İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

GPU ÜZERİNDE YAZILIM TABANLI ANTEN GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Abdullah BAKIRTAŞ

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Telekomünikasyon Mühendisliği Programı

Haziran 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

GPU ÜZERİNDE YAZILIM TABANLI ANTEN GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Abdullah BAKIRTAŞ
(504121302)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Telekomünikasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Selçuk PAKER

Haziran 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 504121302 numaralı Yüksek Lisans Öğrencisi **Abdullah BAKIRTAŞ**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"GPU ÜZERİNDE YAZILIMLI TABANLI ANTEN GERÇEKLENMESİ"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Selçuk PAKER** İstanbul Teknik Üniversitesi	.....
Eş Danışman :		.....
Jüri Üyeleri :	**Prof. Dr. Selçuk PAKER** İstanbul Teknik Üniversitesi	.....
	Doç. Dr. Mesut KARTAL İstanbul Teknik Üniversitesi	.....
	Prof. Dr. Osman PALAMUTÇUOĞULLARI Beykent Üniversitesi	.....
		.....
		.....

Teslim Tarihi : **04 Mayıs 2015**
Savunma Tarihi : **02 Haziran 2015**

ÖNSÖZ

Tezin fikrini veren, gerek mesai içerisinde gerek mesai sonrasında teorik yardımlarıyla destek olan başta tez danışmanım Prof. Dr. Selçuk Paker'e, tezi yazmadan önce benzer iş paketleriyle altyapımı oluşturduğum ve halen çalışmakta olduğum kurumumuz TÜBİTAK BİLGEM'e, mühendislik zekasından ilham aldığım, teknik bilgisiyle ve tecrübesiyle problem çözme yeteneğimi geliştirmemde büyük katkısı olan çalışma arkadaşım ve büyüğüm H. Ercüment Zorlu'ya, öğrenimimde her daim yanımda olan aileme teşekkür ederim.

Mayıs 2015

Abdullah Bakırtaş
(Telekomünikasyon Mühendisi)

İÇİNDEKİLER

Sayfa

KISALTMALAR

3GPP : 3rd Generation Partnership Project
3GPP2 : 3rd Generation Partnership Project 2
YTA : Yazılım Tabanlı Anten
SDA : Software-Defined Antenna
GPU : Graphics Processing Unit
CPU : Central Processing Unit
RF : Radio Frequency
PC : Personal Computer
FPGA : Field Programmable Gate Array
CUDA : Compute Unified Device Architecture
OpenCL : Open Computing Language
SDR : Software-Defined Radio
SDK : Software Development Kit
DF : Direction Finding
AWGN : Additive White Gaussian Noise
ATI : ATI Technologies (Ekran kartı)
Nvidia : Company of GPU
SP : Single Precision Floating Point
DP : Double Pecision Flotiong Point
GT/s : Giga Transfer per second
GFlop/s : Giga Floating point per second
TFlop/s : Tera Floating point per second
MB/s : Mega Byte per second

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

GPU ÜZERİNDE YAZILIM TABANLI ANTEN GERÇEKLENMESİ

ÖZET

Bu tezde, kablosuz haberleşme, uydu haberleşmesi, mobil iletişim, RADAR uygulamaları ve yeni nesil haberleşme sistemleri için genel amaçlı, programlanabilir, yazılım tabanlı anten (YTA) sistemi önerilmektedir. YTA siteminin bileşenleri, resiprok ve yönsüz antenler, alçaltıcı veya yükseltici RF elemanları, sayısallaştırıcı ve yazılım ile kontrol edilebilir demet katsayıları üreten ve işaretleri biçimlendiren GPU'dan oluşur. Bu tezde, anten elemanları, RF bloğu ve sayısallaştırıcı birimlerinin ideal olduğu kabul edilmiş ve bu bileşenlerin üzerinde herhangi bir çalışma yapılmamıştır. Tez kapsamında sunulan YTA sisteminin yazılım modülü, GPU donanımı üzerinde koşmakta ve CPU'lu YTA sistemi ile karşılaştırmalı olarak değerlendirilmektedir. Simülasyonlar ise MATLAB ile üretilen temel bant sentetik işaretler yardımıyla yapılmaktadır.

YTA sistemi, ayrık demet biçimlendirme metodunun farklı katsayı dizileri ile çok defa kombine edilmesiyle tasarlanmıştır. Kombinasyon tamamen yazılımsal olarak yapılmaktadır. YTA'nın programlanabilir olma özelliği, uyarlamalı ve akıllı algoritmaları gerçeklemeye imkan verdiği gibi esnek ve başarılı sistemlerin inşa edilmesine de olanak sağlamaktadır. YTA sistemi, aynı anda birden çok ışıma yönüne kanalize olabilme özelliği ile bütün kanalların verilerini gerçek zamanlı işleme yeteneğine sahiptir. YTA'nın işlem yükünün ve gerçek-zamanda çalışma zorunluluğu olan uygulamaların gereksinimlerinin karşılanması için GPU ve CPU kullanan çözümler bu tezde incelenmektedir. CPU için seri bir YTA algoritması, GPU için ise paralelleştirilmiş YTA algoritması tasarlanmıştır.

YTA algoritmasının GPU üzerindeki işlem süresi hesaplanırken üç farklı süre tanımlanmıştır. Birincisi CPU tarafından verilerin GPU'ya yüklenme süresi (t_u), ikincisi GPU'nun verileri işleme süresi (t_e), üçüncüsü ise GPU tarafından işlenen verilerin CPU'ya geri gönderilme süresidir (t_d). Toplam süre, seri olarak yapılan veri yükleme, veri işleme ve veri indirme sürelerinin toplamı olarak tanımlanmaktadır. CPU için ise toplam süre sadece veri işleme süresini içermektedir. GPU üzerindeki YTA algoritması, hızlı veri işlemesine rağmen verileri yükleme ve verileri geri indirme noktasında, yüksek hacimli transferler yapıldığından ve donanımlar arasında iletim bant genişliğinin sınırlamalarından dolayı yavaş kalmıştır. Buna rağmen, simülasyon sonuçları toplam sürenin çoğunlukla gerçek zaman eşiğinin altında kaldığını göstermektedir. Toplam harcanan sürede iyileştirme yapmak için çözüm olarak asenkron senaryo önerilmiş; fakat bu tez kapsamında asenkron senaryo simüle edilmemiştir. GPU üzerinde koşan algoritma daha hızlı performans gösterdiği ve bu sayede geniş bantlı verileri bile gerçek zaman eşiğinin altında işleyebildiği gözlemlenmiştir. CPU algoritması ise yavaş kalmış ve fazla kaynak harcamasına rağmen verileri gerçek zamanda işleyememiştir.

GPU'lar yüksek hacimdeki verilerin hızlı bir şekilde işlenmesine olanak sağlarmaktadır. Veri hacminin büyüklüğü, GPU'da koşan YTA algoritmasının

performansını da etkileyen bir unsurdur. Bu nedenle, YTA algoritmasının işleyeceği verinin büyüklüğü bir çerçeve boyutu olarak tanımlanmıştır. Çerçeve boyutunun mertebesi önce kaba bir yaklaşımla belirlenmiş, belirlenen değer yakınında çerçeve boyutunun hassas değişimlere tepkisi ölçülmüştür. GPU algoritmasının performansı, çerçeve boyutunun artması ile bir noktaya kadar artmış sonrasında ise fazlaca değişiklik göstermemiştir. Performansının kötü olduğu çoğu durumda bile verileri gerçek-zamanda işleyebilmiştir. Bu durum GPU ile gerçeklenen YTA sisteminin başka sistemlere entegrasyonun ve adaptasyonun hızlı ve başarılı olacağının göstergesidir.

Sonuç olarak; GPU üzerinde koşan YTA algoritmasının CPU üzerinde koşan algoritmaya göre önemli derecede toplam işlem süresini azalttığı gözlemlenmiştir. GPU üzerindeki YTA sisteminin ölçeklenebilir olması nedeniyle, GPU'nun gelişme hızı ve maliyet etkin bir çözüm olması da göz önüne alındığında, yeni nesil haberleşme sistemlerinde ve radar uygulamalarında yaygın bir kullanım alanına sahip olacağı düşünülmektedir.

REALIZATION OF SOFTWARE-DEFINED ANTENNA ON GPU

SUMMARY

In recent years, digital circuits have become more popular compared to analog circuits since it is possible to implement complex systems with lower cost and effort. Field upgradability, scalability, energy efficiency are the most attractive advantages for signal processing used in digital solutions. However, complex digital systems require more advanced signal processing capability. Beamforming in telecommunication systems had been traditionally implemented using analog systems. Nowadays, beamforming is usually implemented on a variety of platforms including specialized DSP's, commodity CPU's, and FPGA's. Beamforming implementation running on GPU processing is a new trend on this area since the parallel processing capability of GPUs allows implementing advanced beamforming algorithms.

A software-defined antenna (SDA) system allows digitally constructed multiple beam responses with different orientation. Construction of these beams is realized on software and provides the flexibility of implementing alternative algorithms, field upgrades, and reduced dependency on hardware. However, digital beamforming comes with the price of high degree of computational complexity. The main contribution of this thesis is to implement an SDA system by leveraging GPUs. In order to demonstrate the performance improvement provided by a parallel algorithm optimized for GPUs, we also implemented multiple beam structure using a serial algorithm for high clock frequency CPUs.

SDA is commonly known as smart antenna or adaptive antenna array, because SDA allows implementation of smart algorithms. Smart antenna technology or adaptive antenna array aim towards wireless communication, mobile communication, satellite communication, Electronic Warfare applications with emphasis on RADAR and new generation of communication systems. Smart antenna provides higher dynamic range, multi-path variation, lower interference, increasing capacity and data rate, and smart-jamming. In mobile communication systems, smart antennas have been suggested to solve interference problem between users and increase coverage capacity. Smart antenna have also been suggested for new generation communication systems which require space-time multiplexing access (SDMA), because antenna array have capability to divide space. Another field of smart antenna on satellite communication is directed-beam provide gain for signals between satellite to ground station. Adaptive directed-beam can also be used to follow satellites which display asynchronous behavior to the ground. Moreover, adaptive antenna array provides electronic scanning for RADAR application in civil or military applications. Using antenna array for electronic scanning enables fast and efficient solution.

Smart antenna or adaptive antenna array have been improved toward GPUs parallel processing capability. We must emphasize GPUs provide cost-effective solution about SDA. The rapid evolution of GPUs which is originally fueled by gaming enthusiasts, got attention from scientific fields with computational problems because

of its high parallel architecture and as a result GPUs are extensively used in computational finance, defense and intelligence, machine learning, manufacturing, media and entertainment, safety and security fields. Today there are GPUs specifically designed for computational problems, leveraging similar architectures with gaming GPUs. In telecommunication field, some algorithms of digital signal processing definitely require parallel structure is derived to GPUs. An example is receiver antenna model of Global Positioning System (GPS) which require to direct 4 satellite at the same time. Its performance depends on directed-beam resolution and reduction of interferences. Researchers from Stanford University implemented and simulated the antenna model shows efficient performance on GPUs[5]. This is good scenario to sign GPUs on SDA's fields. Another example about new generation base station which require dividing space to increase capacity and data rate. Some researches show that SDA system on GPU allows reduced radiation power of base stations.

