinin yanında, bu işlemci toplamda 3584 CUDA çekirdeği ve 28 SM den oluşmuştur. Her bir SM, 128 CUDA çekirdeği içerirken tek bir SM içerisindeki tüm çekirdekler aynı komutları çalıştıracak bütünleşik paralel bir yapı olan MP (Massively Parallel) tipinde imar edilmiştir. Bu tez için, ana deney bilgisayarı olarak Geforce GTX 1080ti işlemcisi ile birlikte 6 çekirdekli i7-8700K CPU işlemcili bir masaüstü bilgisayar kullanılmıştır. Bu bilgisayarın yanında Geforce GT 730M GPU işlemcisi ile birlikte 4 çekirdekli I7-3630QM CPU işlemcili bir dizüstü bilgisayar, Geforce GTX 950M grafik işlemcisi ile 4 çekirdekli I7-6700HQ CPU işlemcili bir dizüstü ve Geforce GTX 1050 Ti GPU işlemcilisi ile I7-7700HQ CPU işlemcili bir dizüstü bilgisayarları da birbirlerinden bağımsız bir biçimde kullanılmıştır. Çizelge 3.1'de bu işlemciler ile birlikte piyasada bulunabilecek üst ve alt sınıftan diğer işlemcilerin üretim tarihleri, teknik özellikleri ve çeşitli açılardan performansları çizelge ve şekil halinde gösterilmiştir.

Çizelge 3.1. Bazı seçilmiş GPU ve CPU işlemcilerin özellikleri

Denek adı	Tipi	İşlemci	Çekirdek sayısı	Çekirdek frekansı	DP performansı	SP performansı	Hat genişliği (GB/s)
A	Masaüstü	I7-8700K	6	3700 Mhz	32.11 Gflops	61.41 Gflops	8
A	Masaüstü	Geforce GTX 1080.Ti	3584	1481 Mhz	11.34 Tflops	354.4 Gflops	484,4
B	Dizüstü	I7-7700HQ	4	3800 Mhz	43.07 Gflops	33.34 Gflops	8
B	Dizüstü	Geforce GTX 1050.Ti	768	1392 Mhz	2.138 Tflops	66.82 Gflops	112,1
C	Dizüstü	I7-6700HQ	4	3500 Mhz	17.66 Gflops	8.39 Gflops	8
C	Dizüstü	Geforce GTX 950M	640	1124 Mhz	1,439 Gflops	44.96 Gflops	28,8
D	Dizüstü	I7-3630QM	4	2400 Mhz	8.83 Gflops	27.30 Gflops	5
D	Dizüstü	Geforce GT 730M	384	900 Mhz	556.8 Gflops	23.20 Gflops	28,8
Ref	-	Tesla P100	3584	1190 Mhz	4.763 Tflops	19.05 Tflops	732,2



Şekil 3.5. Yıllara göre CPU ve GPU işlemcilerin sayısal işlemlerindeki performansları

Şekil 3.5'den anlaşılacağı üzere, sadece çekirdek sayılarındaki artış değil, çekirdek hızlarındaki artışın da hesaplama gücündeki yükselmeye etki ettiği açıkça görülebilmektedir. Bu tezde deki GPU işlem gücünün, CPU işlem gücü yerine seçilmesinin ana sebebi, GPU işlemcilerinin özellikle HPC işlemleri için tasarlanmış olmalarından kaynaklanmaktadır. CPU işlemciler işlem tipi ayırmadan tüm işlemleri yapabilme kabiliyetleri olsa da belli bir konuda üretilmedikleri için GPU işlemcilere göre büyük veri içeren ve çok tekrarlı olan işlemlerde geride kalmaktadırlar. Bunun yanında, GPU işlemcilerinin mimarileri dışındaki işlemleri yerine getirebilme performansları çok düşüktür. İki işlemci arasındaki mantıksal olarak ana farklılık, CPU 'nun tek bir hat üzerinde sıralı işlemleri gerçekleştirmede başarılı olması, GPU 'nun ise birçok hat üzerine bölünmüş paralel işlemleri aynı anda gerçekleştirebilmede başarılı olmasından kaynaklanmaktadır. Her iki işlemcide yapılarının tersine işlemler yapabiliyor olsalar da performans açısından, CPU işlemciler tek hatlı işlemlerde başarılı paralel işlemlerde başarısız; GPU işlemciler de tek hatlı işlemlerde başarısız, paralel işlemler başarılıdırlar.

Değinilmesi gereken bir başka noktada, CPU işlemcilerin GPU işlemcilerine göre yüksek hafıza hızları kullanmasına rağmen GPU işlemcilerin her bir işlem dizesinde birden çok komutu düşük hafıza aralığında tamamlayabilmesidir. Böylece CPU işlemcisi problemin her bir elemanını sırayla sonuçlandırmaya çalışırken, GPU işlemcisi aynı problemi farklı girdiler altında tek bir seferde sonuçlandırabilmektedir. Bu durumun gerçek hayattaki analojisi bir nakliye problemiyle anlatılabilir. Örneğin, belirli bir konumdaki sabit ebatlarda kutular kullanılarak istiflenmiş bir miktar malzemenin bulunduğunu düşünelim. Bu malzemenin başka bir konuma bir araç vasıtasıyla taşınması işlemi de problemi oluştursun. Bu işlem için seçeneklerimiz, hızlı ama taşıma kapasitesi 1-2 kutu ile sınırlı bir araç ile, diğer araca göre yavaş ama tek seferde düzinelerce kutuyu

taşıyabilecek bir diğer yük aracı olsun. Sonuç olarak bu problemde, en hızlı şekilde ve en az enerji harcamasıyla bu işlemin tamamlanması için, en az sefer de tüm kutuları taşıyabilecek olan, yavaş ama işlem kapasitesi yüksek olan ikinci tip aracın seçilmesinin daha mantıklı ve başarılı bir sonuca ulaştıracağı ortadadır. Burada dikkat edilmesi gereken durum bir çözüm bulmaktan çok, problemin tipine göre doğru çözüm aracına karar verilmesidir Sonuçta problem, her iki araçla da çözüme ulaştırılabilir. Ama yük aracı tek seferde yavaş olsa da hızlı olan araca göre tüm taşıma işlemini daha hızlı gerçekleştirebilmiştir.

İlk kez kişisel bilgisayarlar için üretilmiş grafik işlemcisi "RIVA TNT" adıyla, NVIDIA firması tarafından 1995 yılında piyasaya sürülmüştür. Bu yıldan beri gelişimleri devam eden GPU işlemcileri yüksek matematiksel yoğunluktaki bilimsel hesaplamalarda üst düzey avantajlara sahip olsalar da 2007 yılına kadar bilimsel çalışmalarda ki kullanımlarının yeterince yaygın olmadığı görülmektedir. Bunun başlıca sebebi, GPU işlemcisine adını veren tasarım amacından kaynaklanmaktadır. GPU işlemcilerin üretilmelerinin ana sebebi, 3 boyutlu grafik işlemlerindeki sürekli tekrarlı poligon ve ışık hesaplamalarını, yansıma hesaplarını ve 2 boyutlu ekrana yansıtmak üzere projeksiyon hesaplamalarını yapmak, bu yükü CPU işlemcisinin sırtından almaktır. Tüm bu işlemleri yapabilmek için GPU üreticisi firmalar OPENGL gibi yardımcı programlama dillerini geliştirmişlerdir. Her ne kadar bu dillerin amacı grafiksel hesaplamaları yapmak olsa da, iyi programlama yetisine sahip bilim insanları bu yazılım dilleri ve GPU işlemcisinin aynı anda işlem yapabilme yetisini (Paralel programlama) kullanarak karmaşık bilimsel denklemleri GPU işlemcisine çözdürmeyi başarmışlardır.

GPU işlemcilerinin ilk kullanım yıllarında, bu işlemcilere yapım amacı dışında bilimsel hesaplamalar yaptırmak hem çok zor olmuş hem de kontrolü yetersiz kalmıştır. Bu konudaki eksikliği fark eden ve işlemci üretim firması olan NVIDIA, GPU işlemcileri için geliştirdiği CUDA (Compute Unified Device Architecture – Tümleşik Aygıt Mimarisi Hesaplama) adlı programlama dilini ilk kez 2007 yılında ücretsiz olarak kullanıma sunulmuştur. CUDA dilinin yayınlanmasından sonra GPGPU kodlama kavramı oluşmuştur. GPU işlemcisi başta grafiksel amaçlar için üretilmiş olsa da aynı zamanda bilimsel hesaplamalarda da kullanılabilen bir işlemci olmuş ve bu andan itibaren dünya genelinde sonlu elemanlar metodu da dahil olmak üzere, bilimsel hesaplamalar içeren araştırmalarda gözle görülür bir artışa sebep olmuştur. CUDA'nın başarısından sonra OPENCL (Open Computing Language – Açık Hesaplama Dili) gibi benzer GPGPU işlemcisi programlama araçları geliştirilmiştir. OPENCL hem CPU hem de GPU üzerinde çalışabilen paralel programlar yazmayı sağlayan ve birçok işlemci üreticisi markanın (AMD, NVIDIA, INTEL, ARM vs.) desteklediği bir programlama alt yapısıdır. CUDA dilinin diğer benzerlerinden farklı olarak hafıza yönetimi ve paralel organizasyon bakımından üstünlükleri mevcuttur. CUDA programlama dili CUDA Toolkit adında bir dizi programdan oluşan bir set ile beraber sunulmaktadır. Bu set içerisinde CUDA C adlı programlama dili ile beraber CuBLAS, Truss ve Tensorflow gibi birçok özelleştirilmiş kütüphanede bulunmaktadır. Bunlardan CuBLAS, GPU işlemcisine uygun hale getirilmiş paralel fonksiyonlara sahip bir BLAS (Basic Linear Algebra Subprograms – Temel Lineer Cebir Alt Programları) kod kütüphanesidir.

Çizelge 3.2'den de görülebileceği gibi, iki büyük vektör üzerinde bulunan sayıları toplayıp üçüncü bir vektöre yazmak bu tip işlemlere örnek olarak verilebilir. GPU

işlemcilerinin sahip oldukları bu çalışma mantığı gereği Fast Fourier Dönüşümü, Monte Carlo Metodu, Gauss Eliminasyon Metodu ve Lattice Boltzmann Metodu gibi matrix işlemlerini GPU işlemcileri üzerinde yapmak, bu özelliklere sahip olmayan CPU işlemcilerine göre daha etkili olmaktadır.

3.2.1. CUDA ile programlama

CUDA programlama dili C programlama dili komut yapısı baz alınarak hazırlanmış bir dildir. C programlama dili ilk kez 1972 yılında AT&T Bell firması tarafından gene bu firmanın geliştirdiği işletim sistemini oluşturmak üzere hazırlanmış bir programlama dilidir. Bu dil, esnek alt yapısı ve zamanına göre ileri görüşlü mantıksal tasarımı yüzünden tüm dünyada hemen kabul görmüş ve yayılmıştır. Hatta bu dil 1990 yılında ANSI/ISO C ismiyle dünya standardı haline getirilmiştir. Daha sonraki yıllarda geliştirilerek C++ adını almış, bu gelişimi yılar içerisinde devam ederek 2017 yılında ISO C++ v17 sürümü yayımlanmıştır.

CUDA programlama dili C/C++ diline çok benzerlikler gösterse de bu dil içerisine yazılan kodlar CPU ve GPU içerisinde çalışabilecek şekilde ayrı ayrı gruplandırılmıştır. Aşağıdaki örnekte iki farklı vektörü toplayıp, sonucu üçüncü bir vektöre yazan programın CPU ve GPU versiyonları ayrı ayrı verilmiştir.

Çizelge 3.2. Vektör toplama işleminin CPU versiyonu (Host kodu – Parallelleştirme yok)

```
#include <iostream>
#include <math.h>

// iki vektörün elemanlarını toplayan fonksiyon
void toplama(int n, float *x,    float *y)
{
  for (int i = 0; i < n; i++)
      y[i] = x[i] + y[i];
}
int main(void)
{
  int N = 1<<20; // 1Milyon eleman
  float *x = new float[N];
  float *y = new float[N];
   // bilgisayar üzerinde x ve y dizelerini doldur
   for (int    i = 0; i < N; i++) {
    x[i] = 1.0f;
    y[i] = 2.0f;
   }
  // 1Milyon elemanı CPU üzerinde çalıştır
  toplama(N, x, y);

  // Hataları kontrol et  (değerler 3.0f olmalı)
  float maxError = 0.0f;
  for (int i = 0; i <  N; i++)
    maxError = fmax(maxError, fabs(y[i]-3.0f));
  std::cout << "Maks hata: " << maxError << std::endl;
```

Çizelge 3.2 'nin devamı

```
// Hafızayı boşalt
  delete [] x; delete [] y; return 0;
}
```

Yukarıdaki vektör toplama işlemi CPU işlemcisi üzerinde tek bir işlem birimi kullanılarak sıralı bir biçimde gerçekleşmiştir. Aynı uygulamayı GPU işlemcisi üzerinde birçok işlem birimini kullanarak gerçekleştirerek yazmak için toplama() fonksiyonunda bazı değişiklikler yapılmalıdır. Bu amaçla yazılmış, GPU işlemcisi üzerinde çalışmak üzere tasarlanmış olan bu kodlar CUDA dilinde "kernel" olarak isimlendirilmektedir. Ayrıca CUDA dili hem sıralı hem paralel yapıda olduğu için, yazılan kodların GPU işlemcisi üzerinde çalışan kısımlarına "device code", CPU işlemcisi üzerinde çalışan kısımlarına ise "host code" denilmektedir. Aşağıda toplama() fonksiyonun GPU işlemcisi üzerinde çalışan device kodu verilmiştir.

Çizelge 3.3. İki vektörü GPU işlemcisi üzerinde toplayan kernel fonksiyonu

```
// İki vektörün elemanlarını GPU üzerinde toplayan CUDA
Kernel fonksiyonu
__global__ void toplama(int n, float *x, float *y)
{
  for (int i = 0; i < n; i++)
      y[i] = x[i] + y[i];
}
```

3.2.2. CUDA ile bellek yönetimi

CUDA dilinde program yazarken, tüm değişkenlerin yüklenebilmesi için önce GPU işlemcisi üzerinde bu işlemci tarafından ulaşılabilir olan bir belleğe gereksinim vardır. Programın başında ihtiyaç duyulan bellek ile ilgili miktar ve tip gibi bilgiler ile GPU işlemcisinden bu alanın talep edilmesi gerekir. GPU işlemcisinin sahip olduğu bellek üzerinde bir veri alanı ayırmak için cudaMalloc(), bu ayrılan alanı boşaltmak için ise cudaFree() komutları kullanılmaktadır. Çizelge 3.2'de yazılmış olan Host kodunu bir Device koduna dönüştürmek için yapılması gerekenler sadece new komutu yerine cudaMalloc(), bir sonraki adımda ise delete[] komutu yerine cudaFree() komutlarını kullanmaktır. Fonksiyonun yazımı tamamlandıktan sonra GPU işlemcisi için hazırlanmış olan toplama() kernel fonksiyonunun CPU tarafından çağırılması gelmektedir. CUDA kernel çağırma komutu üçlü açısal parantez sözdizimi <<<…>>> ile gerçekleştirilmektedir. Bu komutla birlikte 1 adet GPU iş hattı çalışarak tüm işlemi GPU üzerinde yerine getirip sonuçları gene GPU üzerinde bulunan belleğe yazarak kernel fonksiyonunu tamamlayacaktır. Burada dikkat edilmesi gereken bir başka durum ise GPU üzerinde toplama() kernel fonksiyonu çağrıldıktan sonra CPU tarafından sonucun erişilebilir olması için önce bu kernelin işleminin bitmesinin beklenmesi gerekmektedir. Kernel fonksiyonunun bittiği anı tespit edip bu noktaya kadar beklemeyi sağlayan komut ise cudaDeviceSynchronize() komutudur. Yukarıda sayılan işlemlerin yapılmasıyla beraber oluşan kod Çizelge 3.4'de gösterilmiştir.