There is a misconception around GPUs as being cheaper alternatives for FPGAs because of their highly parallelized architecture. However, FPGAs are more suited for hard real-time problems with no tolerance to latency or jitter in processing times. They also provide very low level control on hardware behavior which enables design of many core soft processors, GPIO capabilities and standalone operation. GPUs, on the other hand, work as accelerators for CPUs, providing far greater efficiency comparing to CPUs at parallelizable tasks. FPGAs too, can be accelerators for CPUs, however, being both an accelerator and a gaming device, GPUs provide easier programming paradigms and connection interfaces. They also provide cost effectiveness, because of the economies-of-scale.

Graphic cards have high computational capacity to compute a scenario. To use this capacity code blocks called “kernel” are initiated by host CPUs. Kernels load data by high speed memory transfers from host to kernel and fetches the results from kernel to host. This cooperation of GPU and CPU is called hybrid programming in the literature. As is seen, time spent on a computation on GPU consists of the duration of data transfer from CPU to GPU (t_u), time spent on execution of kernel (t_e), and duration of data transfer for results from GPU to CPU (t_d). These three times are the main parameters observed for measuring the performance of a specific algorithm on GPU. In the case of IO bound algorithms with large amount of data to copy from host CPU to GPU, tu and td may be dominant. In order to utilize memory bandwidth of the GPU efficiently, there are asynchronous copy mechanisms, leveraging hardware accelerated DMA transfers while doing computations on the GPU, however in the simulations these mechanisms are not implemented and performance analysis is provided as GPU process only duration (t_e) and GPU total duration ($t_u + t_e + t_d$).

GPUs have different memory types, each having differing characteristics. Global memory has high capacity bandwidth and it communicates to host from PCI Express slot. Its performance affects t_u and t_d. There are other global memories which are called constant and texture memory. These memories are read only and they have limited sources. Programmers must pay attention on memory types while developing GPU based algorithms. Another memory type is shared memory which is also called block memory. Any thread in same block can access shared memory. The last memory type is register which is private memory for threads. Registers can only be accessed by their own thread. Registers have the smallest latency among the memory types.

There are two mainstream programming languages for GPUs: NVIDIA specific CUDA C (CUDA) and hardware independent OpenCL. Since CUDA lets the developers to have lower level access to GPU resources, CUDA is selected as the primary programming language, in the scope of this thesis.

Proposed SDA is derived from a receiver scenario (which could easily be a transceiver one). Basic components of the SDA are a reciprocal and omni-directional antenna array, a down-converter, RF elements, a set of Analog to Digital Converters (ADC) and a software-defined beam-former module. Consisting of the thesis excepts to design and analyze all RF components. Only beam-former module is designed and realized on CPUs and GPUs. It is assumed that there is no reflection, refraction, scattering and noise in free space and SDA components. The antenna array is modeled as linear antennas; however, other antenna models can be introduced as well. Beam responses are calculated using by phase steering coefficients. Finally, synthetic base-band signals were introduced on MATLAB to observe simulation results.

In the scope of this thesis, a baseband signal in receiver side and an SDA receiver system are modeled and beam responses are derived from the baseband signal model. Using these models, SDA is directed to a particular angle and a very sharp beam is created. In order to create multiple beams, multiple beam coefficient sets are created. Another parameter affecting the beam created by the SDA is the number of antenna elements. Thesis, elaborates on the effect of these parameter in the following sections. With careful selection of beam coefficients and powerful computation devices like GPUs, an SDA is capable of providing multiple sharp beams using the same set of antennas.

Computational complexity of digital beamforming depends on sampling frequency, size of antenna elements and sampling resolution. SDA system has number of created beam size times more computing complexity than unique digital beamforming. It is extracted from analytical equation. To compute response of SDA have two algorithms which are designed as GPU algorithm and CPU algorithm. GPU algorithm have one cycle less than CPU algorithm and the cycle have biggest counter. It shows that GPU algorithm is the good solution of parallel processing. However, every digital circuit has limited sources so we cannot expect to eliminate this cycle duration at simulation result. In this work, two types of simulation introduced to perform CPU and GPU algorithms on their devices. One shows only processing time (t_e) and the other simulation shows total processing time ($t_u+t_e+t_d$) for GPU versus CPU. According to simulation result, GPU perform much more fast respond from CPU. Its details are executed in section 4.

GPU based algorithm computes one frame at a time. To improve GPU algorithm performance, frame size must be analyzed to find the optimum frame size. First, coarse frame size is selected by power of 2 (2n) and GPU algorithm is simulated. According to simulation result, increasing frame size performed better response until a point which is deep simulation time, after this point frame size does not affect the performance. Fine frame size is predicted with a second simulation. In this simulation, frame size is changed linearly with one-sample steps. Initial frame size is selected by the best coarse frame length. According to this simulation result didn't show significant changes. Thus, we show relation of frame size and processing time. Flexible frame sizes of the algorithm provides further integration possibilities and comes with trade-offs. Larger frame size provides higher performance (e.g. sharper

beam, increased number of beams), higher energy efficiency (e.g. same amount of data, same number of clock cycles, means the same energy required but systems creates better beams) but it introduces latency on the system (e.g. more data required to process and more data is collected within a larger time frame), on the other hand, small frame lengths provide small delays but lower performance, thus lower energy efficiency.

In section 6, for the simulation work, software is implemented to work on a Windows based PC. Software has an interface for SDA pattern visualization and required simulation data can be calculated on this interface. Data of antenna array have to fix determined name and format (see figure 6.2). The software processes the calculated data within a small time frame-short enough for real-time- for required duration. Before using this software, every hardware has to check for qualification for example brand name, GPU compute capability and other CUDA properties.

As a result; SDA on GPU have significant improvement to reduce total processing time comparing to SDA on CPU. Moreover, the scalable algorithm structure would provide further performance gains with future GPUs, leading to easily upgradable and cost effective solutions. In conclusion, SDA have significant performance on GPU. GPU algorithm processing this antenna is much more faster than CPU algorithm. Moreover, GPUs are provided to implement more complexity algorithm versus CPUs. Evolution of GPUs are expected to get higher performance of scalable algorithms.

1. GİRİŞ

Yeni nesil kablosuz haberleşme sistemlerinin önemli bir bileşeni olan akıllı anten teknolojisi [1], RADAR uygulamalarında, mobil iletişimde, uydu haberleşmesinde araştırma konusu olmuş, hatta bazı uygulamalar için standartlarda yerini almıştır. Çünkü akıllı antenler, işaret dinamik aralığını genişletir, çok-yollu çeşitleme sağlar, aynı frekanstaki girişimleri azaltır, kapasite ve veri hızı artırımı sağlar [2-3]. Akıllı antenler, aynı frekanstaki yayınların girişimini azalttığı gibi istenmeyen yönlerdeki işaretleri bastırma (smart-jamming) özelliğine de sahiptir [4-5]. 3G mobil haberleşme sistemlerinde kapasiteyi etkileyen en önemli faktörlerden biri de kullanıcıların aynı frekansı kullanmalarının girişime neden olmasıdır. Akıllı antenler, kapasite ve kapsama alanını genişletmek için, 3GPP ve 3GPP2 standartlarında önerilmiştir [6]. Yeni nesil haberleşme sistemlerinde, uzay bölmeli çoğullama (SDMA) veya çok yollu çeşitleme teknikleri gerektiren uygulamalar için de akıllı antenler çözüm olarak önerilebilir.

Akıllı anten, uydu haberleşmesi için önemli bir kaynak yaratmaktadır [7]. Yeryüzü ile uydu arasında doğrudan görüş olduğunu düşünürsek, yönlendirilmiş dalga biçimi, bu haberleşmede avantaj sağlayacaktır. Ayrıca uydu işaretleri, yeryüzüne çok zayıf ulaştığından girişime karşı çok hassastır. Yönlendirilmiş anten dizisi, zayıf işaretlerin alınması için de çözüm olarak düşünülebilir. Uyarlamalı akıllı anten dizisinin uydu haberleşmesindeki bir başka faydası da dünya ile senkron hareket etmeyen uyduları takip edebilme yeteneği kazandırmasıdır.

Uyarlamalı anten dizisi, RADAR uygulamalarında elektronik tarama imkanı sunmaktadır [8]. Elektronik tarama, daha hassas ve hızlı hedef tespitine imkan vermektedir. Bu uygulamalarda hassas ölçümler elde etmek için çok fazla sayıda anten elemanı gereklidir. Anten elemanı sayılarının yüksek boyutlara ulaşması, yoğun hesaplama yükünü de beraberinde getirmektedir.

Uyarlamalı anten dizileri, sivil ve askeri uygulamaların da ilgisini çekmiştir. Spektrum gözetleme sistemleri (Spectrum Monitoring System) için bazı yön bulma

(Direction Finding) algoritmaları, bir dizi anten ve uyarlamalı demet katsayıları ile yapılabilmektedir.

1.1 Haberleşme alanında GPU

Bir grafiğin kalitesini belirleyen faktörler, boyutu, renk çözünürlüğü, video için çerçeve yenileme hızı gibi sayısal değerlerdir. Son yıllarda grafik kalitesi, çok yüksek boyutlara ulaşmıştır. Aslında, gelişen oyun grafikleri de GPU'ların ne kadar güçlendiğinin bir diğer göstergesidir. Grafik kalitesinin artması, görüntü işleme için gereken çarpma ve toplama gibi matematiksel operasyonların artmasını gerektirir. Yüksek kalitedeki grafikler, modern GPU'lar sayesinde kolayca işlenebilmektedir. Günümüzde bu GPU'lara erişmek ise oldukça kolaydır. Masaüstü bilgisayarlarda, iş istasyonlarında, gömülü sistemlerde, tabletlerde ve cep telefonlarında bulunan GPU'lar, ulaşılabilir, ucuz ve değiştirilebilir cihazlardır.

GPU, her ne kadar grafik işlemcisi olarak geliştirilse de, başka uygulamalar için yeni eğilimler yaratmıştır. Özellikle hesaplamalı bilimlerde paralel algoritmalar, GPU ile hızlı ve efektif olarak işlenebilmektedir. Stanford Üniversitesinde yapılan bir çalışmada GPU, bir dizi anten elamanı ile gerçeklenen GPS alıcısına, gerçek zamanlı veri işleme yeteneği kazandırmıştır. [5]. Bir başka çalışmada ise GPU, yeni nesil baz istasyonları için çözüm olarak önerilmektedir. GPU ile gerçeklenen 4 elemanlı biçimlendirilmiş anten modeli, baz istasyonunun gerektirdiği yayın gücünü azaltmaktadır [9].

1.2 Motivasyon

Yazılım tabanlı anten (YTA), akıllı algoritmaları uygulanabilir kılmasından dolayı literatürde uyarlamalı anten dizileri ya da akıllı anten olarak da anılmaktadır [10]. YTA, bir dizi anteni, yazılımsal olarak gerçekleyerek arzu edilen ışıma örüntüsünü oluşturmaktadır. YTA'nın uyarlamalı ve programlanabilir özelliği, kayda değer bir esneklik sağlamaktadır. Herhangi bir donanımsal değişiklik gerektirmeksizin anten dizisi, kolayca, başka yönlere yönlendirilebilir; böylece donanım bağımlılığı azaltılabilir ve daha esnek ve başarılı sistemlerin inşa edilmesine de olanak sağlanmış olur.

YTA, ayrık demet biçimlendirme algoritmasının farklı katsayı dizileri ile çok defa kombine edilmesiyle tasarlanmıştır. Dolayısıyla ayrık demet biçimlendirme yönteminin gerektirdiği sistem maliyetleri, tasarladığımız YTA'nın da sistem maliyetlerini etkilemektedir. YTA, ayrık demet biçimlendirme algoritmasına çarpan olarak biçimlendirilmiş ışıma cevabı kadar daha hesaplama maliyeti getirmektedir. Çünkü YTA, aynı anda birden çok ışıma yönüne kanalize olabilmekte ve her kanalın verilerini, gerçek zaman süresinde işleme gereksinimine sahiptir. Ayrıca YTA, çok amaçlı ve çok fonksiyonlu bir yapı baz alınarak tasarlandığı için geniş bantlı verilerin işlenebilmesini de hedeflemiştir. Dolayısıyla, örnekleme hızı da YTA'nın performansını belirlemektedir. O halde YTA, çok yüksek boyutlarda işlem yapabilme kabiliyetine sahip olması gerekir.

YTA'nın işlem yükünün karşılanması ve gerçek-zamanlı çalışabilmesi için GPU çözüm olarak önerilmiştir. Nihayetinde YTA modeli için paralel bir algoritma tasarlanması mümkün görülmüş ve bu algoritma GPU'nun paralel işlem yapabilme kabiliyeti ile örtüştürülmüştür.

1.3 Tezin sınırları ve literatüre katkısı

Bu tezde, kablosuz haberleşme, uydu haberleşmesi, RADAR uygulamaları ve yeni nesil haberleşme sistemlerinin sistem maliyetlerini düşürmek için tasarlanan YTA sistemi önerilmektedir. YTA sisteminin bileşenleri, resiprok ve yönsüz antenler, alçaltıcı ve ya yükseltici RF elemanları, sayısallaştırıcı ve yazılım ile kontrol edilebilir demet katsayıları üreten ve işaretleri biçimlendiren GPU modülüdür. Bu tezde, anten elemanları, RF bloğu ve sayısallaştırıcı birimlerinin ideal olduğu kabul edilmiş ve bu blokların üzerinde herhangi bir tasarım ve ya analiz yapılmamıştır. Dolayısıyla bu elemanların performansları değerlendirilmemiştir. Sayısallaştırıcıların senkronizasyon problemleri de tezin kapsamının dışındadır. Yazılım modülü ise hem GPU hem de CPU üzerinde tasarlanmıştır; performansları, MATLAB ile üretilen temel bant sentetik işaretler ile test edilmiştir.

YTA sistemi, çok amaçlı ve çok fonksiyonlu ucuz, esnek ve güçlü bir anten tasarımı sunmuştur. YTA sistemi, donanım bağımlılıkları dışında, kayda değer yazılımsal kural gerektirmeksizin çeşitli demet katsayıları ile kombine edilerek çok sayıda işaret biçimlendirebilir ve geniş bantlı işaretleri, gerçek-zamanda işleyebilir. GPU'nun paralel işlem yapabilme kabiliyetlerinden faydalanmak için YTA algoritması,

paralelleştirilmiş ve YTA sistemine gerçek-zamanda işlem yapabilme kabiliyeti kazandırılmıştır. Yine de YTA sistemi için işaret bant genişliği, anten elemanı sayısı, demet biçimlendirme boyutunun büyüklüğü ve örnekleme çözünürlüğü performans sınırlayıcı etken olarak değerlendirilmelidir. Çünkü bu parametreler, işlem yükünü artıran ve ya azaltan etkiye sahiptir.

GPU programlama ile YTA'nın hem tasarım aşaması kısalmış hem de tasarımdan sonra sistem entegrasyonu için kolay erişilebilir donanım sağlanmıştır. GPU bu özelliği ile FPGA kartlarının alternatifi olabilir. Çünkü FPGA programlama, hem pahalı hem de tasarım süreci uzun zaman süren programlamadır. Neticede FPGA programlama ile gerçeklenen tasarımın performansının iyi planlanmayan çoğu durumlarda düşük olduğu da bilinmektedir. Ayrıca GPU'nun YTA algoritmasına kazandırdığı paralel işlem gücü de PC'nin ana işlemcisi olan CPU'nun performansından çok daha üstündür. Bu nedenlerden dolayı YTA sistemi için GPU'lu çözüm yeni, basit, ucuz ve kolayca test edilebilir bir eğilim yaratmıştır.

YTA, donanımsal değişiklere uyarlamalı çözümler üretebilir. Verilerin gerçek zaman eşiğinin altında işlemesi için örnekleme hızını düşürebilir (decimation), biçimlendirilmiş ışıma boyutunu azaltabilir ve ya veri aldığı anten sayısını azaltabilir. YTA, SDR [14] alıcıları ile kolayca entegre olabilir ve geniş bantlı verilerle gerçek-zamanlı uygulamalara imkan verir. Ayrıca YTA, aynı anda faklı bant genişliğinde ve farklı frekanslardaki işaretleri de işleyebilme kabiliyetine de sahiptir.

Bu tez kapsamında önerilen YTA, yön bulma (DF) konularının ilgi alanında olan anten demeti biçimlendirme kabiliyetine sahip olmasına rağmen yer kestirme (source localization), kaynak ayrıştırma (source seperation) gibi DF konularıyla ilgilenmez. YTA, demet biçimlendirme katsayılarının tanımlanmasına ve değiştirilmesine izin verir; ancak katsayı hesaplanması ve güncellenmesi YTA'nın ilgi odağında değildir. Yani YTA, akıllı, uyarlamalı anten dizileri için gerekli altyapıyı verir; ancak akıllı algoritmaların gerçeklenmesi ile ilgilenmez.

1.4 Tezin akışı

Bu tezin içeriğinde önerilen YTA, ayrık demet biçimlendirme modeli baz alınarak tasarlanmıştır. 2. bölümde ayrık demet biçimlendirmenin analog demet biçimlendirmeye üstünlüğünden kısaca bahsedilmektedir. Sonrasında, ayrık demet

biçimlendirme modeli baz alınarak YTA modeli türetilmektedir. YTA'nın ışıma örüntüsü, faz tarayıcı anten dizisi ile çıkartılmaktadır. 3. bölümde, GPU'nun hesaplamalı bilimlerdeki yerinden, işlem yapabilme kabiliyetinden; heterojen programlamadan ve CUDA mimarisinden genel olarak bahsedilmektedir. 4. bölümde ise YTA algoritması, hem CPU üzerinde hem de GPU üzerinde gerçeklenmektir. Bu algoritmaların performansları 5. bölümde karşılaştırılmaktadır. Sonrasında GPU'lu YTA sisteminin performansı kendi içerisinde tekrardan değerlendirilmektedir. 6. bölümde ise YTA için tasarlanan bir arayüz tanıtılmaktadır. Sonuç ve öneriler 7. bölümdedir.

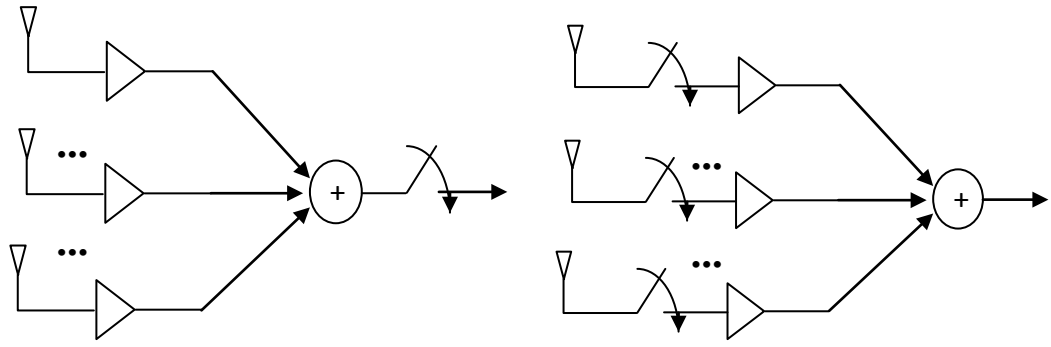
2. YAZILIMSAL ANTEN MODELİ

2.1 Amaç

Resiprok ve yönsüz anten dizisi ile yazılım tabanlı anten tasarlanacaktır. YTA modeli, ayrık demet biçimlendirme algoritmasından faydalanılarak türetilmiştir. YTA alıcı olarak tasarlanmıştır; ancak küçük değişikler ile verici özelliği de kazanabilir. Bu bölümdeki hesaplamalar alıcı model üzerinden analitik olarak yapılacaktır.

2.2 Analog ve ayrık demet biçimlendirme

Analog demet biçimlendirme, anten elemanları tarafından alınan işaretlerin örneklenmeden önce bir dizi analog faz düzeltme elemanları ile çarpılıp, analog toplayıcı ile toplanması sonucu oluşan biçimlendirme şeklidir. İşaret bu aşamadan sonra örneklenerek sayısallaştırılabilir. Ayrık demet biçimlendirme ise anten elemanları tarafından alınan işaretlerin ayrı ayrı örneklenmesinden sonra gerçekleşir. Ayrık demet biçimlendirmedeki çarpıcı elemanlar ve toplayıcı eleman, sayısal donanımlardır. Her iki demet biçimlendirme için söylenilen bu ifadelerin kabaca karşılığı Şekil 2.1'de gösterilmektedir.



Şekil 2.1 : Analog(solda) ve ayrık(sağda) demet biçimlendirme (Genel bakış).