Çizelge 3.4. Vektör toplama işleminin GPU versiyonu (Device kodu – Parallelleştirme yok)

```
#include <iostream>
#include <math.h>
// İki vektörün elemanlarını GPU üzerinde toplayan CUDA Kernel
fonksiyonu
__global__
void toplama(int n, float *x, float *y)
{
  for (int i = 0; i < n; i++)
       y[i]    = x[i] + y[i];
}
int main(void)
{
  int N = 1<<20;
  float  *x, *y, *d_x, *d_y;

  // GPU üzerinden erişelibilir bir bellek ayır
  cudaMalloc(&d_x, N*sizeof(float));
    cudaMalloc(&d_y, N*sizeof(float));

  // Bilgisayar üzerinde x ve y dizelerini doldur
  for (int i = 0; i < N;   i++) {
    x[i] = 1.0f;
    y[i] = 2.0f;
  }
   // Host'dan Device'a bilgileri kopyala
  cudaMemcpy(&d_x, x, N*sizeof(float),DeviceToHost);
  cudaMemcpy(&d_y, y, N*sizeof(float),DeviceToHost);

  // 1M elemanı GPU üzerinde çalıştır
  toplama<<<1, 1>>>(N, x, y);

  // GPU kernelinin  işlemi bitirmesini bekle
  cudaDeviceSynchronize();

   // Device'dan Host'a bilgileri kopyala
   cudaMemcpy(&y, d_y, N*sizeof(float),HostToDevice);

   // Hataları kontrol et (değerler 3.0f olmalı)
  float maxError = 0.0f;
  for (int i =    0; i < N; i++)
    maxError = fmax(maxError, fabs(y[i]-3.0f));
  std::cout << "Maks hata: " <<  maxError << std::endl;

  // GPU üzerindeki belleği boşalt
  cudaFree(d_x);
  cudaFree(d_y);
  return 0;
}
```

Bu kodun performans değerlerini öğrenebilmek için gene CUDA içerisinde bulunan NPROF adlı kernel profil değerlendiricisini çalıştırarak aşağıdaki sonuçları elde edilmiştir.

Çizelge 3.5. toplama() kernel'in NVProf profil değerleri (Paralelleştirme Yok)

```
==264== NVPROF is profiling process 264, command: ./ornek1
==264== Profiling application: ./ornek1
==264== Profiling result:
Time(%)   Time      Calls    Avg        Min        Max        Name
98,93%  337,72ms     1     337,72ms   337,72ms   337,72ms   toplama()
 0,76%  2,5813ms     2     1,2906ms   1,1027ms   1,4786ms   [CUDA memcpy HtoD]
 0,31%  1,0684ms     1     1,0684ms   1,0684ms   1,0684ms   [CUDA memcpy DtoH]
```

Çizelge 3.5'den anlaşılacağı üzere toplama() kernel'linin çalışma süresi 337.72 ms sürmüştür.

3.2.3. CUDA ile iş parçacıklarına ayırma

Çizelge 3.4'deki kodu çalıştırdığımızda, her ne kadar bir milyon elemanlı iki vektörün toplanması işlemi GPU üzerinde gerçekleşmiş olsa da bu işlem sadece tek bir İş Parçacığı, İngilizce terim adıyla Thread kullanılarak gerçekleştirilmiştir. Tüm işlemi birçok thread'e bölerek paralel hale getirmek için öncelikle yapılması gereken aynı kernel kodunu paralel olarak çağırmaktır. Burada ki anahtar CUDA'nın <<<1, 1>>> söz dizimindedir. Bu söz dizilimine "Çalıştırma konfigürasyonu" denir ve GPU'ya kaç tane paralel thread'in aynı anda çalışacağını ifade eder. Burada iki parametre bulunmaktadır. İlki blok sayısı ve ikincisi tek bir blok içerisinde çalışacak thread'lerin sayısını gösterir. GPU üzerinde çalışan thread'lerin sayısı 32'nin katı şeklinde ifade edilir. Buradaki 32 sayısı işlemcinin donanımsal mimarisinin getirdiği bir sınırdır. Hepsi seçilmese bile aynı anda en az 32 thread birden çalışacaktır. Bu sınırlamaya "Warp" denir. Eğer blok ve thread sayıları ayarlanırken 32'nin katı kuralına uyulmaz ise GPU talep edilen thread sayısının bir üst katından threadleri kernel için ayırır ama kullanmaz. Bu da verimlilik kaybına yol açar. Paralel çalışmayı gösterebilmek adına Çizelge 3.6'daki kernel, 32'nin katı olan 256 sayısı kadar adette thread ile paralel bir biçimde çalıştırılacaktır. Bunun için Host kodunda çalıştırma konfigürasyonu <<<1, 256>>> olacak şekilde değiştirerek yeni kernel çağırılır.

Çizelge 3.6. İki vektörü GPU işlemcisi üzerinde paralel olarak toplayan kernel fonksiyonu

```
__global__
void toplama(int n, float *x, float *y)
{
  int index = threadIdx.x;
  int nufus = blockDim.x;
  for (int i = index; i < n; i += nufus)
      y[i] = x[i] + y[i];
}
```

Bu şekilde oluşturduğumuz yeni kernel'in NPROF ile performans değerlendirmesi aşağıdaki gibidir.

Çizelge 3.7. toplama() kernel'in NVProf profil değerleri (1.iyileştirme)

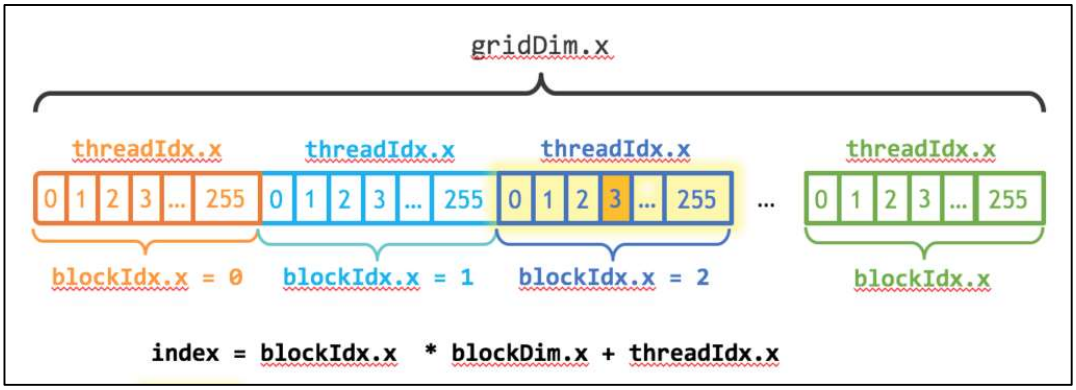
```
==7416== NVPROF is profiling process 7416, command: ./ornek2
==7416== Profiling application: ./ornek2
==7416== Profiling result:
Time(%)  Time      Calls  Avg       Min       Max       Name
35,49%   3,0030ms  1      3,0030ms  3,0030ms  3,0030ms  [CUDA memcpy DtoH]
32,82%   2,7767ms  2      1,3884ms  1,1361ms  1,6406ms  [CUDA memcpy HtoD]
31,69%   2,6819ms  1      2,6819ms  2,6819ms  2,6819ms  toplama
```

Sonuçlara bakıldığında toplama() kernel'inin çalışma süre 2.68ms'ye düşmüştür. 256 Thread kullanılarak yapılan paralelleştirme sonucunda işlem hızı 100 kat artmıştır. Yeni kernel'in içerisinde geçen threadIdx.x söz dizimi, kernel'in blok içerisinde hangi thread numarasında çalıştığını tespit edebilmesini sağlar. blockDim.x ise blok içerisinde tüm thread'lerin miktarını bildirir. Bu iki değer ile her kernel kodu, paralel thread'ler içerisindeki konumu tespit edebilecektir. GPU işlemciler SM denilen Çoklu Akış İşlemci'lerinden, her bir SM birbirinden bağımsız çalışabilen bloklardan ve her bir blok ise içerisinde çalışan birçok thread'den oluşurlar. Çalıştırma konfigürasyonunun ilk parametresi blok sayısıdır. CUDA dilinde, birden çok paralel bloklardan oluşmuş thread'lere "Grid" denilir. Çizelge 3.8'deki kod ile GPU işlemcilerinin bu özelliğini kullanarak olabildiğince çok sayıda thread'i aynı anda çalıştırıp, maksimum paralel verimlilik sağlanmıştır.

Çizelge 3.8. Paralel verimliliği arttırılmış kernel fonksiyonu

```
__global__
void toplama(int n, float *x, float *y)
{
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  int nufus = blockDim.x * gridDim.x;
  for (int i = index; i < n; i += nufus)
    y[i]    = x[i] + y[i];
}
```

Bu kernel içerisinde geçen ve yeni bir komut olan blockIdx.x, kernel'in tüm çağırılan thread'lerin içerisinde kaç numaralı bloğa ait olduğunu gösterir. CUDA ayrıca gridDim.x. komutuna sahiptir. Bu komut çağırılmış Grid'lerin kaç ayrı bloktan oluştuğu bilgisini verir. Bu komutlar yan yana kullanılarak tek bir thread'in GPU içerisindeki konumunu bize sağlarlar. Thread'lerin GPU içerisindeki dizilimi ve konumlarını gösterir durum Şekil 3.6'de gösterilmiştir.



Şekil 3.6. CUDA Paralel tread hiyerarşisi

Böylece yeni kernel'in host tarafından çağırılan kod dizilimi Çizelge 3.9'de yazıldığı gibi olur. Bu kod artık GPU işlemcisinde çalışöaya hazırdır.

Çizelge 3.9. Vektör toplama işleminin GPU versiyonu (Device kodu)

```
#include <iostream>
#include <math.h>
// İki vektörün elemanlarını GPU üzerinde toplayan CUDA Kernel
fonksiyonu
__global__
void toplama(int n, float *x, float *y)
{
int index = blockIdx.x * blockDim.x + threadIdx.x;
int stride = blockDim.x * gridDim.x;
for (int i = index; i < n; i += stride)
y[i] =  x[i] + y[i];
}

int main(void)
{
int N = 1<<20; // 1Milyon eleman
float *x, *y, *d_x, *d_y;

// GPU üzerinden erişelibilir bir bellek ayır
cudaMalloc(&d_x, N*sizeof(float));
cudaMalloc(&d_y, N*sizeof(float));

//  Bilgisayar    üzerinde x ve y dizelerini doldur
for (int i = 0; i < N; i++) {
x[i] = 1.0f;
y[i] = 2.0f;
}
```

Çizelge 3.9'un devamı

```
// Host'dan Device'a bilgileri kopyala
cudaMemcpy(d_x, x, N*sizeof(float),cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N*sizeof(float),cudaMemcpyHostToDevice);

// 1M elemanı GPU üzerinde çalıştır
int blockSize = 256;
int numBlocks = (N + blockSize - 1) /  blockSize;
toplama<<<numBlocks,    blockSize>>>(N, x, y);

// GPU kernelinin işlemi bitirmesini bekle
cudaDeviceSynchronize();

// Device'dan Host'a bilgileri kopyala
cudaMemcpy(y, d_y, N*sizeof(float),cudaMemcpyDeviceToHost);
  std::cout << "Maks hata: " << maxError << std::endl;
  // Hataları kontrol et (değerler 3.0f olmalı)
  float maxError = 0.0f;
  for (int i = 0; i < N; i++)
    maxError = fmax(maxError, fabs(y[i]-3.0f));

  // GPU üzerindeki belleği boşalt
  cudaFree(d_x);
  cudaFree(d_y);

  return 0;
}
```

Oluşturulan son kernel'imizin NPROF ile performans değerlendirmesi aşağıdaki gibidir.

Çizelge 3.10. toplama() kernel'in NVProf profil değerleri (2.iyileştirme)

```
==16556== NVPROF is profiling process 16556, command: ./ornek3
==16556== Profiling application: ./ornek3
==16556== Profiling result:
Time(%)  Time     Calls   Avg      Min        Max        Name
54,64%   2,5507ms    2   1,2753ms  1,1016ms  1,4491ms   [CUDA memcpy HtoD]
35,09%   1,6378ms    1   1,6378ms  1,6378ms  1,6378ms   [CUDA memcpy DtoH]
10,27%   479,39us    1   479,39us  479,39us  479,39us   toplama
```

Son kernel'in süre değerlerine bakıldığında toplama( ) kernel'inin çalışma süre 479.39us 'ye düşmüştür. Bu, bir önceki sonuca göre 5,5 kat daha artış demektir. Böylece blokların Grid olarak kullanımları sonucu işlem hızı paralel olmayan örneğe göre 700 kattan fazla artmıştır.

3.3. Optimizasyon ve BBO Yöntemi

3.3.1. Optimizasyon kavramı

Optimum kelimesinin kökenine bakıldığında, Latince "nihai ideal" manasına geldiği görülmektedir. Optimizasyon kelimesinin manası ise kelimenin içerisinde bulunan "en son" ve "en uygun" anlamlarının bütününü içeren bu hedefe ulaşma işlemi

olarak tanımlanabilir. Tanımsal ve matematiksel birçok karşılığı olsa da pratik anlamda optimizasyon kelimesi ile anlatılmak istenen, çözümün nihai haline ulaşırken "idealler" yani kurallar ve sınırlar çerçevesinde sonucu bulabilmek olarak tanımlanabilir.

Optimizasyon işleminde faydalı bir sonuç elde edilebilmesi için mutlaka sınırların belli edilmesi gerekir. Sınırların belli olabilmesi için ise problemin formülle edilmesi ve bunun içinde formülü oluşturmakta kullanılacak olan tasarım değişkenlerinin belirlenmeleri gerekmektedir. Tanımlanan bu formül "Amaç fonksiyonu" olarak adlandırılmaktadır. Bu aşamada, problemin tipine göre birçok farklı metot takip edilebilir. Bu metotlar, analitik yöntemler ve sezgisel yöntemler olarak iki ana parçaya ayrılırlar.

3.3.2. Optimizasyon Yöntemleri

3.3.2.1. Deterministik yöntem

Deterministik yöntemleri aynı zamanda kesin olarak optimum sonucu veren bir metot olarak da tanımlanabilir. Bu yöntemlerde elde edilen sonuç lokal optimum noktalarına takılmamışsa kesin optimum sonucu verecektir. Deterministik yöntemlerde türevsel çözüm stratejileri kullanılır. Bu yüzden yakınsama hızları oldukça fazladır. Ancak matematiksel olarak türevin oluşabilmesi için problemin tanımlı sınırlarda sürekli olması gerekmektedir. O yüzden sürekli olamayan ayrık değişkenli problemlere uygulanamaz. Ayrıca bu yöntemler başlangıç tahminlerine ihtiyaç duyar. Newton-Raphson, doğrusal programlama ve optimellik kriter metotları deterministik yöntemlerden bazılarıdır.

3.3.2.2. Stokastik/meta-sezgisel yöntem

Bu yöntemler deterministik yöntemler gibi kesin sonuca ulaştıran yöntemler olarak değerlendirilmez. Stokastik yöntemler adından da anlaşıldığı gibi optimum sonuca ulaşmak için tahminsel ve rastgeleliğin belirli kurallar çerçevesinde birlikte kullanılmasıyla elde edilmiş yöntemlerdir.

Stokastik yöntemler ayrık değişkenli ve türevi alınması zor olan denklemlerde kullanılması daha uygun yöntemlerdir. Bu yöntemde rastgele belirlenen değerler ile sonuçlar elde edilir. Yeteri kadar sonuç elde edildikten sonra temel olasılıksal değerler ile ya da Monte Karlo metodu gibi tekniklerle olasılıksal sonuca ulaşılabilir. Yöntem içerisinde olasıksal girdiler kullanıldığı için sonuçlarında belirli miktarlarda hatalar meydana gelebilir. Yöntem geneline bakıldığında tahmin ve sonuç ilişkisine dayandığı için en karmaşık problemlere bile rahatlıkla uygulanabilir ama çözüm süresi problem türüne göre vakit alabilir.

Sezgisel yöntemlerde türevsel tabanlı işlemler yerine, çözüm hafızasındaki mevcut sonuçları ele alarak özgün aday çözümler üreten işlemlere yoğunlaşılır. Bu yöntemlerde girdiler, belli koşullar altında ürettikleri sonuçlara göre sınıflandırılır ve daha sonra değerlerine göre dizilerek bulunan en başarılı sonuçların etkisi altında çözüm uzayının farklı alanlarına doğru yayılma gösterirler. Böylece çok daha farklı ve daha etkili çözümlere erişebilmek mümkündür. Sezgisel (heuristic) arama yönteminin kullanımının tarihsel olarak başlangıcını bulmak pek kolay olamasa da ilk kez sezgisel arama tanımlaması, 1945 yılında matematikçi Alan Turing tarafından kendi geliştirdiği ve

2.Dünya Savaşında kullanılmış olan Enigma adlı şifre kırıcı makinayı tanımlarken kullanılmıştır. Stokastik bileşenlere sahip algoritmalar geçmişte genellikle sezgisel olarak adlandırılsa da son zamanlarda literatürde meta-sezgisel (Metaheuristic) olarak bahsedilmektedir. Fred Glover tarafından "Metaheuristic" kelimesi Glover (1986) kendi çalışmasında ilk kez kullanılmış ve tanımlama olarak, normalde yerel bir arayışta üretilenlerin ötesinde çözümler üretmek için diğer sezgisel yöntemleri yönlendiren ve değiştiren bir ana strateji olduğuna değinilmiştir.