Şekil 2.1'e göre ayrık demet biçimlendirme daha fazla sayısallaştırıcı (ADC) gerektirir. Bu durumun bir dezavantaj olduğu düşünülebilir; ancak ADC'lerden sonra görülen sayısal çarpıcılar ve toplayıcı günümüzde ucuz, efektif ve esnek sonuçlar

verebilmektedir. Ali Akbarian'ın bir makalesinde, temel bant demet biçimlendirme için analog ve ayrık tasarımları, donanımsal olarak karşılaştırılmıştır [11]. Ayrık demet biçimlendirmeyi FPGA kartıyla, analog biçimlendirmeyi ise basit analog devreleri ile gerçeklemiştir. Ayrık demet biçimlendirmenin daha fazla maliyet getirmesine karşı; daha esnek bir yapı ortaya çıkardığını söylemektedir. Zaten sayısal çözümlerin daha esnek bir yapı çıkarması muhtemeldir.

Analog devrelerin çalışma aralıkları, belli kesim frekansları ile sınırlanabilir ya da analog devre elemanları frekans seçici davranabilir. Dolayısıyla analog devre elemanlarının sayısı arttıkça, sistem esnekliğini kaybeder. Ancak ayrık işaretler için filtreleme gibi sitem bileşenleri daha esnek tasarımlar ile gerçeklenebilir. Dolayısıyla ayrık demet biçimlendirmenin çalışma frekans aralığının daha geniş olması muhtemeldir. Ayrık demet biçimlendirmenin sistem maliyetlerini artırması da her koşulda geçerli değildir. Çünkü analog devre elemanlarının hassasiyetleri, artıkça maliyetleri de çok hızlı bir şekilde artmaktadır. Sayısal devre elemanlarının maliyetleri ise hassasiyetlerine göre daha makul düzeyde olabilmektedir.

Ayrık demet biçimlendirmenin bir başka üstünlüğü, esnek kontrol edilebilir demet katsayılarına sahip olmasıdır. Dolayısıyla uyarlamalı bir biçimlendirmeye olanak sağlar. Analog demet biçimlendirmede ise akıllı ve uyarlamalı bir tasarım kurgulamak neredeyse imkansızdır.

Sadece ayrık işaretlerin kopyalanabileceğini, analog işaretlerin ise bölücüler ile çoğullanabileceğini belirtmek gerekir. Ayrık demet biçimlendirmenin bu tez için en önemli üstünlüğü ise aynı kanallardaki işaretler ile birden fazla ışıma demet cevabı üretmesine olanak sağlamasıdır. Dolayısıyla, ışıma demet cevaplarının çoğullanması ayrık demet biçimlendirme yapısı ile mümkündür. Bu çoğullama yalnızca işlem maliyeti getirecektir. Daha sonraki bölümlerde, GPU, işlem maliyetinin çözümü olarak önerilecektir. Analog tasarımın böyle bir senaryoya cevap verebilmesi maliyetlidir. Analog çoğullayıcılar sistemin büyümesine neden olduğu gibi hem sistemin tükettiği gücü arttırır, hem de her bir bölücü elemanının eklenmesiyle beraber yeni kayıplar ortaya çıkartır.

2.3 Temel bant işaret

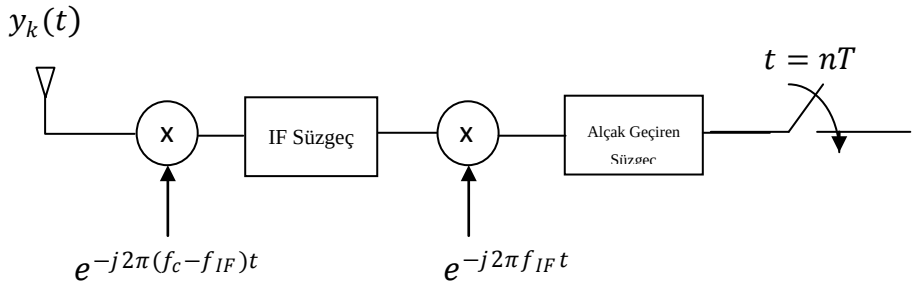
Temel banttaki bir mesaj işareti radyo frekansına çıkartılsın. En genel halde bu ifade;

$$y(t) = x(t)\, e^{j2\pi f_c t} \qquad (\mathbf{2.1})$$

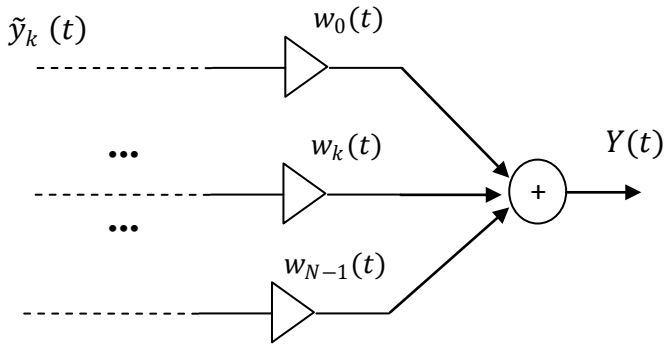
eşitliği ile gösterilir. Bu eşitlikteki $x(t)$ işareti, temel bant IQ modülatör çıkışından ve ya analog işaret üreten temel bant kaynaktan alınmış olabilir. $e^{j2\pi f_c t}$ kompleks sinüsoidal bileşenlerden oluşan taşıyıcı işaret, $y(t)$ ise bant geçiren işarettir. $y(t)$ bant geçiren işaret, hiçbir kanal gürültüsü olmayan bir uzayda, bozulmaya maruz kalmadan yayıldığını kabul edelim. Yeterince uzaklıkta ve anten dizilerini içeren sınırlı bir yüzeyde, $y(t)$'nin dalga yüzlerinden alınan kesitin düzlem dalga formunda olduğunu kabul edelim. Bu sınırlamalar altında alınan işareti modelleyeceğiz.

2.4 Alıcı yapısı

Her bir anten elemanı tarafından alınan işaret, Şekil 2.2'deki alıcı yapısını takip ettiği kabul edilmiştir. Bu yapıya göre, alınan işaret, öncelikle kompleks sinusoidal işaret ile çarpılarak IF bandına, sonrasında ise temel banda indirilip örneklenir. Böylece, temel bant işaret reel ve sanal bileşenlerle elde edilir. Sonrasında, her bir alıcı çıkışındaki temel bant işaret, Şekil 2.3'deki yapıyı takip eder. Bu yapıya göre sayısallaştırılmış işaretler önce $w_k(n)$ katsayılarıyla çarpılır ve ardından tek bir işarette toplanır.



Şekil 2.2 : k. anten elemanının alıcı yapısı



Şekil 2.3 : Ayrık biçimlendirme blok yapısı

Şekil 2.3'deki blok yapısı aslında ayrık demet biçimlendirme blok yapısıdır. $w_k(t)$ demet katsayılarıdır. Demet katsayıları, zamandan bağımsız değişmeyen ya da zamanla değişen katsayılar olabilir. $Y(t)$ ise demet cevabıdır. Demet cevabı, sayısallaştırılmış işaretin Şekil 2.3'deki blok yapısında kombine edilerek elde edilir.

2.1 denklemindeki bant geçiren işaret alıcı tarafına zamanda gecikerek ulaşır. Her anten elemanı için bu ifade;

$$y_k(t) = x(t - \Delta t_k)\, e^{j\,2\pi f_c (t-\Delta t_k)} \qquad (\mathbf{2.2})$$

şeklinde gösterilebilir. k indisi, 0 dan $N\text{-}1$'e kadar değişen anten elemanlarını göstermektedir. $y_k(t)$ her bir antenin aldığı işareti göstermektedir. Δt_k her bir işaretin zamanda gecikmesini göstermektedir. 2.2 denkleminden görüldüğü üzere, zaman gecikmesi, hem temel bant işaretin zaman bölgesinde hem de f_c taşıyıcı frekansı ile çarpılan terimde görülmektedir. Mesaj işaretinin bandının çok dar ve sınırlı olması koşulu altında, gecikmenin sadece taşıyıcı frekansına bağlı olduğu kabul edilecektir.

Şekil 2.2'deki yapıya göre her bir alıcı çıkışında elde edilen temel bant işaret ise şu şekildedir.

$$\tilde{y}_k\,(t) = x(t - \Delta t_k)\, e^{-j\,2\pi f_c \Delta t_k} \qquad (\mathbf{2.3})$$

2.3 denklemi, 2.2 denkleminden taşıyıcı çıkartılarak elde edilmiştir.

2.5 Faz kayması

2.3 denklemindeki ifadeyi ele alalım. Bu ifadedeki $\tilde{y}_k\,(t)$'nin fourier dönüşümü

$$\tilde{Y}_k(f)\; = X(f)\, e^{-j\,2\pi f_c \Delta t_k}\; e^{-j\,2\pi f \Delta t_k} \qquad (\mathbf{2.4})$$

şeklindedir. $X(f)$'in yanındaki üstel ifadeler tek bir yerde toplandığında 2.5 denklemi elde edilir.

$$\tilde{Y}_k(f) = X(f)\, e^{-j\,2\pi (f_c + f)\Delta t_k} \qquad (\mathbf{2.5})$$

$x(t)$ işaretini bant sınırlı ve dar bantlı bir işaret olarak kabul etmiştik. Bu durumda denklem 2.5'deki f frekansı, işaretin bant genişliği ile sınırlı kalacaktır. İşaretin bant genişliğini Bw ile sınırlayalım. Bu durumda üstel ifadenin fazı

$$2\pi(f_c + f)\Delta t_k \;\overline{<}\; 2\pi(f_c + b)\Delta t_k \qquad (\mathbf{2.6})$$

eşitsizliği ile gösterilsin. Ayrıca Δt_k'nın $2\pi(f_c + Bw)$ çarpanına göre çok daha küçük olduğunu, dolayısıyla faz teriminin [0 1) aralığında kaldığını kabul edelim. Bu durumda kompleks üstel ifade osilasyona neden olmayacaktır. f_c taşıyıcı frekansı bant sınırlı $x(t)$ işaretinin band genişliğinden çok daha büyük olduğu varsayılmaktadır $(f_c \gg Bw)$. O halde 2.6 eşitsizliğindeki faz terimi, f_c taşıyısı frekansının etkisi altındadır.

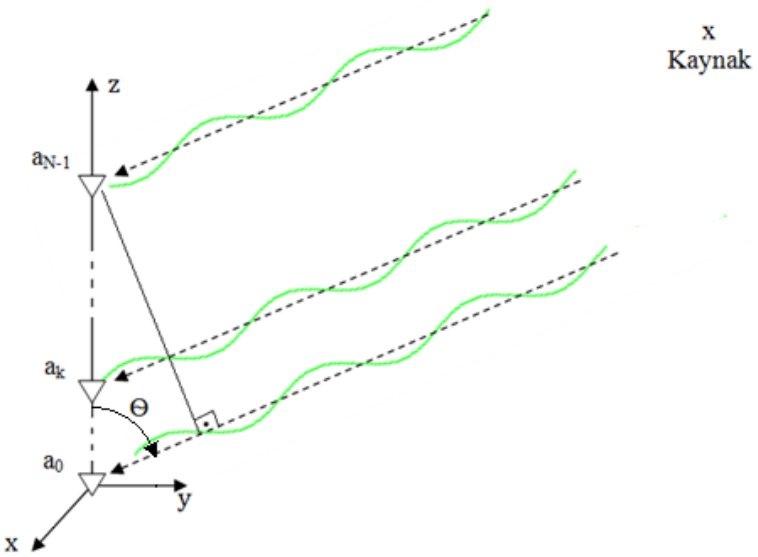
2.6 Demet katsayıları

Öncelikle belirtmek gerekir ki; bu tez, demet katsayısı hesaplama üzerine yoğunlaşmamıştır. Demet katsayıları, 2.5 denklemindeki faz terimini yok etmek için ve zamandan bağımsız olarak seçilmiştir. Mesaj işaretinden kaynaklanan faz kayması ihmal edildiğinde demet katsayıları 2.6 eşitliğine seçilebilir.

$$w_k = e^{j2\pi f_c \Delta t_k} \qquad (\mathbf{2.7})$$

2.7 Lineer anten modeli için gecikmelerin hesaplanması

Lineer anten modeli, anten elemanlarının doğrusal bir çizgi boyunca dizilmesiyle oluşturulan modeldir. Anten elemanlardan ilkinin merkezini orijinde kabul edelim. Diğer anten elemanları, z eksenin pozitif kısmında ve z ekseni üzerinde, Şekil 2.4'deki gibi yerleştirelim. Antenler arasındaki mesafe eşit ve d kadar olduğu durumda, k. antenin orjine uzaklığı kd ile hesaplanabilir.



Şekil 2.4 : Lineer anten modeli.

İşaret kaynağının yeterince uzakta olduğunu kabul ettiğimizde, alıcı tarafına ulaşan dalgaların düzlem dalga formunda olduğunu düşünebiliriz. Bu durumda antenler arasındaki gecikme yol farklarından faydalanılarak belirlenebilir. Bir referans düzlem dalga yüzü seçelim ve bu dalganın a_0 anten elemanına ulaştığı süreyi Δt_0 kabul edelim. O halde belirlenen düzlem dalganın *k.* anten elemanlanına ulaşma süresi şu şekilde ifade edilir.

$$\Delta t_k = \Delta t_0 - kdcos\Theta/C \qquad (\mathbf{2.8})$$

2.8 denklemindeki *C*, ışık hızıdır. Düzlem dalgaların yayılma hızı sabit ve ışık hızına eşit kabul edilmiştir. Θ, düzlem dalganın yayılma yönü ile anten elemanlarının dizildiği doğru arasında kalan açıdır.

Δt_0 referans süresi her anten elemanda görülecektir. Bu süreyi sıfır kabul edip, denklem 2.8'yi, 2.7'da yerine koyduğumuzda elde edilen demet katsayıları;

$$w_k = e^{-j2\pi f_c kdcos\,\Theta/C} \qquad (\mathbf{2.9})$$

şeklinde elde edilir.

2.8 Demet cevabı

Demet cevabı, her bir anten elemanı tarafından alınan işaretlerin demet katsayıları ile çarpılıp toplanması ile elde edilir. Alıcı yapısı, Şekil 2.3'de gösterilen ayrık demet biçimlendirmenin matematiksel ifadesi ise şu şekilde ifade edilir.

$$Y(t) = \frac{1}{N}\sum_{k=0}^{N-1} w_k\, \tilde{y}_k\,(t) \qquad (\mathbf{2.10})$$

2.10 denklemindeki $1/N$ çarpanı $Y(t)$ demet cevabını normalize etmek içindir.

2.9 Demet cevabının yönlendirilmiş anten karakteristiği

Sinusoidal işaretin fazındaki Θ, çok uzaktaki bir kaynağın yaydığı dalganın anten dizisine geldiği açıdır. Θ değiştirilerek elde edilen demet katsayıları ile anten dizisi, farklı yönlere yönlendirilebilir.

Denklem 2.3'deki $\tilde{y}_k\,(t)$ işaretini, denklem 2.10'da yerine koyalım. $\tilde{y}_k\,(t)$'nin fazında görünen gecikmelerin yerine ise lineer anten modeli için elde ettiğimiz 2.8

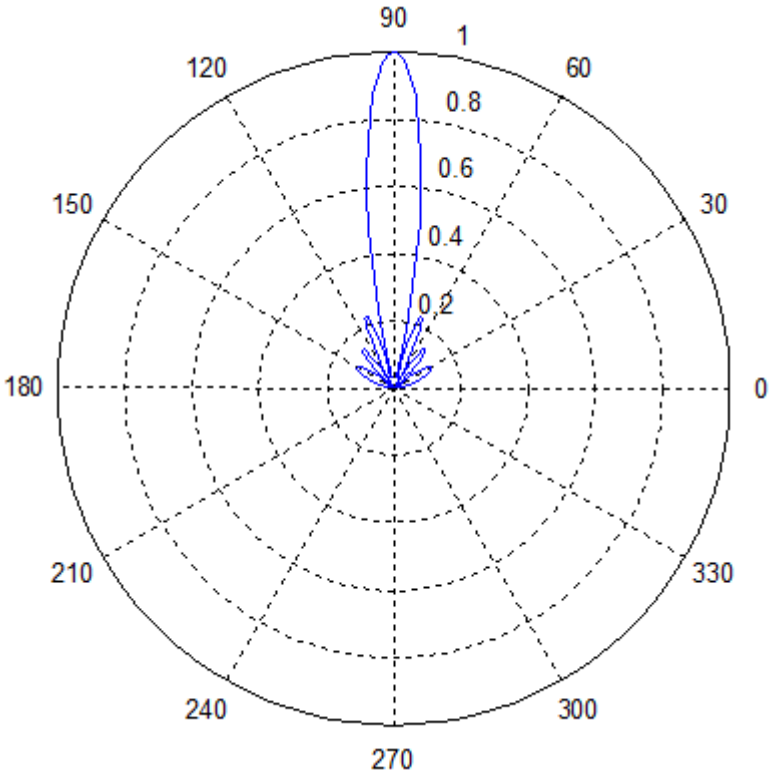
bağıntısını koyalım ve Δt_0 gecikme süresini 0 alalım. Anten elemanlarının demet katsayılarını, $w_k = 1$ olarak seçelim. Bu durumda demet cevabı;

$$Y(t) = \frac{1}{N}\sum_{k=0}^{N-1} x\,(t)\, e^{j\,2\pi f_c k d \cos\,\Theta/C} \qquad (\mathbf{2.11})$$

şeklinde elde edilir. 2.11 denklemi düzenlendiğinde ve $x\,(t) = 1$ sabit olarak alındığında ise

$$Y(t) = \frac{1}{N}\left[\frac{1 - e^{j\,2\pi f_c N d \cos\,\Theta/C}}{1 - e^{j\,2\pi f_c d \cos\,\Theta/C}}\right] \qquad (\mathbf{2.12})$$

eşitliği elde edilir. 2.12 denklemindeki $Y(t)$ ifadesi, $[0^o,\ 180^o)$ aralığındaki maksimum değerini, $\Theta = 90^o$'da almaktadır. $\Theta = 90^o$, işaretin anten dizilerine tam karşıdan geldiğini söylemektedir. Θ değiştirilerek diğer açılardaki demet cevaplarını da gözlemleyelim. $N = 8$ elemanlı lineer dizi anten için $\Theta \in [0^o,\ 180^o)$ arasında 1'er derecelik açılarla değiştirilerek hesaplanan ışıma örüntüsü, Şekil 2.5'da görülmektedir.



Şekil 2.5 : Işıma örüntüsü (analitik sonuç).

Şekil 2.5'deki ışıma örüntüsünü simülasyon ile elde etmek de mümkündür. MATLAB'ta $[0^o,\ 180^o)$ arasında 1'er derecelik çözünürlükle, anten dizisine geldiği

varsayılan sentetik sinüsoidal işaretler üretilmiş, yine demet katsayıları $w_k = 1$ olarak seçilmiştir. 2.10 denklemine göre hesaplanan demet cevaplanın enerjisi, sinusoidal işaretin 1 periyot süresi boyunca hesaplanmıştır. Bu durumda elde edilen ışıma örüntüsü, denklem 2.10'daki analitik hesaplama ile aynı sonucu vermiştir.

2.10 YTA için çoklu ışıma örüntüsü çıkarımı

Birden fazla ışıma örüntüsü, aslında ayrık demet biçimlendirmenin avantajlarından biridir. Ayrık demet biçimlendirme, yazımsal olarak fazla karmaşıklık getirmemesiyle beraber, donanımsal olarak da maliyetleri arttırmamaktadır.

Resiprok ve yönsüz anten elemanları, herhangi bir fiziksel müdahaleye uğratılmadan, çeşitli yönlere yönlendirilebilirler. Bu durumun maliyeti ise işaretin farklı demet katsayıları kombine edilmesidir.

Önce denklem 2.9 ile farklı yönler için demet katsayılarını bir matris oluşturacak şekilde elde edelim.

$$w_{k,i} = e^{-j\,2\pi f_c k d \cos\Theta_i / C} \qquad (\mathbf{2.13})$$

$$i = 0, 1, 2, \ldots\ldots$$

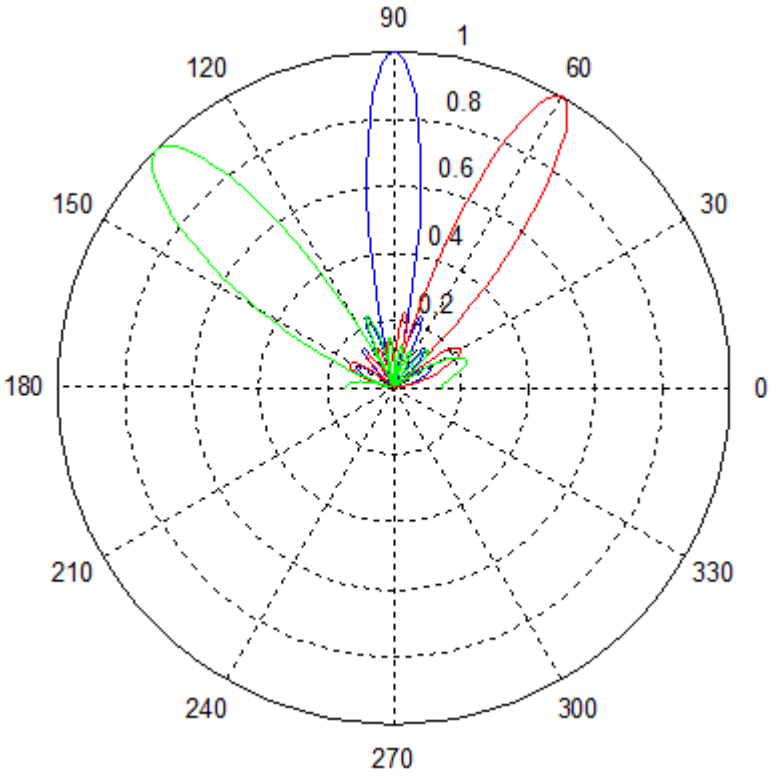
eşitliği elde edilir. Θ_i herhangi bir yönü temsil eden dalganın geliş açısıdır. i indisinin sayısı ise istenilen demet sayısı kadardır.