Metasezgisel algoritmalar yıllar içinde gelişmiş ve birçok isimde literatürlerde yerlerini almışlardır. Bunlardan en etkili olanlarından Genetik Algoritma, 1988 yılında David Goldberg ve John Holland tarafından literatüre kazandırılmıştır Goldberg ve Holland (1988) .Genetik algoritmayı, Darwin'in evrim teorisi ve doğal seleksiyon kavramlarının matematiksel operatörlerle ifade edilmesi olarak nitelendirebilir. Mutasyon, çaprazlama, uyum ve en iyinin doğada kalması ve genlerini aktarması bu metodun temel çalışma mantığını oluşturmaktadır. Bir diğer etkili meta-sezgisel yöntem ise parçacık sürü optimizasyon metodudur. İlk kez Eberhart ve Kennedy (1995) makalelerinde bu metodu dünyaya tanıtmışlardır. Bu metot, kuş ve balık sürülerinin hareketleri üzerine geliştirilmiş bir optimizasyon yöntemidir. Bu yöntem, sürü içindeki birimlerin hareket ederken çevrelerindeki diğer birimlerin pozisyonlarına, aralarındaki mesafelere göre konumlanması, yeni konumdaki yerlerine göre verimliliklerinin belirlenmesi esasına dayanan bir yöntemdir. Böylece sürü içindeki canlılar sürekli daha iyi noktalara doğru hareket ederler. Parçacık sürü optimizasyon metodu, sonraki yıllar içerisinde birçok topluluk tabanlı optimizasyon yönteminin oluşmasına öncü olmuştur.

3.3.3. Optimizasyon problemi

Belirli kısıtlamalar çerçevesinde bilinmeyen etkenlerin hesaplanmasını kapsayan problemler optimizasyon problemleri içerisine girerler. Çözüme başlamadan evvel problemin nereye doğru evirileceğine kesin olarak karar verilmelidir. Problem minimize edilecek bir maliyet veya maksimize edilecek bir kar fonksiyonunu amaç edinebilir.

3.3.3.1. Amaç fonksiyonu

Amaç, tanımlanan maksimum ya da minimum değer amacına göre bu iki yönden birine doğru oluşturulabileceği gibi bunların kombinasyonu da olabilir. Bazı değerlerin maksimum ve diğerlerinin minimum olması durumunda istenen çözüm karma bir fonksiyondur. Bu duruma bir ürünün satın alınması için doğru performans ve fiyat aralığının belirlenmesi örnek verilebilir.

3.3.3.2. Tasarım değişkenleri

En basit tanımla optimizasyon problemindeki amaç fonksiyonunun değiştiren katsayı ya da katsayılardır. Bu katsayılar belirli aralıklarda sonsuz ya da belirli sayıda değer alabilmelerine göre sonsuz veya kesikli olmak üzere ikiye ayrılırlar. Betonarme bir yapı projesindeki beton sınıfları ve kullanılan çelik çapları bunlara örnek olarak verilebilir. Tasarım parametrelerin tamamı optimizasyon probleminde tasarım değişkeni olarak tanımlanmayabilir, buna bir örnek vermek gerekirse problemde verilen yapı içinde sadece tek bir beton sınıfı kullanılıyorsa bu değerin amaç fonksiyonu içerisinde bir

değişken olarak girilmesi elde edilen sonuçların her birinde de aynı değer mevcut olacağı için sonuçta herhangi bir fark oluşturmayacaktır.

3.3.3.3. Sınırlayıcı fonksiyonlar

Optimizasyon sonucu oluşan ama performans değerlerini sağlamayan yapılar amaç fonksiyon değerleri ne kadar iyi olursa olsun uygun tasarım olarak kabul edilemezler. Bu sebeple bu tasarımlar mevcut çözüm kümesinden çıkarılması gereklidir. Bu şekilde konulan sınırlandırmaların matematiksel ifadelere dökülmüş hallerine sınırlayıcı fonksiyonlar denilmektedir. Optimizasyonun parçaları olan tüm bu fonksiyonlar bölüm 3.3.4 detaylı olarak anlatılmıştır.

3.3.4. BBO optimizasyonunun matematiksel modellemesi

Çelik kirişlerden oluşan optimum bir çerçeve sistemin imalatı için öncelikle bu sistemi oluşturan her parçanın uygulanabilirlik ve dayanım bakımından ebatlarının boyutlandırılması, daha sonra da ekonomik bakımdan koşullandırılması gerekmektedir.

3.3.4.1. Amaç fonksiyonu

Amaç fonksiyonu, çerçeve sistemin ağırlığının minimum olmasına göre şekillendirilmiştir.

$$Minimum\ W = \sum_{r=1}^{ng} \left(m_r \cdot \sum_{s=1}^{t_r} l_s\right) \tag{3.13}$$

Burada, W çerçeve sistemin toplam ağırlığı, m_r ise her bir yapı elemanının birim ağırlığıdır. Bu birim ağırlık standart çelik çizelgesindeki elemanlardan oluşan r grubunun üyesidir. t_r ise grup r içerisindeki toplam elemanları ifade ederken, ng tüm çerçeve sistemdeki toplam grup sayısını ifade etmektedir. l_s ise s elemanının uzunluğunu ifade eder.

3.3.4.2. Sınırlayıcı fonksiyonlar

Çelik kirişlerden oluşan çerçeve sistemlerin tasarımındaki sınırlandırmalar LRFD-ASIC standartları doğrultusunda aşağıda listelenmiştir.

1. Dayanım Sınırlandırıcıları: Dış etkiler sebebiyle oluşan içsel kuvvetlere karşı çerçeve sistemin tüm elemanlarının bu kuvvetlere mukavemet göstermesi gerekmektedir.
2. Deplasman Sınırlayıcıları: Çerçeve sistemdeki kirşlerin deplasmanı ve yanal hareketleri “ASCE Ad Hoc Committee Report” Tablosu içerisindeki değerleri aşmamalı.
3. Geometrik Sınırlayıcılar: Çerçeve sistem içerisinde seçilmiş olan kiriş ve kolanlara ait kesitlerin kiriş – kolon ve kolon – kolon birleşimleri esnasında birbirlerine bağlanabilecek olacak şekilde uyumlu olmaları gereklidir.

Listelenen bu sınırlandırıcıların matematiksel tariflerinden ilki olan dayanım sınırlandırıcıları için burulmanın dayanıma etkisi göz ardır edildiği koşullarda, kolon ve kirişlerde geniş başlıklı I (W) kirişler için aşağıdaki eşitsizlik söz konusudur.

$$g_{s,i} = \left(\frac{P_u}{\phi P_n}\right)_{il} + \frac{8}{9}\left(\frac{M_{ux}}{\phi_b M_{nx}} + \frac{M_{uy}}{\phi_b M_{ny}}\right)_{il} - 1.0 \leq 0 \qquad \frac{P_u}{\emptyset P_n} \geq 0.2\ için \qquad (3.14)$$

Ya da

$$g_{s,i} = \left(\frac{P_u}{2\phi P_n}\right)_{il} + \left(\frac{M_{ux}}{\phi_b M_{nx}} + \frac{M_{uy}}{\phi_b M_{ny}}\right)_{il} - 1.0 \leq 0 \qquad \frac{P_u}{\emptyset P_n} < 0.2\ için \qquad (3.15)$$

Burada i elemanı için, M_{nx} güçlü eksen boyunca (x ekseni) nominal eğilme dayanımını, M_{ny} zayıf eksen boyunca (y ekseni) nominal eğilme dayanımı, M_{uy} zayıf eksen üzerindeki gerekli eğilme dayanımını, M_{ux} güçlü eksen üzerindeki gerekli eğilme dayanımını, P_n nominal eksenel dayanımı (gerilme ve basınç), P_u gerekli eksenel dayanımı (gerilme ve basınç) ve l ise yükleme durumunu ifade etmektedir.

Çelik çubuklarda sehim ve deplasmanlar 'ASCE Ad Hoc Committee report' tarafından belirli limitler içerisinde tanımlanmıştır. Kabul edilebilir ötelenme limitleri bina yüksekliği olan (H)'ın 1/750 ila 1/250 değerleri arasında kabul edilmiştir. Tavsiye edilen değer ise (H/400) olarak bildirilmiştir. Yüksek sehim sonucu kirişlerde meydana gelebilecek çatlakların önüne geçebilmek için sehim sınırlandırılması şarttır. Sehim sınırlandırıcıları için formül aşağıdaki gibidir.

$$g_{d,j} = \frac{\delta_{jl}}{\delta_j^u} - 1 \leq 0 \quad j = 1, \dots, n_{sm}\ , \qquad l = 1, \dots, n_{lc} \qquad (3.16)$$

Burada, δ_{jl} , j numaralı elemanın l numaralı yükleme durumu altındaki maksimum sehmini, δ_j^u çelik çubuğun alabileceği sehimin üst limitini, n_{sm} sehim limitinin uygulandığı elemanların tamamını ve n_{lc} ise yükleme sayısını temsil etmektedir.

Son olarak çerçeve sistemlerde iki tip sınırlandırıcıya daha değinilecektir. Bunlar en üst kat ve katlar arası öteleme sınırlayıcılarıdır. En üst kat öteleme sınırlayıcısı şu şekilde ifade edilir.

$$g_{tdj} = \frac{\left(\Delta_{top}\right)_{jl}}{Limit} - 1 \leq 0 \quad j = 1, \dots, n_{jtop}\ , \qquad l = 1, \dots, n_k \qquad (3.17)$$

Burada n_{jtop} en üst kattaki düğüm noktalarının sayısını, $\left(\Delta_{top}\right)_{jl}$ en üst kattaki j numaralı düğüm noktasının l numaralı yükleme durumunu ifade etmektedir. Öteleme sınırlama değeri genellikle yapı yüksekliğinin 400'de biri olarak alınır.

Çok katlı çelik çerçeve yapılarda her kat arasındaki yatay deplasmanların sınırlandırılmaları gerekmektedir. Bu sınır ise genelde kat yüksekliğinin 300'de biri kadardır.

$$g_{idj} = \frac{(\Delta_{oh})_{jl}}{Limit} - 1 \leq 0 \quad j = 1, \dots, n_{st} \ , \qquad l = 1, \dots, n_{lc} \tag{3.18}$$

Burada n_{st} kat sayısını, n_{lc} yükleme durumu sayısını, $(\Delta_{oh})_{jl}$ j numaralı kattaki ötelemenin l numaralı yükleme durumundaki değerini ifade etmektedir.

Kolanların ve kirişlerin matematiksel ebatlarının seçiminin yanı sıra mantıksal olarak da birbirlerine uyumları gerekmektedir. Örneğin, çerçeve sistem içerisindeki katlardaki kolonların üst katlardan aşağı katlara doğru gittikçe kesit alanlarının büyümesi istenen ve mantıklı bir seçimdir. Tam aksine bir seçim hem yapıya fazla yük bindirecek hem de ekonomik olmayacaktır. Bir diğer durum ise kolon kiriş bağlantılarında kolon başlık büyüklüğünün bu kolona bağlanacak kirişin kesit alanına eşit ya da daha büyük olması şartıdır. Aksi halde kiriş kolona bağlanamaz ve sistem içerisindeki vazifesini yerine getiremez.

Bu durumları matematiksel olarak ifade edersek, kolon-kolon uyumu ile ilgili olarak aşağıdaki denklem yazılabilir.

$$g_{cdi} = \frac{d_{ic(\text{üst})}}{d_{ic(alt)}} - 1 \leq 0 \quad ic = 1, \dots, n_{cc} \tag{3.19}$$

$$g_{cmi} = \frac{W_{ic(\text{üst})}}{W_{ic(alt)}} - 1 \leq 0 \quad ic = 1, \dots, n_{cc} \tag{3.20}$$

Burada, n_{cc} çerçeve sistemindeki kolon-kolon birleşimlerinin sayısını, $W_{ic(\text{üst})}$ üst kolonun kesitinin birim ağırlığını, $W_{ic(alt)}$ alt kolonun kesitinin birim ağırlığını, $d_{ic(\text{üst})}$ üst kolonun W kesitinin derinliğini ve $d_{ic(alt)}$ alt kolunun W kesitinin derinliğini gösterir.

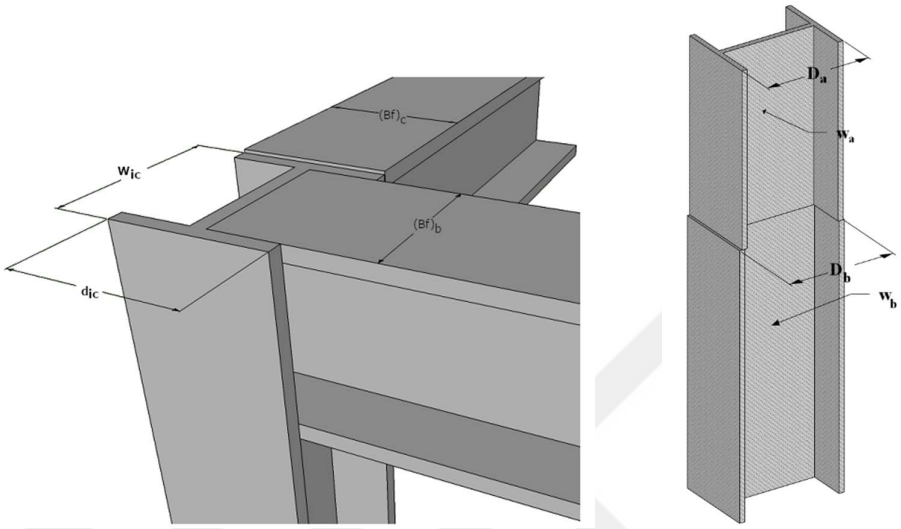
Kiriş ve kolon bağlantılarında sınırlandıracak iki durumu söz konusudur. Eğer kiriş, kolonun flanşına bağlanıyorsa, kolon flanş genişliği, kirişin flanş genişliğine eşit ya da daha büyük olmalıdır. Eğer kiriş kolonun gövdesine bağlanıyor ise kirişin flanş genişliği $(d - 2t_b)$ değerine eşit ya da küçük olmalıdır. Burada t_b kolona ait flanş kalınlığı, d ise gene kolona ait W kesitindeki derinliktir. Bu sınırlamalara ait denklemler aşağıdaki gibidir.

$$g_{bci} = \frac{\left(b'_{fb}\right)_{ib}}{(d_{cl})_{ib} - 2\left(t_{fl}\right)_{ib}} - 1 \leq 0 \ , \qquad ib = 1, \dots, n_{cb} \tag{3.21}$$

Ya da

$$g_{bbi} = \frac{\left(b_{fbk}\right)_{ib}}{\left(b_{fc}\right)_{ib}} - 1 \leq 0 \ , \qquad ib = 1, \dots, n_{cb} \tag{3.22}$$

Burada, kiriş-kolon sınırlayıcıları sayısı, b_{fbk} , b'_{fb} ve b_{fc} Şekil 3.7 de görünen B1, B2 kirişleri ve kolona ait flanş ölçüleri, d_{cl} kolonun derinliğini ve t_{fl} ise flanşın kalınlığını simgelemektedir.



Şekil 3.7. Kiriş-Kolon ve Kolon – Kolon Geometrik Sınırlanması

3.3.4.3. Performans değerinin hesaplanması

Bu fonksiyonun ana amacı, sınırlandırıcı fonksiyonlardan gelen değerleri, yapının ağırlığı çerçevesinde tek bir değere dönüştürmektir. Bunu yaparken tüm sınırlandırıcı fonksiyonlarından geçmiş yapıları, geçememiş yapılardan ayırmaya yönelik olarak farklılığı vurgulayıcı bir yöntem içermesidir.

$$f_{cost} = W_y * p_c * \ [1 + (g_s \ + \ g_d \ + (g_{bb} + g_{bc})]^2 \tag{3.23}$$

Bu denklemde W_y yapının toplam ağırlığını, g_s dayanım sınırlayıcı değerini, g_d deplasman sınırlayıcı değerini, g_{bb}ve g_{bc} sırasıyla kolon-kolon ve kiriş-kolon sınırlayıcı değerleri, p_c ise yapı maliyet katsayısını temsil etmektedir. Denklemden de anlaşılacağı üzere, yapının sınırlandırıcı fonksiyonların herhangi birinden geçememesi durumunda oluşacak değerler kareleri oranında arttırılıp f_{cost} değerine etkileri yükseltilmiştir. Böylece sınırlandırıcıları geçemeyen yapılar, geçelere oranla çok daha yüksek değerler alarak popülasyon kümesinde son sıralarda yer almaları sağlanmıştır.