Denklem 2.10'dan çok yönlü demet cevabını türetelim. Bu durumda çok yönlü demet cevabı şu şekildedir.

$$Y_i(t) = \frac{1}{N}\sum_{k=0}^{N-1} w_{k,i}\, \tilde{y}_k\,(t)\,; \qquad i = 0,1, \ldots\ldots \qquad (\mathbf{2.14})$$

$Y_i(t)$, i. demet cevabıdır. $w_{k,i}$, i. demetin katsayı dizisidir. $\tilde{y}_k\,(t)$ ise 2.3 denkleminde de belirtilen k. anten elemanının aldığı işarettir.

Θ'nın 60^o, 90^o ve 135^o'deki ışıma örüntüleri, Şekil 2.6'de görülmektedir. Bu şekil, 8 elemanlı lineer dizilmiş bir anten dizisi ve denklem 2.9'de belirlenen katsayı kuralı ile yapılan simülasyon sonucu elde edilmiştir. Bu katsayı kuralı ile elde edilen ışıma örüntüsü, faz tarayıcı anten olarak da bilinir [12-21].



Şekil 2.6 : YTA ışıma örüntüsü (Simülasyon sonucu).

135^{o}'deki analobun genişliği daha tombik görünmektedir. Bu durum katsayı seçme kuralımızın sonucudur. Farklı katsayılar seçerek, farklı demetler oluşturulabilir. Bu şekil örnek olarak verilmiştir.

Anten elemanlarının sayısı arttıkça, ışıma örüntülerinin analobu daralır. Bu durumda daha keskin açı kestirimi yapmak mümkün olabilir. Denklem 2.14'deki açı çözünürlüğünün artması da daha fazla ışıma demet cevabı oluşturulabileceğini gösterir. Her iki durum da sayısal işlem yükünü arttırır. YTA için anten elemanı sayısı ve açı çözünürlüğü performansı etkileyen parametrelerdir.

FFT tabanlı demet biçimlendirme de birden çok ışıma demeti üreten bir yapıya sahiptir; ancak fazla esneklik sağlamamaktadır [13]. FFT tabanlı biçimlendirmede, FFT boyutu ışıma örüntülerin sayısını vermektedir ve demet cevapları birbirine dik olmak zorundadır. YTA ise istenildiği takdirde tüm anten elemanlarından aldığı verileri işleyebileceği gibi, bazı durumlarda daha az anten kullanarak demet biçimlendirmesini de ayarlayabilir. Ayrıca YTA'nin herhangi bir katsayı sınırlaması yoktur.

2.11 YTA için ayrık modelin oluşturulması

Daha önceki bölümlerde elde edilen analitik denklemler baz alınarak ayrık model türetilmiştir. O halde, ayrık model için gereken parametreleri çıkartalım. Öncelikle yapacağımız kabuller hatırlatılmaktadır.

Her bir anten elemanından gelen işaretlerin senkron örneklendiği kabul edilmiştir. Anten elemanlarının senkron örneklemesi bu tezin kapsamında değildir.

Nyquest teoremine göre örnekleme frekansı seçilecektir. Örnekleme frekansı, sayısal işlemler için performansı doğrudan etkileyen bir parametredir. Bu yüzden çok yüksek örnekleme frekansı seçilmemelidir. Aşırı örneklenmiş işaretin teorik olarak bir kaybı olmaz. Hatta işaretler pratikte hiçbir zaman bant sınırlı olmayacağından yüksek örnekleme frekansının daha az girişime neden olacağını söyleyebiliriz. Ancak işaret kaynağını bant sınırlı ve çok dar bantlı kabul etmiştik .

En genel halde alınan işaretin $t=nT$ anlarında örneklediğimizi düşünürsek, 2.14 denklemini şu şekilde yeniden düzenleyebiliriz.

$$Y_i(t = nT) = \frac{1}{N}\sum_{k=0}^{N-1} y_k(nT)\, w_{k,i} \qquad (\mathbf{2.15})$$

$$n = 0,1, \dots \dots$$

N anten sayısı ve örnekleme anlarını değişken olarak tuttuğumuzda ise elde ettiğimiz denklem şu şekilde yeniden ifade edilebilir.

$$Y_i(n, N) = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{x}\,(n)\, e^{-j\,2\pi f_c k d \sin\Theta_i / C}\; w_{k,i} \qquad (\mathbf{2.16})$$

$$T = \frac{1}{Fs}, \qquad n = 0, 1, 2, \dots.$$

2.16 denklemindeki f_c, taşıyıcı işaretin frekansını, N anten sayısını, d lineer model için antenler arasındaki mesafeyi, Θ_i ise her bir demet yönünü göstermektedir. Fs örnekleme frekansı n ise zamanda alınan örnekleri belirtmektedir. Bu parametreler, YTA sisteminin simülasyonlarında performans belirleyici olacaktır.

Denklem 2.16 ile belirlenen YTA ayrık biçimlendirme modelinin hesaplama karmaşıklığı 4. bölümde belirlenecek ve bu modeldeki parametrelerin kurgulanan senaryolara etkisi tartışılacaktır. Bu parametrelerin simülasyonlar üzerindeki etkisi

ise 5. bölümde değerlendirilecektir. Bir sonraki bölümde GPU'nun yazılımsal ve donanımsal özellikleri incelenecektir. GPU donanımı ve GPU programlama üzerinde yeterli tecrübe ve bilgiye sahip olman kişiler bu bölümü atlayabilir.

3. HESAPLAMALI BİLİMLERDE GPU

GPU, grafik işleme ünitesidir. İngilizcesi 'Graphics Processing Unit'in baş harfleri ile oluşan kısaltmadır. GPU'lar kişisel bilgisayarlarda, iş istasyonlarında, oyun konsollarında, grafik yaratmak için kullanılır. Grafik işleme konusunda son derece yüksek performans gösteren cihazlardır. GPU'ların yüksek paralel yapıları, çok sayıda işlemi kısa süre içerisinde yapmak için geliştirilmiştir ve grafik yaratımını hızlandırıcı çok sayıda matematiksel işlemcileri barındırır.

GPU son yıllarda, grafik yaratımı ile sınırlı kalmamış ve pek çok alanın ilgisini çekmeye başlamıştır. GPU'lar bazı senaryolarda CPU'ların alternatifi, bazı senaryolarda ise FPGA'lerin alternatifi olmuştur. Ayrıca GPU, CPU ile koordinasyonlu çalışabilmesinden dolayı bazı senaryolara, melez çözümler üretmiştir.

3.1 Güçlü ekran kartları

Ekran kartları, masaüstü bilgisayarlarda, taşınabilir bilgisayarlarda, akıllı telefonlarda ve gömülü sistemlerde bulunmaktadır. Ekran kartlarının üzerinde grafik işlemcisi, RAM ve diğer yardımcı bileşenler bulunabilir. Ekran kartları ile anakart arasındaki bağlantı ise son yıllarda 'PCI express' yuvaları ile sağlanmaktadır. PCI Express hattı, çok geniş bantlı verileri iletebilme yeteneğine sahiptir. GPU'lar, bu hatlardan arzu ettiği bant genişliğindeki verileri almaktadır. Şekil 3.1'de güçlü bir ekran kartı ile 'PCI express 3' yuvası görülmektedir.

Şekil 3.1 : GPU (Solda), Anakart (Sağda)

PCI express 3.0 için veri iletme hızından bahsedebiliriz. Nathan Edwards bir makalede, PCI express 3.0 hattının teorik ve pratikteki hızını değerlendirmiştir. Bu kaynağa, PCI express 3.0'ın her bir hattının pratikteki hızı 8 GT/s (Giga Transfer per second)'dir. PCI Express 3'ün teorik hızı biraz daha fazla olmasına rağmen, pratikte elde edilen hız daha düşüktür. Çünkü, bu iletişimde hata düzeltme kodlaması vardır. Her bir transfer, 1 biti temsil etmektedir. 8 bit'in 1 byte olduğu hatırlatılırsa bir hat üzerinden 985 MB/s hızı ile iletim yapılabilir. x16, 16 kanalı ifade etmektedir. Bu durumda toplam transfer hızı 985x16 MB/s olacaktır [19]. PCI express 3'ün veri iletim hızı, YTA algoritmasında etkili olacağından, bu değerlere göz atılmıştır. Veri bant genişliğinin CPU ve GPU arasındaki iletim hızının fiziksel sınırlarına dayanabileceği unutulmamalıdır. Sonuç olarak, PCI express gerçekten yüksek veri iletim hızına sahiptir ve güçlü bir ekran kartı ile CPU arasında yüksek bant genişliğinde veri iletimine olanak sağlamaktadır.

Ekran kartları, kolayca değiştirilebilir ve bir üst sürüme yükseltilebilir. Bu sayede, GPU'ların gelişmesi ile birlikte tasarlanan ölçeklenebilir algoritmaların performansları, yazılımsal müdahaleler yapılmadan iyileştirilebilir.

3.2 Hesaplamalı bilimlerde GPU

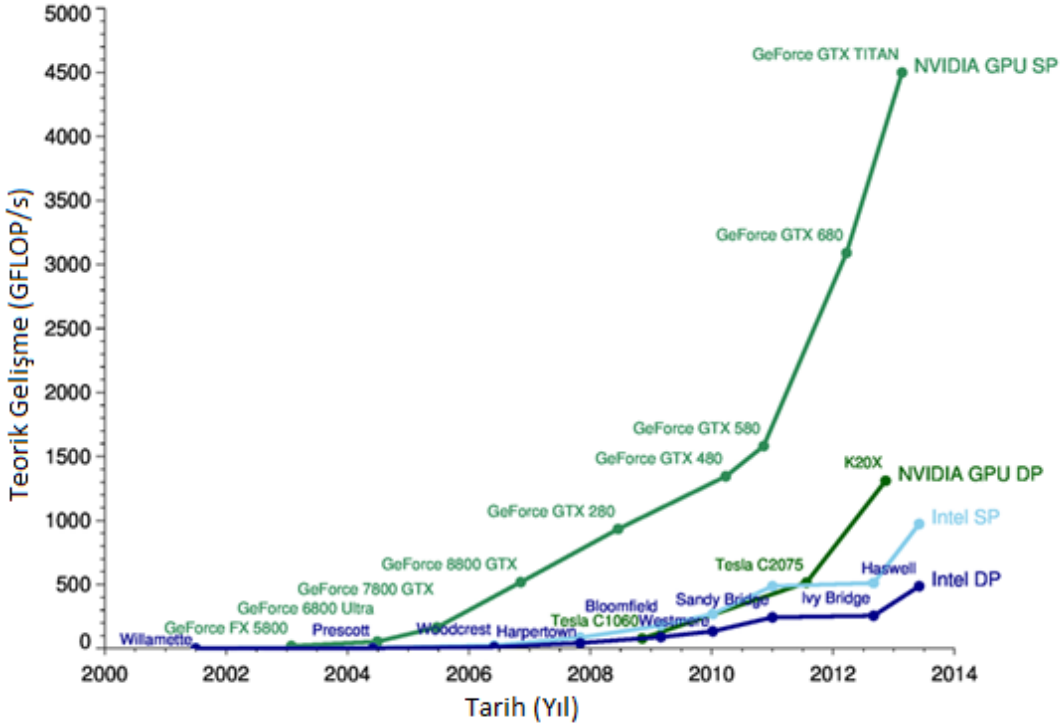
GPU'lar her ne kadar grafik işlemek için tasarlansa da paralel işlem yapabilme kapasitesi farklı alanların da ilgisini çekmiştir. Finansal hesaplama, savunma ve istihbarat, makine öğrenmesi, üretim, medya ve eğlence, yeraltı kaynaklarının araştırılması, güvenlik ve gizlilik, bu alanlara örnek olarak verilebilir. Hesaplamalı bilimlerde tanımlanan bazı problemler için Nvidia'nın önerdiği çözümler mevcuttur [22].

GPU, CPU ve FPGA gibi bileşenlerin alternatifi olmaya başlamıştır. Literatürde GPU ile CPU'nun performanslarının karşılaştırılması da aslında onun, bazı uygulamalar için, CPU'nun alternatifi olduğunu göstermektedir. Fakat GPU'yu etkin kılan özellik, CPU'nun alternatifi olması değildir. Keza GPU'nun CPU ile uyum içinde çalışması onun daha belirgin özelliğidir. Bir başka deyişle GPU programlama aslında melez bir programlamadır. Temel amacı CPU'nun yerini almak değildir. GPU'nun katkısı, CPU'nun işlem yükünü hafifletmek, dahası CPU'nun yetiştiremeyeceği bazı işlemleri, yetiştirmesidir.

Bu tezdeki çalışmamdan önce GPU'nun işlem gücünden faydalandığım başka tasarımlar da mevcuttur. FFT tabanlı kanallaştırıcı, DDC tabanlı kanallaştırıcı, FFT ve DDC melez kanallaştırıcısı ve bazı ayrık demodulatörler bunlara örnektir.

3.3 Performans ve gelecek

GPU işlem yapabilme kapasitesi, her geçen yıl üstel olarak artmaktadır. Şekil 3.2'de GPU'nun CPU'ya karşı gelişme hızını gösteren bir grafik mevcuttur. GPU'ların paralel işlem yapmaya elverişli olması, onların gelişimini daha hızlı yapmaktadır. CPU'lar ise seri işlemler için tasarlanmış cihazlardır. Seri işlemler için önemli etken olan saat frekansı, neredeyse fiziksel sınırlara dayanmıştır. Bu yüzden CPU gelişiminin daha yavaş olması beklenebilir. Tabi ki; CPU'lar için de paralel yaklaşımlar mevcuttur. Fakat, burada yapılan mukayesede CPU ve GPU'nun birincil görevleri baz alınmıştır.



Şekil 3.2 : GPU ve CPU Gelişme süreci

Şekil 3.2'deki grafikte Nvidia'nın GPU cihazları ile Intel'in CPU cihazları karşılaştırılmıştır [20]. Bu cihazların isim etiketlerinin yanında görülen SP, 'Single precision floating point'i, DP ise 'double precision floating point'i ifade etmektedir. Grafiğe göre, NVIDIA GPU SP'nin işlem yapabilme gücü, diğer NVIDIA GPU DP,

Intel SP ve Intel DP'ye göre oldukça hızlı gelişmiştir. Intel'in CPU'larının SP ve DP'si için fark fazla değildir. Fakat NVIDIA'nın GPU'ları, SP ve DP için oldukça farklı görülmektedir.

2004'e kadar CPU ve GPU'lar arasında fazla fark yok iken, GPU'lar son yıllarda gelişimini oldukça hızlandırmıştır. CPU'ların 10 yıllık gelişme oranı yaklaşık 60 kat iken GPU'ların 10 yıllık gelişme oranı ise yaklaşık 1000 kattır. Bu grafiğe göre, Geforce GTX Titan kartı 2014 yılında yaklaşık 4.5 TFlop/s işlem yapabilme kabiliyetine sahip olduğunu söyleyebiliriz. GPU'ların daha hızlı gelişiminin başlıca nedeni, transistor sayısı ile alakalıdır. GPU için daha fazla transistor, daha fazla çekirdek, daha fazla işlem gücü demektir.

Ayrıca Şekil 3.2'deki, GPU'ların gelişme hızının artışı, GPU'ların geleceği hakkında ümit vermektedir. Nvidia bünyesinde çalışan William Dally'e göre, GPU'ların işlem yapabilme gücü 20 TFlop/s'lere kadar ulaşacaktır [16]. Bir başka olumlu beklenti ise, transistor boyutlarının gelecekte daha da küçük olmasıdır. Bu durumda, GPU'lar daha fazla işlemciye sahip olabilecektir.

3.4 Programlanabilirlik

Günümüzde GPU programlama, yaygınca bilinen programlama dilleri ile kolayca yapılabilmektedir. Bu programlama dilleri, her kesimden programcıya hitap etmektedir. C / C++ / C # , MATLAB, FORTRAN, OCTAVE programlama dilleri bunlardan örnek olarak verilebilir.

GPU mimarileri, ticari kaygılardan ötürü gizli tutulmaktadır; fakat bu mimariler üzerindeki araç kiti ile, mimariden bağımsız programlama yapılabilmektedir. CUDA geliştirme kiti, Nvidia marka GPU için yaratılmıştır. OpenCL araç kiti ise hem Nvidia üzerinde hem de diğer ekran kartları üzerinde çalışmaktadır. Bu tezin içeriğinde yapılan simülasyonlar, Nvidia marka GPU ve CUDA geliştirme ortamı ile gerçeklenmiştir.

3.5 Sonuçlarda duyarlılık

GPU programlama, DP sayılar ile işlem yapabilmektedir. GPU programlama ilk çıktığı yıllarda yalnızca FP sayılar ile yapılmaktaydı. Bu durum bir dezavantaj olarak görülebilir; fakat FP sayılar bile çoğu durumda yeterli hassasiyete sahiptir.

GPU'lar duyarlılık konusunda, CPU'lara üstünlük sağlamaz; ancak GPU'ların bir başka alternatifi olan FPGA programlamaya üstünlük sağlarlar. Son nesil FPGA kartları haricindeki FPGA kartları "fixed point" sayılar ile programlanmaktadır. Dolayısıyla programcı her adımında, dikkatli olmalıdır ve tasarımı için gerekli hassasiyeti kendi çabasıyla sağlamalıdır.

3.6 Enerji tüketimi

GPU'lar, çok az enerji tüketen birimler değildir. Masaüstü bilgisayarlarda, güçlü bir ekran kartını beslemek için çoğu kez 2 tane paralel hat kullanılır. GPU'ların enerji tüketimini ancak göreceli olarak inceleyebiliriz.

CPU'lar Şekil 3.2'de görülen grafiğe göre yavaş adımlarla gelişmektedir. CPU'ların enerji tüketimleri ise bir o kadar hızlı artmaktadır. GPU'lar için ise bu durumu daha olumludur. İşlem yapabilme gücüne göre tükettiği enerji, CPU'ya göre azdır. GPU'lar ancak CPU ile karşılaştırıldığında enerji verimliliğinden bahsedebiliriz. GPU'lar, FPGA kartlarına göre ise çok daha fazla enerji tüketir.

3.7 Heterojen programlama

Heterojen programlamada CPU ev sahibi (Host); GPU ise çekirdek (Kernel) konumundadır. Kernel işleyeceği verileri, Host'tan alır ve işlediği verileri Host'a geri gönderir. GPU programlama, PC'nin iki ana işlemcisi olan CPU ve GPU'dan ayrı ayrı yararlanmayı baz alır. CPU'lar seri işlemler için GPU'dan daha verimli cihazlardır. Bir işlemin girişi, bir başka işlemin çıkışına bağlı ise bu işlemler seri işlemlerdir. GPU, seri işlemlerde yüksek performans gösteremez. Seri işlemlerde, çalışma frekansı daha yüksek ve daha güçlü çekirdeklere sahip olan CPU, GPU'ya göre daha yüksek bir performans gösterir. İşte bu yüzden GPU da CPU'nun tam olarak alternatifi olmadığını söylemiştik. Oysa GPU, CPU ile koordinasyonlu bir şekilde çalışabilmektedir. Paralel ve seri işlemleri doğru işlemcilere paylaştıran bir yaklaşım, tek başına GPU ve ya CPU'yu programlayan yaklaşımlardan daha yüksek performans gösterir ve daha verimli güç tüketimi sağlar [17].

3.8 GPU programlama dilleri

GPU programlama dilleri, CUDA ve OpenCL olarak bilinmektedir. Daha doğrusu bu platformlar, grafik kartları için geliştirme ortamlarıdır.

CUDA (Compute Unified Device Architecture), 2006 yılında, NVIDIA tarafından GPU'lar ile genel amaçlı hesaplama yapmak için tasarlanmış mimaridir. CUDA sadece Nvidia ekran kartları üzerinde çalışmaktadır. Fakat dünya üzerinde 300 milyondan fazla CUDA destekli GPU olduğu tahmin edilmektedir.

OpenCL (Open Computing Language) ise Apple şirketi tarafından 2008 yılında kar amacı gütmeyen teknoloji şirketleri birliği Khronos Group'a önerilen platformdur. OpenCL, kabul gördükten sonra pek çok şirketin katkılarıyla hazırlanan heterojen hesaplama platformudur; AMD, Intel, NVIDIA ve ARM tarafından desteklenmektedir.