3.3.5. Biocografya optimizasyon yöntemi

Biyocografya Optimizasyon yöntemi, bölgesel etki altında kalan bireylerin, bu etkiler altındaki hareketleri çerçevesinde en iyi sonuca ulaşmaya çalışan bir optimizasyon yöntemidir. Bu yöntemde bireyler bulundukları alana uyum indekslerine göre (SIV) göç etme ya da bu bölgede kalma eğilimi gösterirken, bir yandan da mutasyona uğrayarak

dışsal etkilerin yönlendirmeleriyle (HSI) başka bölgelere göç etme eğilimi içinde bulunurlar. Bu hareketler doğrultusunda sonuç tek bir nokta etrafında odaklanmaktan kurtulup dışsal değerlerin etkileri doğrultusunda daha iyi bir çözüm bölgesine hareket edebilmekte hem de doğru sonuç etrafında odaklanabilmektedir.

3.3.5.1. Göç aşaması

Göç aşamasında bir habitattaki canlılar, belirli bir popülasyon kümesi içerisinde bir habitattan diğer habitata hareket ederler. Hareketleri esnasında popülasyon içerisindeki her bir canlı (tasarım değişkeni), maruz kaldıkları dış etkiler sonucu değişime uğrarlar. Bu işlem sonucu göç eden popülasyona değişen çözüm, göç alan habitata ise referans çözüm denilmektedir. Referans çözüm, popülasyon içerisindeki çözümler arasından rulet yöntemi vasıtasıyla seçilir. Her habitatın (çözümün) referans çözüm olma olasılığı aşağıda belirtilen formülle hesaplanan habitatın göç alma olasılık oranına bağlıdır.

$$P(x_j)=\frac{\mu_j}{\sum_{i=1}^{N}\mu_i};\ j=1,\ldots, PS \tag{3.24}$$

Bu denklemde, N habitattaki canlı sayısını, μ_j göç verme katsayısını ve PS ise popülasyon sayısını temsil etmektedir.

3.3.5.2. Mutasyon aşaması

Göç hareketi içerisinde olan popülasyondaki bireyler, dış etkiler çerçevesinde özelliklerinde değişimlere, mutasyonlara uğrayarak diğer habitatlara eklenirler. BBO algoritmasında mutasyon işlemi üretilen rasgele sayının daha evvel belirlenmiş bir mutasyon oranının altında kalması durumunda gerçekleşir. Mutasyon gerçekleşirse göç aşamasında değişen çözümün mutasyona uğrayan tasarım değişkeni rasgele belirlenir.

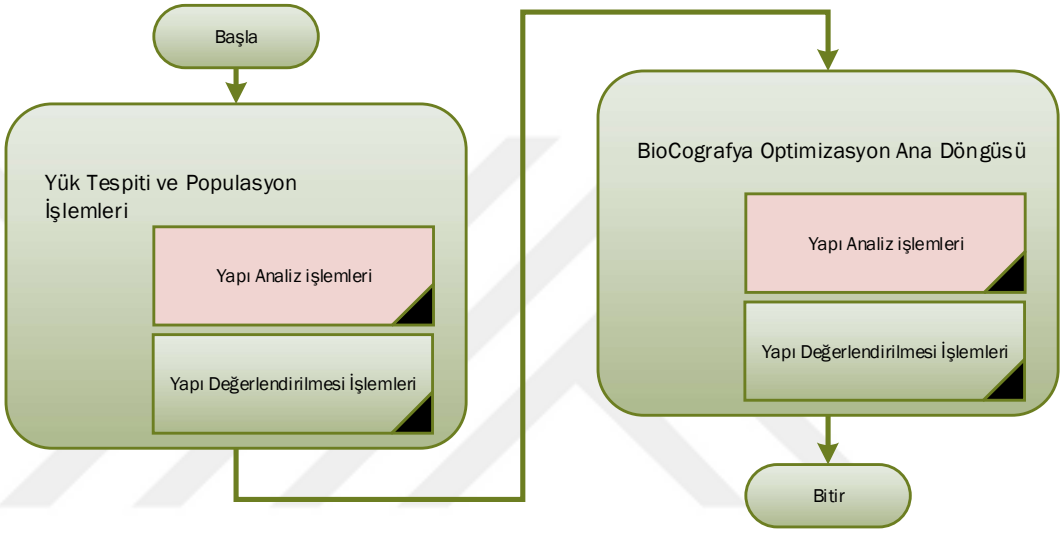
$$x_i=x_{li}+\text{rand}(0,1)(x_{ui}-x_{li});\ i=1,\ldots, PS \tag{3.25}$$

Bu denklemde x_i mutasyona uğrayan çözümün tasarım değişkenini, x_{ui} ve x_{li} değişkenin üst ve alt dizayn limitlerini PS ise popülasyon sayısını temsil etmektedir. rand(0,1) ise 0 ile 1 değerleri arasında rastgele bir değer atayan bir fonksiyondur.

BBO yönteminin algoritmik gösterimi bölüm 3.4.2'de verilen akış diyagramlarında ayrıntılı olarak gösterilmiştir.

3.4. Çerçeve Sistemlerin GPU ile Optimizasyon Modeli

Önceki bölümlerde anlatılmış olan çerçeve sistem analizinin, BBO tekniği kullanılarak optimizasyon ile birleştirilmesi işlemi temelde bir dizi yardımcı ve ana döngüye indirgenerek bu bölümde sunulmuştur. Tüm işlem belirli modüllere ayrılarak en küçük işlem adımları tespit edilebilmiş, bu adımlar içerisinden bazıları sadece GPU işlemcisi üzerinde çalışmak üzere yeniden yapılandırılmıştır. Aşağıdaki Şekil 3.8'de tüm programsal ilerleme, işleyiş şeması yardımıyla gösterilmiştir.

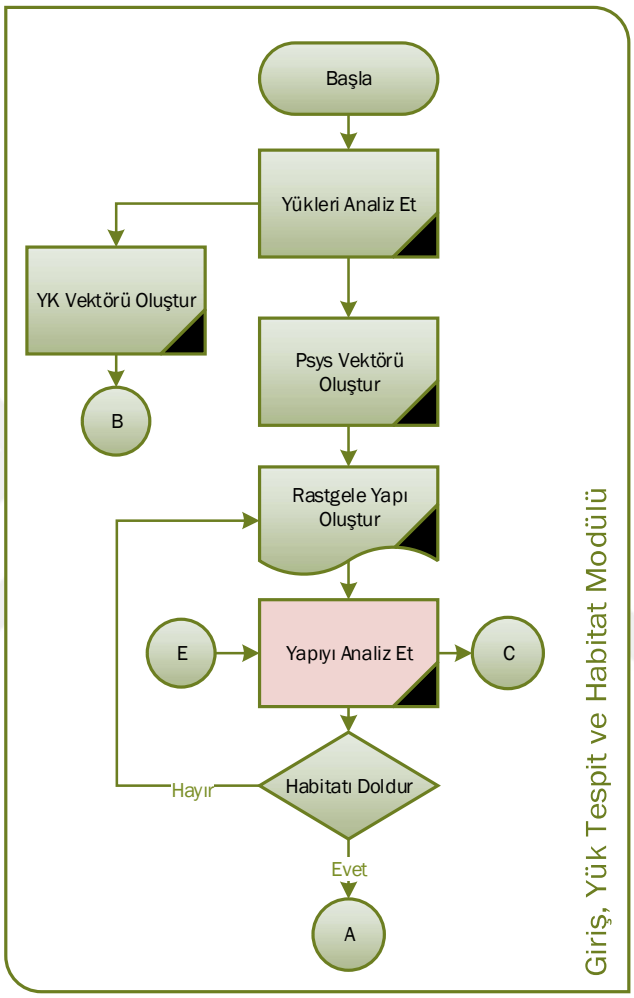


Şekil 3.8. Programsal İşleyiş Şeması

Burada geçen modüller Şekil 3.8'deki şemayı izleyerek belirli bir iterasyona ulaşana kadar devam ederler. İleriki bölümlerde her bir modülün işleyişi ayrı başlıklar altında anlatılmıştır. Bu işleyiş şeması içerisinde geçen, "Yapı Analiz İşlemleri" alt modülü sadece GPU işlemcisi üzerinde, diğer modül ve alt modüller ise CPU işlemcisi üzerinde çalışmak üzere tasarlanmışlardır.

3.4.1. Yük tespiti ve popülasyon modülü

Bu modülün içerisinde gerçekleşen işlemlerin ve alt modüllerin akış şeması Şekil 3.9'daki gibidir.



Şekil 3.9. Yük Tespit ve Habitat Modülü Akış Şeması

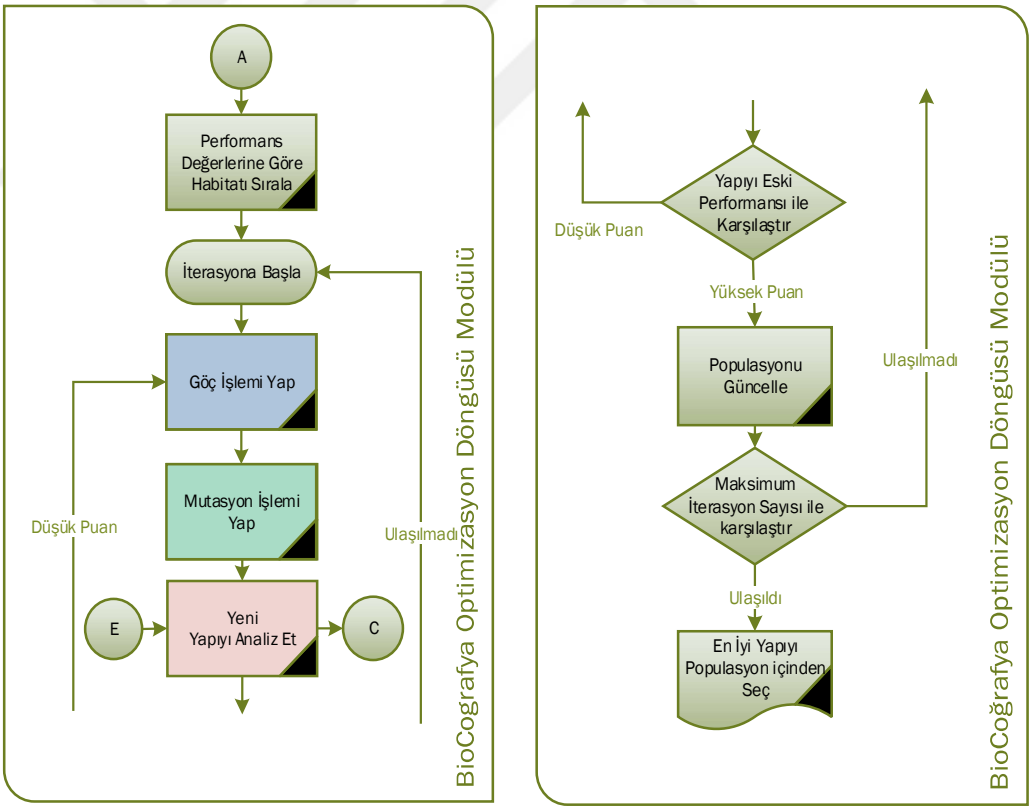
Şekil 3.9'daki şemadan da anlaşılacağı üzere, programın başlangıcından hemen sonra yapılan ilk işlem, yapı içerisindeki her bir elemanın üzerlerindeki noktasal veya yayılı yükleri, gene aynı elemanın uç noktalarına aktarmaktır. Böylece sistem yük vektörü (P_{sys}) elde edilir. Yük doğrudan yapının düğüm noktasına ekti ederse dış yük P_{sys} vektörüne eklenir. Bu işlem denklem (3.9)'da anlatılmıştır. Transformasyon matrisinin kullanıldığı bu döngüsel işlem, bu tez için hazırlanan program içerisinde, tüm düğüm noktalarındaki her bir yön için tek tek formüle edilmiş ve transformasyon matrisi kullanılmadan hesaplaması gerçekleştirilmiştir. Böylece işlem hızı kazanımı elde edilmeye çalışılmıştır.

Daha sonra BBO metodunun ilk aşaması olan popülasyon oluşturma işlemine geçilmiş ve LRFD-ASIC Tablosu içerisinden rastgele kirişler seçilerek yapı popülasyonu kümelen dirilmiştir. Daha sonra her bir yapı için oluşturulan karakteristik bilgiler "Yapı Analiz Modülü" içerisine yollanmış (C) ve modülden "Yapı Performans Değeri" bilgisi dönmüştür. (E)

Sonraki aşamada BBO içerisindeki popülasyon sayısına ulaşılıp ulaşılmadığı gözlenmiş ve ulaşıldı ise ana optimizasyon döngüsüne geçilmiş (A), ulaşılmadı ise yukarıdaki işlemler ile döngüye devam edilmiştir.

3.4.2. BioCoğrafya ana optimizasyon döngüsü modülü

Bu modül içerisinde bir önceki modülden gelen popülasyon bilgileri tek tek indekslenip BBO algoritmasındaki mantık çerçevesinde mutasyona uğratılarak yeni değerleri çerçevesinde popülasyon kümesine son şekli verilmiş daha sonra bu iyileştirme aşaması birçok iterasyon tekrar etmesiyle en iyi yapı performans değerine sahip yapı bulunmuştur. Bu modülün akış diyagramı Şekil 3.10'deki gibidir.



Şekil 3.10. BioCoğrafya Ana Optimizasyon Modülü Akış Şeması

Modülün ilk parçasında popülasyon içerisindeki tüm yapılar sahip oldukları yapı performans değerlerine göre 0 – 1 arasında bir değer alarak yüksek Performans

değerinden, düşük değere doğru sıralanmışlardır. Daha sonra, BBO algoritması mantığı gereği, rastgele olmak koşuluyla aralarından bazıları mutasyona uğratılmış, böylece bireyin göç etme ve bu göç sonucunda yeni ortamına uyum sağlama durumu gerçekleştirilmiştir. Sonraki adımda bu bireyin yeni yapısal performans değerlendirilmesi yapılıp, bu değer ile beraber popülasyona eklenmiştir. Tüm popülasyondaki yapılar performans değerlerine göre karşılaştırılmış ve aralarından en iyi olan yapı seçilmiştir. Tüm işlemler ile tek bir iterasyon gerçekleşmiş olmaktadır.

Daha sonra ki aşamada iterasyon sayısı artırılıp, maksimum iterasyon sayısına ulaşılıp ulaşılmadığına bakarak ana döngüye devam edilmiştir. Maksimum iterasyona ulaşıldığında ise döngü durdurulmuş ve program sonlandırılmıştır.

Böylece her bir döngüde hem popülasyon içerisindeki en iyi birey seçilmiş, hem de kalan bireyler göç ettirilerek düzenli olarak daha iyiye doğru mutasyona uğratılmışlardır. BBO algoritması sonucu her bir iterasyonda, bir öncekine göre aynı ya da daha iyi performanslı bireyler elde edilmiştir.

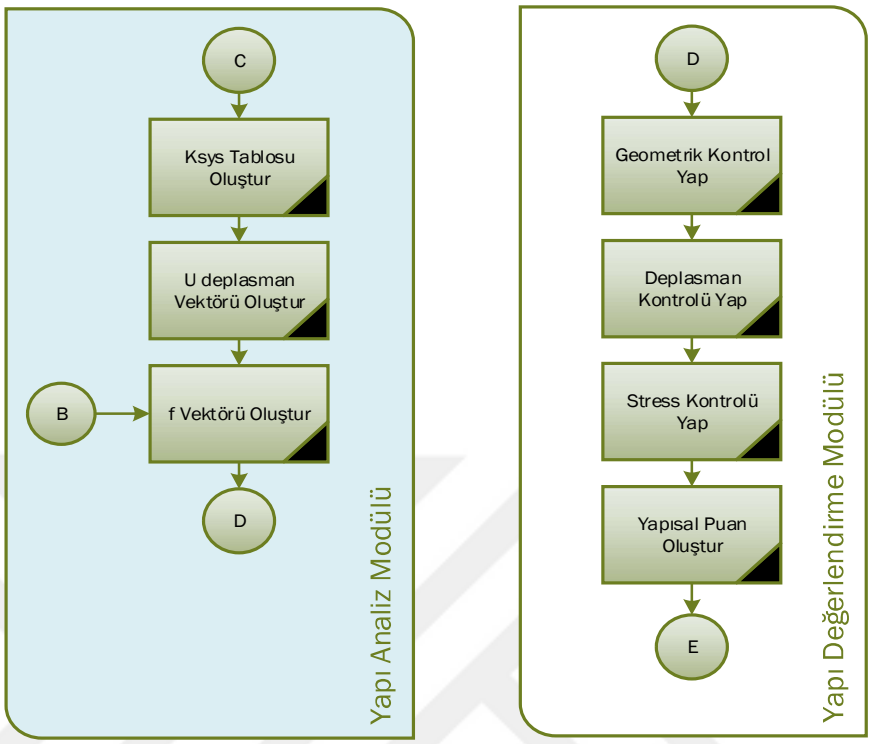
3.4.3. Mutasyon ve göç modülleri

Bu aşamada önceki modülden gelen habitat, göç alma olasılığına göre ya dışarıdan göç alır ya da dışarıya göç verir. Bu durum denklem (3.24) ile formüle edilmiştir. Göç aldığı durumda var olan bireyleri güncelleyen habitat, göç verme durumunda dışarıya göndereceği popülasyon kümesini rulet metodu ile seçip, doldurur.

Oluşan popülasyon kümesi içerisindeki bireyler dış etkenler çerçevesinde mutasyona uğrarlar, fakat tüm bireyler aynı anda ve eşit miktarlarda mutasyona uğramazlar. Bu durumu belirlemek için önceden tanımlanmış 0 ile 1 arasında bir eşik mutasyon değeri belirlenir. Daha sonra her birey için bu aralık da rastgele bir sayı belirlenerek eşik değeri aşması durumunda özellikleri yine rastgelelik çerçevesinde mutasyona uğratılıp, popülasyon kümesine geri eklenir. Bu işlem tüm göç eden bireyler için tekrar edilip popülasyon mutasyon işlemi tamamlanır.

3.4.4. Yapı ve değerlendirme analiz modülleri

Tüm programın bu aşaması, hem döngüsel olarak birçok kereler çağırılıyor olmasından hem de bütün FEM matematiksel işlemlerini ve LRFD-ASIC sınırlandırıcı koşullarının tamamının hesaplanması işlemlerini barındırıyor olmasından ötürü en çok zaman tüketen kısım olmuştur. Bu aşama bir bütün olarak tasarlandığı halde, bu tez çalışması için GPU uyarlaması çerçevesinde, işlem yükü en ağır olan kısmı GPU işlemcisinde, kalanı kısmı ise CPU işlemcisinde hesaplanacak şekilde, paralel ve hibrit olarak yeniden programlanmıştır. Bu modülün akış şeması Şekil 3.11'de gösterilmiştir.



Şekil 3.11. Yapı Analiz ve Değerlendirme Müdülü

Bu modüller program içerisinden, birden çok kere çağırıldığı için devam eden bir işlemin parçası olmaktan çok, belli girdileri ve sonuç olarak belirli çıktıları olan bir fonksiyon gibi düşünülmelidirler.

Bu modüller içerisinde yapılan işlemlere bakılırsa, öncelikle verilen yapı bilgileri çerçevesinde FEM işlemi gerçekleştirilip yapıya etkiyen yükler bulunur. Daha sonra bulunan yükler ile Bölüm 3.3.2 de geçen sınırlandırıcı koşullara göre tahkikleri yapılır. BBO algoritması için gereken ve sınırlandırıcı koşullara, ağırlıkla ve maliyet değerinin puanlaması işlemleri bir sonraki aşamada tamamlanıp, tüm işlem tek bir performans rakamına ulaşılması ile sonlanır.

"Yapı Analiz Modülü", GPU içerisinde birçok parçadan oluşan ve paralel olarak çalışmasının yanında ayrıca ilk devreye giren modüldür. Bu modül üç ana parçadan oluşmaktadır. Bunlardan ilki olan "Ksys modülü", denklem (3.8) anlatılan işlemler ile elde edilir. Bu işlemler içerisindeki yapı elemanlarının birbirleriyle bağlantıları yoluyla sonuçları birleştirilen üçlü matris çarpımı işlemi gerçekleştirilmektedir. Bu işlem sonucu oluşacak matrisin her bir elemanı GPU işlemcisi içerisindeki thread'lere atanarak matrisin oluşumu paralel bir biçimde gerçekleştirilmiştir.

Bundan sonraki aşama olan "U deplasmanının hesaplanması" işlemi, Denklem (3.10) ile anlatılmıştır. Bu denklemdeki Ksys matrisi yapıdaki düğüm noktaları sayısının her bir düğüm noktasına etkiyen üç boyutlu yüklerin sayısı olan altı ile çarpılmasıyla elde edilen sayının karesi büyüklüğündedir. Bu işlem esnasında GPU işlemciye uyarlanmış

olan Paralel CuBLAS kütüphanesi fonksiyonları kullanılarak Gauss eliminasyon metodu uygulanmış ve her bir yapı yükleme koşulu için ayrıca paralel Gauss eliminasyon işlemi gerçekleştirilmiştir.

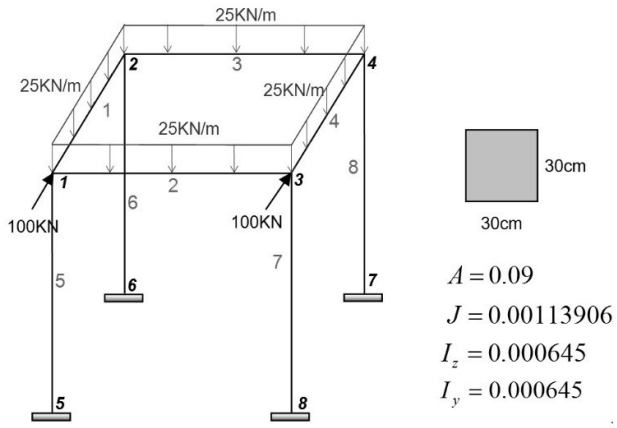
Bu bölümü irdelersek, yazılan kod içerisinde, Ksys matrisinin oluşumu, denklem (3.10)'un Gauss ile çözümü ve her bir yükleme koşulunun hesaplanması için iç içe geçecek şekilde paralellik uygulanmıştır. Bu işlemin sonucu ile denklem (3.11)'de anlatıldığı gibi her bir çerçeve elemanında oluşan kesit tesirleri hesaplanmış olur. Daha sonra denklem (3.12)'de anlatıldığı gibi düğüm noktasına etki etmeyen yükler lokal koordinatlara dönüştürülür ve çubuk iç tesiri değerlerinden çıkarılır ve tüm işlem tamamlanır.

Bu esnada yapılan transformasyon matris işlemleri yerine çubuk uç noktalarına her bir yönde etki eden tüm yükler için tek tek formüler geliştirilmiş ve bu formüller tüm düğüm noktaları için birer GPU işlemcisi thread'i atanarak paralel bir biçimde işlem hızı kazanımı elde edilerek tamamlanmıştır.

4. BULGULAR VE TARTIŞMA

4.1. Programın Doğruluğunun Test Edilmesi

Bu tez çalışması için hazırlanan, belirli kısımları GPU ve CPU işlemcilerinde çalışabilen analiz programının çözümlerinin doğruluğunun tespiti için Şekil 4.1'deki 8 elemanlı örnek çözdürülmüş, sonucun doğruluğu Sap2000 programı ile kanıtlanmıştır.



Şekil 4.1. Doğrulama için Sap2000 ile Karşılaştırılan 8 Elemanlı Örnek

Bu örnek, tez için kullanılan analiz programıyla çözülmesi sonucu oluşan düğüm noktalarındaki deplasmanlar Çizelge 4.1 'de gösterilmiştir.

Çizelge 4.1. -8- Elemanlı Örneğin Hesaplanan Deplasman Miktarları

Düğüm No	X- Deplasman (m)	Y-Deplasman - (m)	Z-Deplasman- (m)	θ-X (Radyan)	θ-Y (Radyan)	θ-Z (Radyan)
1	0.90319E-06	-0.12404E-04	-0.27672E-02	-0.57910E-03	0.79385E-09	-0.16082E-03
2	0.90088E-06	-0.30957E-04	-0.27546E-02	-0.25475E-03	0.84763E-09	-0.16082E-03
3	-0.90095E-06	-0.12404E-04	-0.27673E-02	-0.57910E-03	0.83961E-09	0.16082E-03
4	-0.90327E-06	-0.30957E-04	-0.27546E-02	-0.25475E-03	0.82861E-09	0.16082E-03

Çizelge 4.2. -8- Elemanlı Örneğin Düğüm Noktalarındaki Kuvvetler

Düğüm Başlangıç No	Düğüm Bitiş No	FX (kN)	FY (kN)	FZ (kN)	MX (kN-m)	MY (kN-m)	MZ (kN-m)
1	1	0.5825E02	0.7213E01	-0.2502E-04	0.3290E-04	0.4818E-04	-0.6346E02
1	2	-0.5825E02	0.9279E02	0.2502E-04	-0.3290E-04	0.5190E-04	-0.1077E03
2	1	0.8322E01	0.5000E02	0.1133E-03	0.1172E-03	-0.2247E-03	0.2221E02
2	2	-0.8322E01	0.5000E02	-0.1133E-03	-0.1172E-03	-0.2284E-03	-0.2221E02
3	1	0.5825E02	0.7213E01	-0.2639E-04	0.4064E-04	0.5317E-04	-0.6346E02
3	2	-0.5825E02	0.9279E02	0.2639E-04	-0.4064E-04	0.5241E-04	-0.1077E03
4	1	0.8322E01	0.5000E02	0.1217E-03	0.1067E-03	-0.2418E-03	0.2221E02
4	2	-0.8322E01	0.5000E02	-0.1217E-03	-0.1067E-03	-0.2295E-03	-0.2221E02
5	1	0.5721E02	0.4175E02	-0.8322E01	-0.1782E-03	0.2221E02	0.6346E02
5	2	-0.5721E02	-0.4175E02	0.8322E01	0.1782E-03	0.1108E02	0.1035E03
6	1	0.1428E03	0.5825E02	-0.8322E01	-0.1903E-03	0.2221E02	0.1077E03
6	2	-0.1428E03	-0.5825E02	0.8322E01	0.1903E-03	0.1108E02	0.1253E03
7	1	0.5721E02	0.4175E02	0.8322E01	-0.1885E-03	-0.2221E02	0.6346E02
7	2	-0.5721E02	-0.4175E02	-0.8322E01	0.1885E-03	-0.1108E02	0.1035E03
8	1	0.1428E03	0.5825E02	0.8322E01	-0.1860E-03	-0.2221E02	0.1077E03
8	2	-0.1428E03	-0.5825E02	-0.8322E01	0.1860E-03	-0.1108E02	0.1253E03

Bu örnek, tez için kullanılan analiz programıyla çözülmesi sonucu oluşan düğüm noktalarındaki deplasmanlar Çizelge 4.1'de ve düğüm noktalarındaki kuvvetler Çizelge 4.2'de gösterilmiştir. 8 elemanlı örneğin tez için üretilen analiz programında denenmesinden sonra aynı örnek SAP2000 programı ile kontrolü yapılmış ve aşağıdaki Çizelge 4.3'de ve Çizelge 4.4'de sırasıyla Deplasman ve uç nokta kuvvet değerleri elde edilmiştir.

Çizelge 4.3. SAP2000 programından alınan deplasman değerleri

Düğüm No	X-Deplasman (m)	Y-Deplasman (m)	Z-Deplasman -(m)	θ-X (Radyan)	θ-Y (Radyan)	θ-Z (Radyan)
1	0	0	-0.0028	-0.5794E-03	0	-0.16057E-03
2	0	0	-0.0028	-0.5794E-03	0	-0.16057E-03
3	0	0	-0.0028	-0.5794E-03	0	-0.16057E-03
4	0	0	-0.0028	-0.5794E-03	0	-0.16057E-03
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0

Çizelge 4.4. SAP2000 programı ile elde edilen elemanların uç nokta kuvvetleri

Düğüm Başlangıç No	Düğüm Bitiş No	FX (kN)	FY (kN)	FZ (kN)	MX (kN-m)	MY (kN-m)	MZ (kN-m)
1	1	-58,2514	7,2116	0	0	0	63,4638
1	2	-58,2514	-92,7884	0	0	0	-107,69
2	1	-8,3216	50	0	0	0	-22,2066
2	2	-8,3216	-50	0	0	0	-22,2066
3	1	-8,3216	50	0	0	0	-22,2066
3	2	-8,3216	-50	0	0	0	-22,2066
4	1	-58,2514	7,2116	0	0	0	63,4638
4	2	-58,2514	-92,7884	0	0	0	-107,69
5	1	-57,2116	-8,3216	41,7486	0	22,2066	-63,4638
5	2	-57,2116	-8,3216	41,7486	0	-11,0799	103,5306
6	1	-142,788	-8,3216	58,2514	0	22,2066	-107,69
6	2	-142,788	-8,3216	58,2514	0	-11,0799	125,3159
7	1	-57,2116	8,3216	41,7486	0	-22,2066	-63,4638
7	2	-57,2116	8,3216	41,7486	0	11,0799	103,5306
8	1	-142,788	8,3216	58,2514	0	-22,2066	-107,69
8	2	-142,788	8,3216	58,2514	0	11,0799	125,3159

Daha sonra Çizelge 4.5 deplasman değer farkları ve Çizelge 4.6 uç nokta kuvvet farkları tabloları oluşturulmuştur.

Çizelge 4.5. SAP2000 programı ve hazırlanan hibrit program arasında oluşan deplasman farkları

Düğüm No	X- Deplasman (m)	Y- Deplasman - (m)	Z-Deplasman- (m)	θ-X (Radyan)	θ-Y (Radyan)	θ-Z (Radyan)
1	0,00000	-0,00001	0,00003	0,00000	0,00000	0,00000
2	0,00000	-0,00003	0,00005	0,00032	0,00000	0,00000
3	0,00000	-0,00001	0,00003	0,00000	0,00000	0,00032
4	0,00000	-0,00003	0,00005	0,00032	0,00000	0,00032

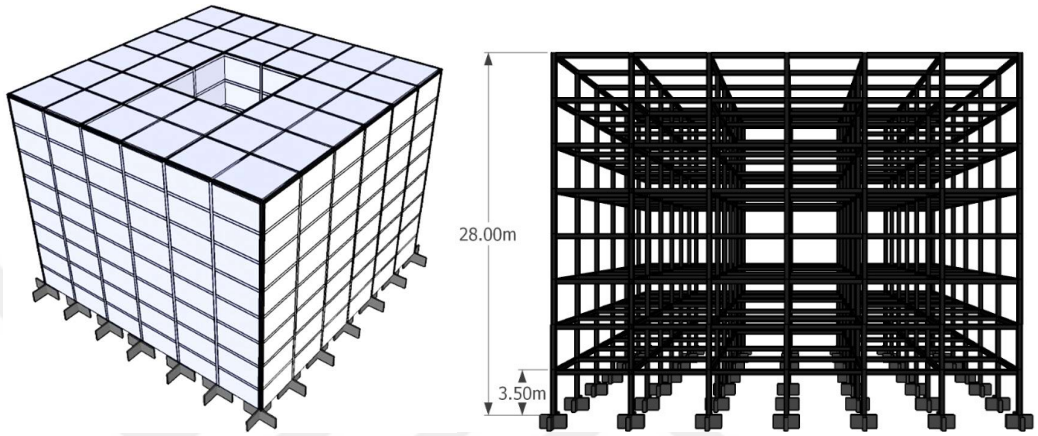
Çizelge 4.6. SAP2000 programı ve hazırlanan hibrit program arasında oluşan eleman uç nokta kuvvet farkları

Düğüm Başlanğıç No	Düğüm Bitiş No	FX (kN)	FY (kN)	FZ (kN)	MX (kN-m)	MY (kN-m)	MZ (kN-m)
1	1	-0,00140	0,00140	-0,00003	0,00003	0,00005	0,00380
1	2	0,00140	0,00160	0,00003	-0,00003	0,00005	-0,01000
2	1	0,00040	0,00000	0,00011	0,00012	-0,00022	0,00340
2	2	-0,00040	0,00000	-0,00011	-0,00012	-0,00023	-0,00340
3	1	0,00040	0,00000	0,00012	0,00011	-0,00024	0,00340
3	2	-0,00040	0,00000	-0,00012	-0,00011	-0,00023	-0,00340
4	1	-0,00140	0,00140	-0,00003	0,00004	0,00005	0,00380
4	2	0,00140	0,00160	0,00003	-0,00004	0,00005	-0,01000
5	1	-0,00160	-0,00040	0,00140	-0,00018	0,00340	-0,00380
5	2	0,00160	0,00040	-0,00140	0,00018	0,00010	-0,03060
6	1	0,01200	-0,00040	-0,00140	-0,00019	0,00340	0,01000
6	2	-0,01200	0,00040	0,00140	0,00019	0,00010	-0,01590
7	1	-0,00160	0,00040	0,00140	-0,00019	-0,00340	-0,00380
7	2	0,00160	-0,00040	-0,00140	0,00019	-0,00010	-0,03060
8	1	0,01200	0,00040	-0,00140	-0,00019	-0,00340	0,01000
8	2	-0,01200	-0,00040	0,00140	0,00019	-0,00010	-0,01590

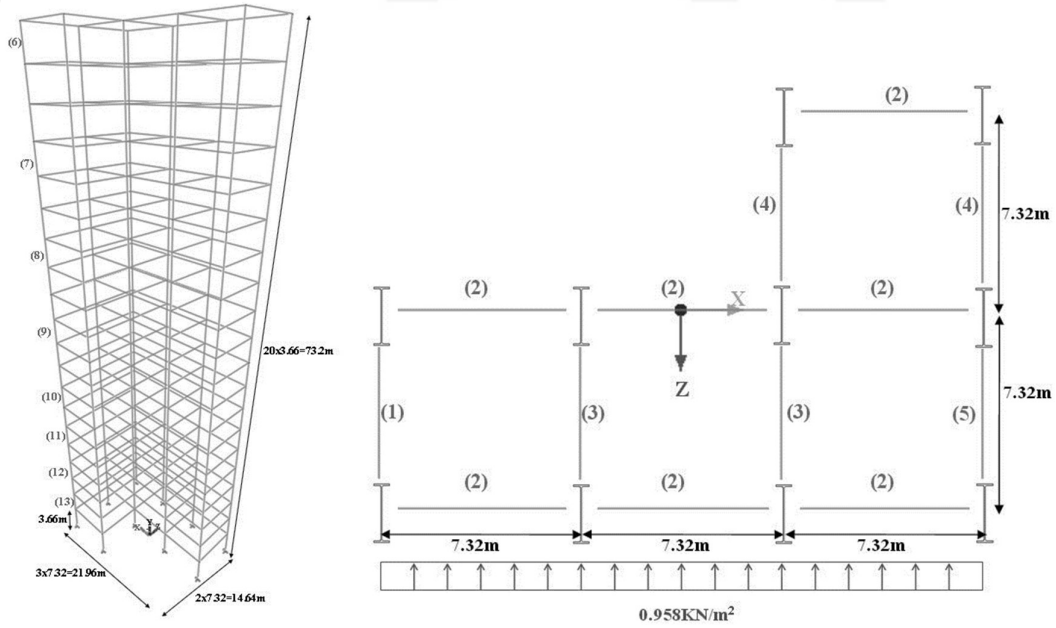
Deplasman farkları için Çizelge 4.5 gösterilen karşılaştırma tablosuna baktığımızda, Sap2000 programı ve C++/CUDA ‘da yazılmış analiz programı arasındaki farkların 0 ile 1,62% m değerleri arasında kaldığını görülmektedir. Bunun dışında Çizelge 4.6 ‘da gösterilen kuvvet uç noktası farkları tablosuna baktığımızda oluşan yük farklarının 0 ile 0,02% aralığında ve moment farklarının ise 0 ile 0,03% kN-m aralığında olduğu tespit edilmiştir. Bu da oluşan farkların ihmal edilebilecek miktarda olduğunu ve yazılan programın sonuçlarının doğru olduğunu göstermektedir.

4.2. GPU ve CPU karşılaştırılması

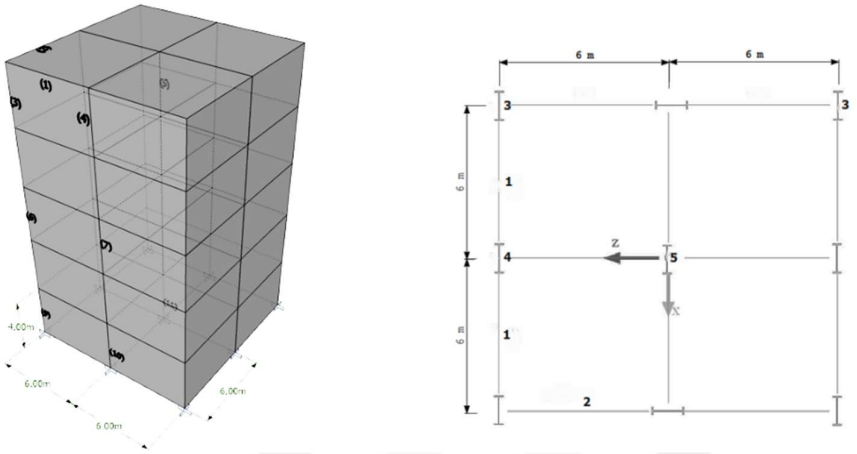
Bu tez için giderek artan sayıdaki 3 ayrı yükleme tipi altında, üç farklı eleman sayısına sahip yapılar kullanılmıştır. Bunlardan birincisi 105 elemanlı, ikincisi ise 460 elemanlı ve üçüncüsü 1024 elemanlı yapılardır. Her bir yapı tipi için kullanılan Yükleme kombinasyonlarının sayıları ise 4, 7 ve 12'dir.



Şekil 4.2. Çerçeve Yapı Örneği – 1024 Elemanlı



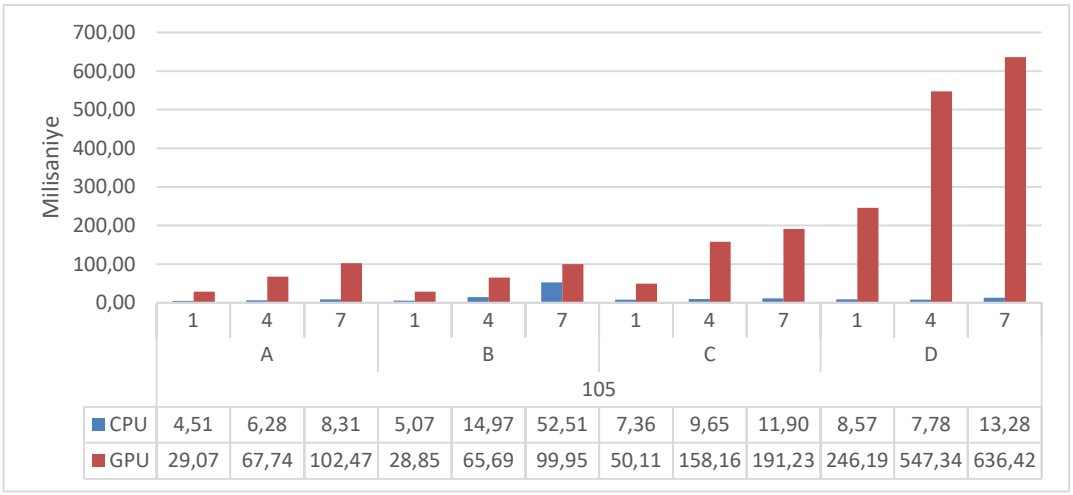
Şekil 4.3. Çerçeve Yapı Örneği - 460 Elemanlı



Şekil 4.4. Çerçeve Yapı Örneği - 105 Elemanlı

Ayrıca bu tezde yazılan Yapısal analiz ve BBO algoritması uygulaması Çizelge 3.1'deki özellikleri verilen 4 ayrı denek bilgisayar üzerinde bağımsız olarak çalıştırılmış ve performans değerleri toplanmıştır. Bu dört denek içerisinden "A" bilgisayarı bir masaüstü bilgisayar iken, kalan diğer denekler dizüstü tabir edilen taşınabilir bilgisayarlardan seçilmişlerdir.

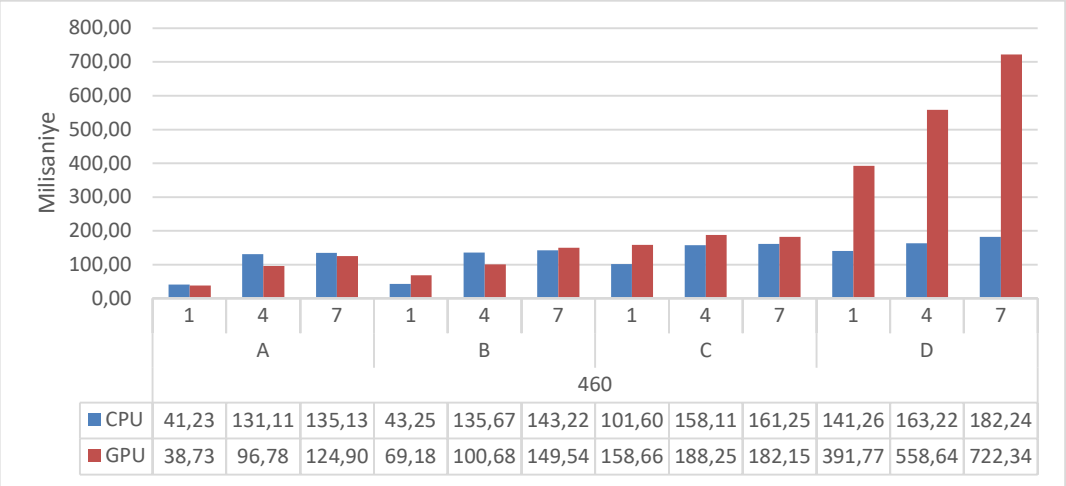
Her denek bilgisayarda aynı deney CPU ve GPU işlemcisi kullanılarak iki kere tekrar edilmiştir. Şekil 4.5'de tüm deneklerin 105 elemanlı örneği 1, 4 ve 7 yükleme tipleri altında çözmeleri sonucu oluşan süre değerleri Çizelgesi verilmiştir.



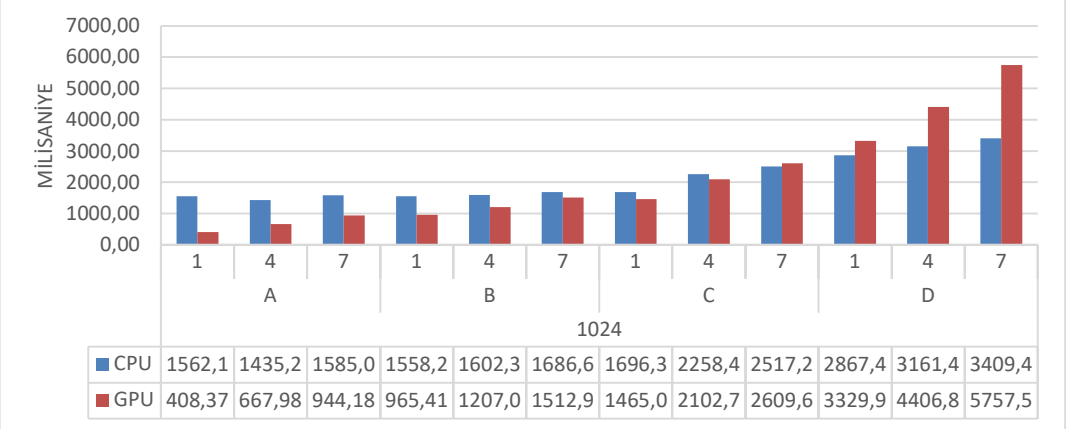
CPU	4,51	6,28	8,31	5,07	14,97	52,51	7,36	9,65	11,90	8,57	7,78	13,28
GPU	29,07	67,74	102,47	28,85	65,69	99,95	50,11	158,16	191,23	246,19	547,34	636,42

Şekil 4.5. Çerçeve Yapı Örneği - 105 Elemanlı Yapı 4,6 ve 7 ayrı Yükleme durumu altındaki CPU/GPU Performans Grafiği

İkinci bir deney olarak 460 elemanlı yapı örneği, 1, 4, ve 7 ayrı yükleme tipi altında tüm denek bilgisayarlar üzerinde çözülmüş ve oluşan performans değerleri Şekil 4.6 ‘de gösterilmiştir.



Şekil 4.6. Çerçeve Yapı Örneği - 460 Elemanlı Yapı 1,4 ve 7 ayrı yükleme durumu altındaki CPU/GPU Performans Grafiği



Şekil 4.7. Çerçeve Yapı Örneği - 1024 Elemanlı Yapı 1,4 ve 7 ayrı yükleme durumu altındaki CPU/GPU Performans Grafiği

Üçüncü ve son yapılan deney 1024 elemanlı yapı örneğine aittir. Bu yapı da diğer örnekler gibi 1,4 ve 7 ayrı yükleme tipi altında tüm denek bilgisayarlar üzerinde çözülmüş ve oluşan performans değerleri Şekil 4.7‘da gösterilmiştir.

Bu iki örneğin performans değerlerine baktığımız zaman en bariz görünen farklılık GPU işlemci performansının eleman sayısı düşük olan örnek de CPU işlemci performansını geçemediğidir. Ama 1024 elemanlı örnek de “A” denek bilgisayarına

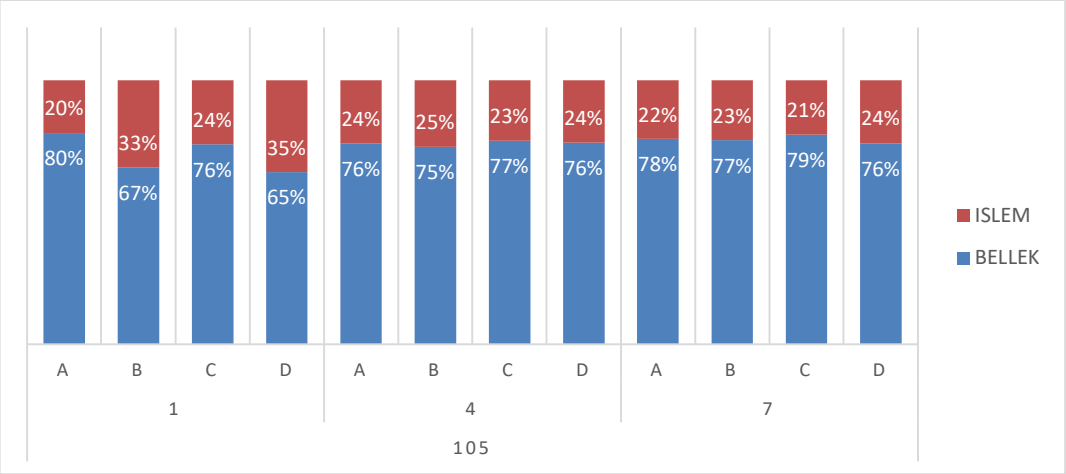
baktığımızda 3 katından fazla bir performans kazanımı elde edildiği görülmüştür. Bu durumun sebeplerini irdeleyebilmek için aynı denek üzerinde, GPU işlemcisi içerisinde hesaplama esnasında ki olayların listesi ve süreleri çıkartılarak yazılımın profil analizi hazırlanmıştır.

Çizelge 4.7. "A" Denek bilgisayarı GPU kerneli İşleyiş Profili Örneği

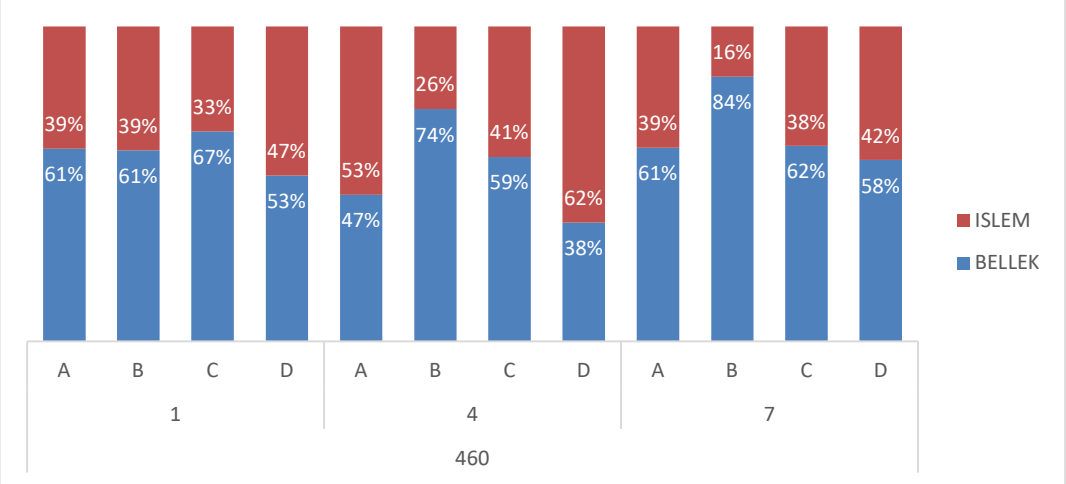
==10840== NVPROF is profiling process 10840
==10840== Profiling application: ./BBO_v3 105 7 32 0 1
==10840== Profiling result:

Time(%)	Time	Calls	Avg	Min	Max	Name
30,32%	50,405ms	2690	18,737us	2,9120us	53,215us	gemmk1_kernel
24,23%	40,269ms	18830	2,1380us	1,1520us	4,7040us	axpy_kernel_
15,34%	25,499ms	21690	1,1750us	991ns	9,4710us	CUDA memcpy DtoH
13,91%	23,129ms	70	330,41us	288,12us	375,42us	trsv_ln_exec_up
6,24%	10,367ms	10	1,0367ms	1,0320ms	1,0401ms	gpu_FULLsm
3,50%	5,8099ms	2690	2,1590us	1,7280us	3,9680us	copy_kernel
3,31%	5,5022ms	2690	2,0450us	1,5680us	2,5600us	scal_kernel_val
2,43%	4,0458ms	10	404,58us	402,97us	407,61us	gpu_XDtoF
0,46%	757,88us	10	75,787us	74,655us	77,054us	gpu_Fsonyap
0,13%	217,37us	80	2,7170us	2,4320us	4,5120us	CUDA memcpy DtoD
0,09%	149,89us	70	2,1410us	2,0480us	2,4320us	dtrsv_init_up
0,03%	44,639us	17	2,6250us	704ns	6,7520us	CUDA memcpy HtoD
0,01%	24,032us	21	1,1440us	576ns	4,7680us	CUDA memset
API calls:						
57,24%	1,77871s	21787	81,640us	13,500us	454,00us	cudaMemcpy
18,90%	587,21ms	35	16,777ms	800ns	502,51ms	cudaFree
10,45%	324,56ms	27070	11,989us	8,5000us	2,0307ms	cudaLaunchKernel
9,19%	285,52ms	1	285,52ms	285,52ms	285,52ms	cudaDeviceReset
2,87%	89,143ms	22	4,0520ms	4,5000us	88,543ms	cudaMemset
0,54%	16,846ms	30	561,52us	121,20us	1,1032ms	cudaThreadSynchro
0,42%	13,111ms	51210	256ns	100ns	64,200us	cudaGetLastError
0,34%	10,447ms	34	307,28us	6,7000us	1,0493ms	cudaMalloc
0,02%	721,70us	70	10,310us	9,4000us	14,600us	cudaEventQuery
0,02%	716,20us	285	2,5120us	200ns	120,50us	cuDeviceGetAttrib

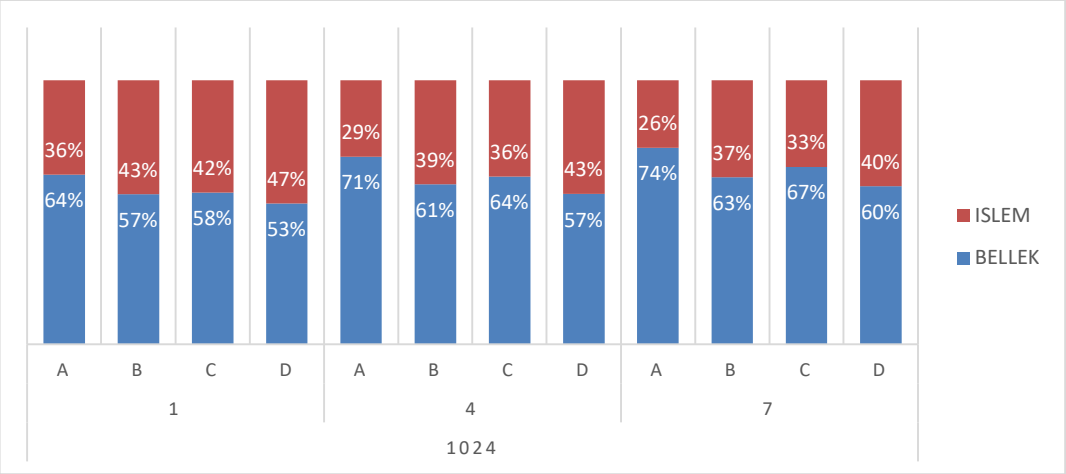
Yukarıdaki Çizelge 4.7 profil Çizelgesinden de görüleceği üzere GPUişlemcisi içerisindeki işlenen tüm adımların zamanları ve tüm işlem sürecindeki yüzdeleri tespit edilmiştir. Bütün deneklerin tüm örnekler ve türevleri altındaki profilleri çıkarılarak süreç boyunca yapılan matematiksel işlemler ve CPU işlemcisi ile GPU işlemcisi arasındaki bellek transfer işlemlerinin yüzdeleri toplanarak Şekil 4.8, Şekil 4.9 ve Şekil 4.10'da özetlenmiştir.



Şekil 4.8. 105 Elemanlı Yapıya ait 1,4 ve 7 ayrı yükleme durumu altındaki GPU Kernelinin İşlem/Bellek, zaman/yüzde Grafiği



Şekil 4.9. 460 Elemanlı Yapıya ait 1,4 ve 7 ayrı yükleme durumu altındaki GPU Kernelinin İşlem/Bellek, zaman/yüzde Grafiği



Şekil 4.10. 1024 Elemanlı Yapıya ait 1,4 ve 7 ayrı yükleme durumu altındaki GPU Kernelinin İşlem/Bellek, zaman/yüzde Grafiği

Yukarıdaki çizelgelerde açıkça görülmektedir ki, çerçeve sistemlerin FEM metodu ile optimizasyonu içerisindeki GPU işlem parçalarının tükettikleri toplam zamanın ortalama %60'a yakını verinin CPU işlemcisi host ile GPU işlemcisi device arasında oluşan transfer işlemlerinden oluşmaktadır. Kalan %40'lık süre aslında GPU işlemcisinin tüm matematiksel işlemleri yaparken tükettiği gerçek süredir. Bellek zamanlarından arındırılmış olarak işlem sürelerini CPU işlem süresi ile karşılaştırsa sadece 1024 elemanlı örnek deki "A" denek bilgisayarında 9,5 kat daha fazla süre performansı elde edilebilecektir.

105 elemanlı örneğin sonuçlarına bakarsak yapılan toplam GPU işlem süresi ortalama %23 oranında olmasına karşın host ile device arasındaki bellek gecikmeleri sebebi ile oluşan süre kayıpları GPU işlemecisini toplam süre bakımından çok gerilere indiği görülmektedir. 460 elemanlı örnekde CPU ve GPU arasındaki farklar kısmen kapanmaya başladığı, fakat eski teknolojilere sahip GPU işlemcilerinde süre avantajının hala CPU işlemcisinde olduğu anlaşılmaktadır. Son olarak 1024 elemanlı örneğe bakıldığında durumun ilk iki örneğe göre GPU tarafına doğru değiştiği ve GPU işlemci sürelerinin CPU işlemci sürelerine göre daha hızlı olduğu görülmektedir. Fakat son örnekte gene teknoloji farkı sebebi ile D denek bilgisayarı hız oranlarında geri kalmıştır.

Bu performans çizelgelerinin incelenmesi ile anlaşılan bir diğer farklı ve etkin nokta ise işlemci mimarisindeki değişimlerin, veri yolu genişliğindeki değişimlere oranla çok daha fazla bir miktarda performansa etki etmesidir. SM 3.0 mimarisine sahip "D" denek bilgisayarı hariç diğer bilgisayarlarda işlemcilerinin daha yeni olmalarından ötürü paralel matematiksel hesaplama kabiliyetleri daha iyi olan komutlara sahiplerdir. Ve bu komutların performansı, sonuçlarda açıkça kendini göstermiştir. "D" bilgisayarındaki GPU işlemcisinin diğerlerine göre komut seti eksikliği olması sebebiyle bu denek de diğer deneklerden farklı olarak çalıştırılabilecek eski tip yazılımsal çözümler kullanılmıştır. SM 3.0 mimarisi içerisinde paralel olarak ondalık sayıları atomik olarak toplamayı sağlayabilen bir komut seti, diğer deneklerin işlemcilerinin aksine bulunmamaktadır.

Diğer işlemcilerde bu paralel toplamayı sağlayan “atomicadd()” fonksiyonu bulunurken, SM 3.0 işlemcisinde olmayan bu komut yerine Çizelge 4.8’de gösterildiği gibi alternatif bir metot uygulanarak bu sorun aşılmıştır.

Çizelge 4.8. Atomicadd() komutuna alternatif fonksiyon uyarlaması

```
__device__  double atomicAdd(double* address, double val)
{
    unsigned  long long int* address_as_null =
                          (unsigned long long
int*)address;
    unsigned long long int old = *address_as_null,
assumed;
    do {
        assumed = old;
        old = atomicCAS(address_as_null  , assumed,
                        __double_as_longlong(val +
                        __longlong_as_double(assumed)));
    } while (assumed != old);
    return __longlong_as_double(old);
}
```

Yukarıdaki şekil içerisinde geçen kod ile SM 3.0 mimarisi işlemcinin içindeki her bir thread aynı anda bu komut dizilimini çalıştırarak bellekteki vektör ve matrislerle toplama işlemlerini gerçekleştirebilmişlerdir. Bu dizilimin temel mantığı, SM 1.0’dan beridir tüm komut setlerinde var olan atomicCAS() fonksiyonunu kullanarak ilgili veri alanında atomik sıralama işleminin benzeri bir durumunu canlandırmaktır. Böylece aynı veri üzerine birçok thread aynı anda yazamayacak, veri kesişimi (Data Racing) durumunun önüne geçilebilecektir.

Çizelge 3.1’de denek bilgisayarlar arasındaki CPU işlemci performans değerlerine bakıldığında çok belirgin bir biçimde, her deneğin işlemci yılı ile performansı arasında doğrudan bir ilişki olduğu görülmektedir. İşlemci model numarası bir üst seviyeye çıktıkça işlemci frekansı, bant genişliği gibi değerlerde artış olmuş, hatta bazı işlemcilerde çekirdek sayısı da paralel olarak artığı için aradaki fark daha da çok artmıştır.

5. SONUÇLAR

Bu çalışma doğrultusunda yapılan analizler incelediğinde sonuçlar performans tabanlı olarak, birkaç grup altında toplanabilmektedir. Bunlardan ilki yapısal mimari mantıklarının farklılıkları sebebi ile ikiye ayrılan CPU ve GPU işlemci grubudur. Bu farklılık da en belirgin olan özellik, işlenen veri sayısının artmasıyla doğru orantılı olarak ciddi oranda artış göstermeye eğilimli olmasıdır. Bu durum en iyi, düşük hafıza gerektiren ve daha az işlem barındıran 105 elemanlı örneğin, daha yüksek hafıza gerektiren ve daha fazla işlem barındıran 1024 elemanlı örnek ile karşılaştırılmasıyla anlaşılmaktadır. Birinci örneğin çözüm işlemlerinin klasik CPU işlemcileri ile GPU işlemcilerine göre her yükleme tipi altında yüksek performans da tamamlanabildiği gözlemlenmiştir. Bu sonuca rağmen CPU işlemci performansı 1024 elemanlı yapının çözümleri esnasında ilk örnekteki performansını yitirmiş, GPU işlemcisi karşısında geride kalmıştır. Buradan da anlaşıldığı üzere yoğun veri içeren tekrarlı işlemlerde GPU paralel işlemcisi çok daha iyi bir sonuç verme eğilimindedir.

Sonuçları gruplandırabileceğimiz ikinci etken ise farklı model GPU işlemcileri arasındadır. Bu işlemciler üretim yıllarına göre farklı GPU işlemci mimarilerine sahiptir. Bu mimariler sırasıyla Çizelge 5.1'de gösterilmiştir.

Çizelge 5.1. Denek Bilgisayarların Belirleyici GPU Özellikleri

Denek Adı	Üretim Yılı	SM Mimarisi(Major.Minor)	İşlemci Mimari Adı	GPU Adı	Veri Yolu Genişiliği (GB/sn)
A	2017	SM 6.1	Pascal	1080 titan	484,4
B	2016	SM 6.1	Pascal	1050 titan	112,1
C	2015	SM 5.0	Maxwell	GTX 950M	28,8
D	2013	SM 3.0	Kepler	GT 730M	28,8

Çizelge 5.1'de görülmektedir ki, GPU işlemcileri arasındaki performans farklılıkları sadece veri yolu genişliği tarafından değil, aynı zamanda işlemci mimarisi yani işlemci komut setindeki farklılıklar sebebiyle de oluşmaktadır. "D" bilgisayarının, GPU işlemcisinin diğer deneklere göre kısmen eski olmasından ötürü, tüm tez çalışmasının kalbini oluşturan matris çarpma işlemlerinde kullanılan atomik toplama işlemlerini desteklememesi çok büyük bir performans kaybına yol açmıştır. Bu atomik toplama işlemleri alternatifleri birçok satırdan oluşan fonksiyonlar şeklinde olmasından ötürü ve bu fonksiyonların defalarca çağırılması ciddi performans kaybına sebep olmuştur.

Tüm bu çalışma sonucu elde edilen bilgiler ışığında söylenebilir ki, içinde aynı işlemi çok tekrarlı bir şekilde kullanan FEM işlemlerini sıralı bir biçimde çözmek yerine paralel olarak çözmenin performans kazanımları sağlayacağı açıktır. Paralelizm mantığı ile performans elde edilebilmesi, kesinlikle donanımsal değişkenlere çok bağlı ve işlemin veri tüketimi ile çok alakalıdır. GPU işlemcisi içerisinde olan olaylar ile ilgili profil çizelgeleri göstermiştir ki daha az veri hareketi oluşturan her türlü işlem, tekrarı ya da

boyutları ne kadar çok olursa olsun, sonuçta çok ciddi bir performans kazanımı sağlayacaktır.

Deney sonuçları göstermiştir ki, birçok çeşitte ve sayıda işlemcinin bir arada bulunması ile oluşan süper bilgisayarların yapabildiği FEM ve optimizasyon kombinasyonu olan simülatif uygulamaların, günlük hayatımızda sıradan işlerimiz için kullandığımız ve performans bakımından masaüstü bilgisayarlardan geride olan dizüstü bilgisayarlar ile de yapılabilmektedir. Bu kapanan devasa farklığın arkasında ki en büyük güç günümüzde her bilgisayarın içerisinde bulunan Grafik İşleme Ünitesi ve paralel programlama mantığıdır.

Bunların dışında sonuçlar göstermiştir ki, GPU içerisinde ya da GPU ile CPU işlemcileri arasında olan her türlü veri transferi işlemleri çok ciddi süre kayıpları yaratmaktadır. Bu sebeple, analiz programının içerisinde host ve device arasında olabildiğince veri transferine engel olunabilmesi için, kodun en çok tekrar gerektiren fonksiyonları başta olmak üzere, mümkünse tüm fonksiyonlarının GPU içerisinden çalışacak hale getirilmesinin, uygulamanın hibrit çalışmasına nazaran çok daha verimli olmasını sağlayacağı kanaatini oluşturmuştur.

Bu çalışma sonucunda elde edilen optimum tasarımlar gerçek ölçekli yapı tasarımlarına uygulanabilir niteliktedir. Bu sebeple önerilen tasarımlar ile daha ekonomik daha verimli yapılar yapılabilecektir. Özetle, çalışma ülke ekonomisine ve dünyadaki kaynakların daha verimli tüketilmesine yarar sağlayacaktır.

Yukarıda bahsedilen durumlara ek olarak, eğer programlama teknikleri ilgili yönde gelişir ise GPU paralel hesaplama teknolojisinin inşaat mühendisliğinde optimizasyon ve FEM gerektiren çerçeve sistemlerin dışındaki diğer yapı tasarım problemleri içinde kullanılabileceği; daha basit ve günlük yaşantıda kullanılan standart bilgisayarlarında ağır hesaplamaları zorlanmadan yapabileceği yada gerçek zamanlı optimize edilmiş, iteratif simülasyonların gene bu teknolojiler yardımıyla günlük hayatta kullanılabileceği görülmüştür. Gelecekte bu belirtilen konularla ilgili araştırma ve geliştirme çalışmalarının yapılması önerilmektedir.

6. KAYNAKLAR

Afzali, S. H., Darabi, A. ve Niazkar, M. 2016. Steel frame optimal design using MHBMO algorithm. *International Journal of Steel Structures*, 16, 2: 455-465.

Akbari, J. ve Ayubirad, M. S. 2017. Seismic Optimum Design of Steel Structures Using Gradient-Based and Genetic Algorithm Methods. *International Journal of Civil Engineering*, 15, 2A: 135-148.

Amadio, C., Bedon, C., Fasan, M. ve Pecce, M. R. 2017. Refined numerical modelling for the structural assessment of steel-concrete composite beam-to-column joints under seismic loads. *Engineering Structures*, 138, 394-409.

Artar, M. 2016. Optimum design of steel space frames under earthquake effect using harmony search. *Structural Engineering and Mechanics*, 58, 3: 597-612.

Aydogdu, I. 2017. Cost optimization of reinforced concrete cantilever retaining walls under seismic loading using a biogeography-based optimization algorithm with Levy flights. *Engineering Optimization*, 49, 3: 381-400.

Aydogdu, I., Akın, A. ve Saka, M. P. 2016. Design optimization of real world steel space frames using artificial bee colony algorithm with Levy flight distribution. *Advances in Engineering Software*, 92, 1-14.

Aydogdu, I., Carbas, S. ve Akın, A. 2017. Effect of Levy Flight on the discrete optimum design of steel skeletal structures using metaheuristics. *Steel and Composite Structures*, 24, 1: 93-112.

Aydogdu, I., Efe, P., Yetkin, M. ve Akın, A. 2017. Optimum design of steel space structures using social spider optimization algorithm with spider jump technique. *Structural Engineering and Mechanics*, 62, 3: 259-272.

Aydoğdu, İ. 2010. Optimum design of 3-d irregular steel frames using ant colony optimization and harmony search algorithms.

Azad, S. K. 2017. Enhanced hybrid metaheuristic algorithms for optimal sizing of steel truss structures with numerous discrete variables. *Structural and Multidisciplinary Optimization*, 55, 6: 2159-2180.

Balogh, T. ve Vigh, L. G. 2017. Optimal Fire Design of Steel Tapered Portal Frames. *Periodica Polytechnica-Civil Engineering*, 61, 4: 824-842.

Barraza, M., Bojorquez, E., Fernandez-Gonzalez, E. ve Reyes-Salazar, A. 2017. Multi-objective Optimization of Structural Steel Buildings under Earthquake Loads using NSGA-II and PSO. *Ksce Journal of Civil Engineering*, 21, 2: 488-500.

Bojorquez, J., Ruiz, S. E., Ellingwood, B., Reyes-Salazar, A. ve Bojorquez, E. 2017. Reliability-based optimal load factors for seismic design of buildings. *Engineering Structures*, 151, 527-539.

Boscardin, J. T., Yepes, V. ve Kripka, M. 2019. Optimization of reinforced concrete building frames with automated grouping of columns. *Automation in Construction*, 104, 331-340.

Bybordiani, M. ve Azad, S. K. 2019. Optimum design of steel braced frames considering dynamic soil-structure interaction. *Structural and Multidisciplinary Optimization*, 60, 3: 1123-1137.

Camp, C. V. ve Huq, F. 2013. CO2 and cost optimization of reinforced concrete frames using a big bang-big crunch algorithm. *Engineering Structures*, 48, 363-372.

Carbas, S. 2016. Design optimization of steel frames using an enhanced firefly algorithm. *Engineering Optimization*, 48, 12: 2007-2025.

Changizi, N. ve Jalalpour, M. 2017. Stress-Based Topology Optimization of Steel-Frame Structures Using Members with Standard Cross Sections: Gradient-Based Approach. *Journal of Structural Engineering*, 143, 8:

Chen, C. L., Yousif, S. T., Najem, R. M., Abavisani, A., Pham, B. T., Wakil, K., Mohamad, E. T. ve Khorami, M. 2019. Optimum cost design of frames using genetic algorithms. *Steel and Composite Structures*, 30, 3: 293-304.

Daloglu, A. T., Artar, M., Ozgan, K. ve Karakas, A. I. 2018. Optimum Design of Braced Steel Space Frames including Soil-Structure Interaction via Teaching-Learning-Based Optimization and Harmony Search Algorithms. *Advances in Civil Engineering*,

Dede, T. 2018. Jaya algorithm to solve single objective size optimization problem for steel grillage structures. *Steel and Composite Structures*, 26, 2: 163-170.

Ding, Z. H., Li, J., Hao, H. ve Lu, Z. R. 2019. Structural damage identification with uncertain modelling error and measurement noise by clustering based tree seeds algorithm. *Engineering Structures*, 185, 301-314.

Dogan, E. 2014. Solving design optimization problems via hunting search algorithm with Levy flights. *Structural Engineering and Mechanics*, 52, 2: 351-368.

Dogan, E., Seker, S., Saka, M. P. ve Kozanoglu, C. 2018. Investigating the effect of joint behavior on the optimum design of steel frames via hunting search algorithm. *Advanced Steel Construction*, 14, 2: 166-183.

Eberhart, R. ve Kennedy, J. 1995. Particle swarm optimization. 4, Article Number, 1942-1948

Erdal, F. 2017. A firefly algorithm for optimum design of new-generation beams. *Engineering Optimization*, 49, 6: 915-931.

Erdal, F. ve Saka, M. P. 2013. Ultimate load carrying capacity of optimally designed steel cellular beams. *Journal of constructional steel research*, 80, 355-368.

Erdal, F., Tunca, O. ve Doğan, E. 2017. Optimum Design of Composite Corrugated Web Beams Using Hunting Search Algorithm. *International Journal Of Engineering & Applied Sciences*, 9, 2: 156-168.

Faghihmaleki, H., Nejati, F. ve Masoumi, H. 2017. Using the Concept of Neural Network in the Evaluation of Fragility Curve of Steel Moment Frame. *Jordan Journal of Civil Engineering*, 11, 4: 534-548.

Fernandez-Caban, P. L. ve Masters, F. J. 2018. Hybridizing particle swarm and big bang-big crunch optimization methods to explore then exploit the design domain of large planar frame structures. *Computers & Structures*, 202, 1-14.

Gholizadeh, S., Davoudi, H. ve Fattahi, F. 2017. Design of steel frames by an enhanced moth-flame optimization algorithm. *Steel and Composite Structures*, 24, 1: 129-140.

Gholizadeh, S. ve Milany, A. 2018. An improved fireworks algorithm for discrete sizing optimization of steel skeletal structures. *Engineering Optimization*, 50, 11: 1829-1849.

Gholizadeh, S. ve Mohammadi, M. 2017. Reliability-Based Seismic Optimization of Steel Frames by Metaheuristics and Neural Networks. *Asce-Asme Journal of Risk and Uncertainty in Engineering Systems Part a-Civil Engineering*, 3, 1:

Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13, 5: 533-549.

Goldberg, D. E. ve Holland, J. H. 1988. Genetic algorithms and machine learning.

Hadidi, A., Jasour, R. ve Rafiee, A. 2016. On the progressive collapse resistant optimal seismic design of steel frames. *Structural Engineering and Mechanics*, 60, 5: 761-779.

Hasancebi, O. 2017. Cost efficiency analyses of steel frameworks for economical design of multi-storey buildings. *Journal of Constructional Steel Research*, 128, 380-396.

Hasancebi, O. ve Carbas, S. 2014. Bat inspired algorithm for discrete size optimization of steel frames. *Advances in Engineering Software*, 67, 173-185.

Jalili, S., Hosseinzadeh, Y. ve Taghizadieh, N. 2016. A biogeography-based optimization for optimum discrete design of skeletal structures. *Engineering Optimization*, 48, 9: 1491-1514.

Karimi, F. ve Vaez, S. R. H. 2019. Two-stage optimal seismic design of steel moment frames using the LRFD-PBD method. *Journal of Constructional Steel Research*, 155, 77-89.

Kaveh, A. ve BolandGerami, A. 2017. Optimal design of large-scale space steel frames using cascade enhanced colliding body optimization. *Structural and Multidisciplinary Optimization*, 55, 1: 237-256.

Kaveh, A. ve Ghazaan, M. I. 2017. Optimum Design of Skeletal Structures Using PSO-Based Algorithms. *Periodica Polytechnica-Civil Engineering*, 61, 2: 184-195.

Kaveh, A., Vaez, S. R. H. ve Hosseini, P. 2017. Modified Dolphin Monitoring Operator for Weight Optimization of Frame Structures. *Periodica Polytechnica-Civil Engineering*, 61, 4: 770-779.

Kazemzadeh Azad, S. ve Hasancebi, O. 2013. Upper bound strategy for metaheuristic based design optimization of steel frames. *Advances in Engineering Software*, 57, 19-32.

Kociecki, M. ve Adeli, H. 2013. Two-phase genetic algorithm for size optimization of free-form steel space-frame roof structures. *Journal of Constructional Steel Research*, 90, 283-296.

Kociecki, M. ve Adeli, H. 2015. Shape optimization of free-form steel space-frame roof structures with complex geometries using evolutionary computing. *Engineering Applications of Artificial Intelligence*, 38, 168-182.

Le, L. A., Bui-Vinh, T., Ho-Huu, V. ve Nguyen-Thoi, T. 2017. An efficient coupled numerical method for reliability-based design optimization of steel frames. *Journal of Constructional Steel Research*, 138, 389-400.

Li, H. Y., Li, Z. H. ve Teng, J. 2016. A dynamic analysis algorithm for RC frames using parallel GPU strategies. *Computers and Concrete*, 18, 5: 1019-1039.

Lu, M. F. ve Ye, J. H. 2017. Guided genetic algorithm for dome optimization against instability with discrete variables. *Journal of Constructional Steel Research*, 139, 149-156.

Maheri, M. R., Askarian, M. ve Shojaee, S. 2016. Size and Topology Optimization of Trusses Using Hybrid Genetic-Particle Swarm Algorithms. *Iranian Journal of Science and Technology-Transactions of Civil Engineering*, 40, 3: 179-193.

Maheri, M. R., Shokrian, H. ve Narimani, M. M. 2017. An enhanced honey bee mating optimization algorithm for design of side sway steel frames. *Advances in Engineering Software*, 109, 62-72.

Moradi, S. ve Alam, M. S. 2017. Multi-criteria optimization of lateral load-drift response of posttensioned steel beam-column connections. *Engineering Structures*, 130, 180-197.

Oh, B. K., Park, J. S., Choi, S. W. ve Park, H. S. 2016. Design model for analysis of relationships among CO2 emissions, cost, and structural parameters in green building construction with composite columns. *Energy and Buildings*, 118, 301-315.

Pal, J., Banerjee, S., Chikermane, S. ve Banerji, P. 2017. Estimation of fixity factors of bolted joints in a steel frame structure using a vibration-based health monitoring technique. *International Journal of Steel Structures*, 17, 2: 593-607.

Pan, C. D., Yu, L., Chen, Z. P., Luo, W. F. ve Liu, H. L. 2016. A hybrid self-adaptive Firefly-Nelder-Mead algorithm for structural damage detection. *Smart Structures and Systems*, 17, 6: 957-980.

Pan, L. Q., He, C., Tian, Y., Su, Y. S. ve Zhang, X. Y. 2017. A region division based diversity maintaining approach for many-objective optimization. *Integrated Computer-Aided Engineering*, 24, 3: 279-296.

Papavasileiou, G. S. ve Charmpis, D. C. 2016. Seismic design optimization of multi-storey steel-concrete composite buildings. *Computers & Structures*, 170, 49-61.

Phan, D. T., Lim, J. B. P., Tanyimboh, T. T. ve Sha, W. 2017. Optimum design of cold-formed steel portal frame buildings including joint effects and secondary members. *International Journal of Steel Structures*, 17, 2: 427-442.

Pholdee, N., Bureerat, S. ve Yildiz, A. R. 2017. Hybrid real-code population-based incremental learning and differential evolution for many-objective optimisation of an automotive floor-frame. *International Journal of Vehicle Design*, 73, 1-3: 20-53.

Prendes-Gero, M. B., Alvarez-Fernandez, M. I., Lopez-Gayarre, F., Drouet, J. M. ve Junco, J. R. V. 2016. Cost optimization of structures using a genetic algorithm with Eugenic Evolutionary theory. *Structural and Multidisciplinary Optimization*, 54, 2: 199-213.

Qiao, S. F., Han, X. L., Zhou, K. M. ve Li, W. C. 2017. Conceptual configuration and seismic performance of high-rise steel braced frame. *Steel and Composite Structures*, 23, 2: 173-186.

Rahmzadeh, A., Ghassemieh, M., Park, Y. ve Abolmaali, A. 2016. Effect of stiffeners on steel plate shear wall systems. *Steel and Composite Structures*, 20, 3: 545-569.

Saka, M. P., Carbas, S., Aydogdu, I. ve Akın, A. 2016. Use of Swarm Intelligence in Structural Steel Design Optimization. 7, 43-73

Saka, M. P. ve Geem, Z. W. 2013. Mathematical and metaheuristic applications in design optimization of steel frame structures: an extensive review. *Mathematical problems in engineering*, 2013,

Saka, M. P. ve Geem, Z. W. 2013. Mathematical and Metaheuristic Applications in Design Optimization of Steel Frame Structures: An Extensive Review. *Mathematical Problems in Engineering*,

Shallan, O., Maaly, H. M., Sagiroglu, M. ve Hamdy, O. 2019. Design optimization of semi-rigid space steel frames with semi-rigid bases using biogeography-based optimization and genetic algorithms. *Structural Engineering and Mechanics*, 70, 2: 221-231.

Shen, T., Nagai, Y. ve Gao, C. 2019. Optimal Building Frame Column Design Based on the Genetic Algorithm. *Cmc-Computers Materials & Continua*, 58, 3: 641-651.

Truong, V. H. ve Kim, S. E. 2017. An efficient method for reliability-based design optimization of nonlinear inelastic steel space frames. *Structural and Multidisciplinary Optimization*, 56, 2: 331-351.

Turan, M. ve Doğan, E. 2015. Kavramsal Hidrolojik Modellerin Farklı Optimizasyon Algoritmaları İle Kalibrasyonu-Calibration of Conceptual Hydrological Model by Different Optimization Algorithms. *Celal Bayar Üniversitesi Fen Bilimleri Dergisi*, 11, 2:

Wrzesien, A. M., Phan, D. T., Lim, J. B. P., Lau, H. H., Hajirasouliha, I. ve Tan, C. S. 2016. Effect of stressed-skin action on optimal design of cold-formed steel square and rectangular-shaped portal frame buildings. *International Journal of Steel Structures*, 16, 2: 299-307.

Xu, Z., Lu, X. Z., Guan, H. ve Ren, A. Z. 2014. High-speed visualization of time-varying data in large-scale structural dynamic analyses with a GPU. *Automation in Construction*, 42, 90-99.

Yazdi, H. M. 2016. Implementing Designer's Preferences using Fuzzy Logic and Genetic Algorithm in Structural Optimization. *International Journal of Steel Structures*, 16, 3: 987-995.

Ye, J., Hajirasouliha, I., Becque, J. ve Pilakoutas, K. 2016. Development of more efficient cold-formed steel channel sections in bending. *Thin-Walled Structures*, 101, 1-13.

ÖZGEÇMİŞ

TEVFİK OĞUZ ÖRMECİOĞLU

Tevfik.oguz@gmail.com

ÖĞRENİM BİLGİLERİ

Yüksek Lisans 2018-2020	Akdeniz Üniversitesi Fen Fakültesi, İnşaat Mühendisliği Bölümü, Antalya
Lisans 2004-2006	Akdeniz Üniversitesi Fen Fakültesi, İnşaat Mühendisliği Bölümü, Antalya

MESLEKİ VE İDARİ GÖREVLER

Alt Yapıdan Sorumlu Uzman Mühendis 2018-2019	Antepe A.Ş.
Denizli-Tavas TİGH Şantiye Şefi 2016-2018	Yanartaş İnşaat Ltd.Şti
Ortak & Yapım Yönetimden Sorumlu Müdür 2007-2016	Örmecioğlu İnşaat Ltd.Şti

ESERLER

Ulusal bilimsel toplantılarda sunulan ve bildiri kitaplarında basılan bildiriler

1- • Tunca, O., Aydoğdu İ., Örmecioğlu, T., 2018, "Grasshopper-Genetic Optimization Algorithm Based Design of Structures" International Conference on Applied Mathematics, Istanbul.