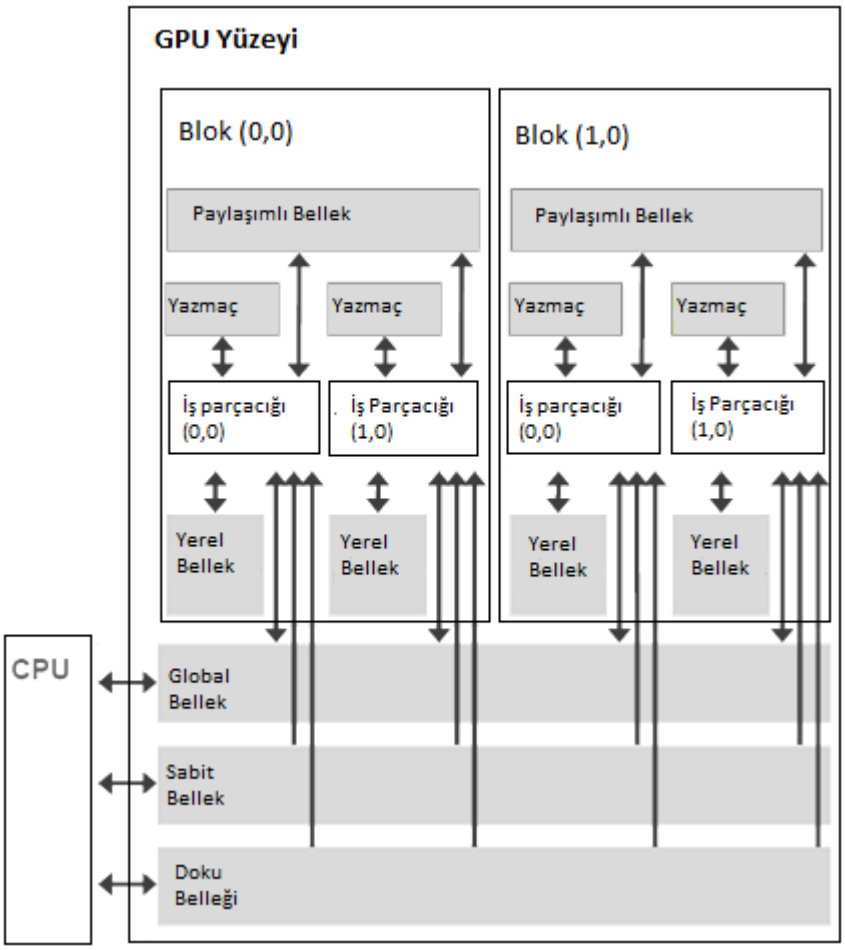
CUDA genel anlamda C ve C++ dilleri üzerinde kurulmuştur. CUDA, açık kaynaklı geliştirme ortamı değildir. CUDA'nın gelişimi, Nvidia tarafından yapılmaktadır. OpenCL ise açık kaynaklı bir yazılımdır. Birçok platform üzerinde çalışacak fonksiyon arayüzüne sahiptir.

Bu tezde önerilen YTA, CUDA C geliştirme ortamı ile gerçeklenmiştir. CUDA C programlama dili için Nvidia tarafından hazırlanan dokümanda başlangıç seviyesinden yüksek seviyelere kadar örnekler mevcuttur [23].

3.9 CUDA bellek mimarisi

Ekran kartlarında veri işlemek için öncelikli olarak CPU tarafındaki verinin GPU tarafındaki belleklere kopyalanması gerekir. GPU'da koşan bir programdan alınan verimi arttırmak için ekran kartının bellek mimarisini tanınması ve kodun tasarım aşamasında en uygun belleklerin seçilmesi gerekir. Sınırlı kaynaklardan seçilen bellekler farklı gecikmelere neden olmaktadır [15]. CUDA bellek mimarisi, Nvidia marka ekran kartlarını programlamada çok önemli bir yere sahiptir.

Şekil 3.3'de CUDA bellek mimarisi görülmektedir. Bu mimaride bulunan bellek türleri tanıtılacaktır.



Şekil 3.3 : GPU Bellek Mimarisi

CUDA mimarisi, bir yazılımsal taban üzerinde bulunan bloklardan ve her bloklardaki iş parçacıklarından oluşmaktadır. Her bloğa ve iş parçacığına ayrılmış bellekler vardır. Ayrıca tüm iş parçacıklarının ve blokların da ortak bellekleri vardır. Bu belleklerin kaynağı sınırlıdır.

İş parçacıklarının, herhangi bir belleğe erişme hızı farklıdır. Her zaman en hızlı erişilebilen belleği kullanmak mümkün olmayabilir. Çünkü, o bellek tamamen kullanılmış olabilir ya da başka bir iş parçacığının veya bloğun o belleğe erişmesi gerekebilir. Örneğin bir dizinin elemanlarının önce farklı katsayılar ile ölçeklenip sonra birikimli olarak toplandığını düşünelim. Bu durumda çarpma işlemi ayrı iş parçacıklarında yapılabilir. Çünkü her bir eleman, ayrı bir katsayı ile çarpılacaktır. O halde, sadece iş parçacıklarına özgü olan hızlı bir bellek kullanılabilir. Ancak, daha sonraki toplama işlemi, birikimli olduğundan, başka bir iş parçacığının bu işlemi yapabilmesi için tüm dizi elemanlarına erişmesi gerekir. O halde tüm dizi elemanlarının toplama işlemini yapacak iş parçacığının belleğine kopyalanması

gerekir ya da en başta çarpma işleminin yapıldığı iş parçacığı elemanları, çarpım sonuçlarını ortak bir belleğe yazması gerekirdi. Bu ve bunun gibi nedenlerden dolayı, bellek seçimleri programın performansını etkileyebilir.

Şekil 3.3'de görülen bellek türlerinin genel olarak hız sıralaması şu şekildedir.

Yazmaç > Paylaşımlı Bellek > Global Bellek = Yerel Bellek

Global belleklerin kendi içerisindeki hız sıralaması ise şu şekildedir.

Sabit Bellek > Doku Belleği > Global Bellek

Bu bellek türlerini inceleyelim.

3.9.1 Global bellek

Adından da anlaşılacağı üzere global bir bellektir. Tüm bloklardaki iş parçacıkları, bu belleğe erişebilir. Bellekten bir şey okuyabilir, belleğe bir şey yazabilir. CPU ile GPU arasındaki ilk bağlantının kurulduğu bellek birimlerinden biridir. PCI Express hattı üzerinden GPU'ya gönderilen veri ilk olarak global belleğe kopyalanır. GPU tarafındaki veri de bu bellekten CPU tarafına kopyalanabilir. Global bellek, GPU'nun en büyük boyuttaki belleğidir. CPU tarafından global belleğe veri kopyalama hızı ile global bellekten CPU'ya veri kopyalama hızı birbirinden çok az farklılık gösterebilir. Global belleğin kendi içerisinde, bir yerden başka yere veri kopyalama hızı ise çok daha hızlıdır. Global belleğin teorik bant genişliği, Nvidia GTX 680 marka ekran kartı için 192.2 GB/s'dir [24].

Global bellek büyük boyutta belleğe sahip olmasına rağmen; yine de tasarlanan programın bu bellek boyutunu aşabileceği unutulmamalıdır. Bir başka dikkat edilmesi gereken konu ise belli bir sürede, veri işleme zorunluluğu olan programlardır. Böyle bir program için GPU ve CPU arasındaki iletişim hızı yeterli olmalıdır.

3.9.2 Doku belleği

Sadece okunma özelliği olan global bellek türüdür. İş parçacıkları için Doku belleğinden veri okumak, Global bellekten veri okumaktan daha hızlıdır. Doku belleği, 2D ve 3D olarak tanımlanabilmektedir. Bu özelliği programcıya, indekslemelerde kolaylık sağlamaktadır.

3.9.3 Sabit bellek

Global belleğin sadece okuma özelliği olan türüdür. Programın ilk çalışma anında yükleyeceği ve sonradan değiştirmeyeceği değişkenler, bu bellekte saklanabilir. Sabit bellek, global bellekten daha hızlı performans göstermektedir.

3.9.4 Paylaşımlı bellek

Paylaşımlı bellek, bloklar için önemli bir bellek türüdür. Bu belleğe, sadece aynı bloğun iş parçacıkları erişebilir. İş parçacığının paylaşımlı belleğe erişimi, global belleğe erişimine göre yaklaşık 100 kat hızlıdır [25]. Bir blok, art arda yapması gereken işlemleri paylaşımlı bellek ile saklayabilir. Çünkü bir bloğun işlem sonuçlarını başka bir işlem için saklayabileceği en iyi bellektir. Bellek boyutu ise global belleğe göre çok azdır. Fakat çoğu kez, iyi tasarlanmış bir kod için yeterli büyüklükte olmaktadır.

3.9.5 Yazmaç

Her bir iş parçacığının özel belleğidir. Her iş parçacığı, sadece kendi yazmacını okuyabilir ya da kendi yazmacına veri yazabilir. İş parçacıklarının en hızlı eriştiği bellek türüdür. Yazmaçlar, dar kaynaklı belleklerdir. Bu yüzden, yazmaç değişkenlerinin fazla olmamasına özen gösterilmelidir.

3.9.6 Yerel bellek

Yerel bellek, global bellekten iş parçacığına özel alan tanımlanmasıyla oluşan bellektir türüdür. Yani yerel belleğin, global bellekten farkı yalnızca, bir iş parçacığına özgü olmasıdır. Bu belleğin hızı ise global bellek ile benzerlik göstermektedir.

4. YTA ALGORİTMASININ TASARLANMASI

Bu bölümde YTA'nın ışıma demet cevaplarını elde etmek için CPU ve GPU algoritmaları tasarlanacak ve algoritmaların üzerinde koşacağı donanımlar tanıtılacaktır.

4.1 Hesaplama karmaşıklığı

Hesaplanacak işlem sayısı, bölüm 2.11'da elde edilen ayrık model baz alınarak elde edilecektir. Denklem 2.16'yı tekrar hatırlayalım.

$$Y_i(n,N) = \sum_{k=0}^{N-1} \tilde{x}\,(n)\; e^{-j\,2\pi f_c k d \sin\Theta_i / C}\; w_{k,i} \qquad (\mathbf{4.1})$$

Bu denklemin parametreleri, tasarladığımız yazılım için aynı zamanda performans parametresi olacaktır. Yapılacak bazı kabuller ile gereken hesaplama karmaşıklığını hesaplayalım. 1 saniyelik veri üzerinde çalıştığımızı varsayalım. Toplam işlem sayısı, örnekleme frekansı *Fs* olan kompleks örnek sayısı üzerinden hesaplanacaktır. *N* tane anten elemanı olduğu durumda, ADC'lerden alınan veri sayısı *NFs* kompleks örnek olduğu görülmektedir.

Öncelikle, tek bir demet (*i=0*) için gereken işlem miktarını hesaplayalım. Demet cevabı için *NFs* tane kompleks çarpma yapılacaktır. *(N-1)Fs* tane kompleks toplama, toplam sembolünden gelecektir. Her bir kompleks çarpma 4 çarpma ve 2 toplamadan oluşmaktadır. O halde toplam işlem sayısı şu şekildedir.

$$2(N-1)Fs + 2NFs = (4N-2)Fs\ \ Toplama \qquad (\mathbf{4.2a})$$

$$4NFs\ Çarpma \qquad (\mathbf{4.2b})$$

Elde edilmek istenen demet sayısı *D* olmak üzere, 4.2a ve 4.2b denklemeleri *D* ile çarpılarak toplam işlem sayısı gösterilebilir.

$$O(.) = DFs\,\{\,(4N-2)\,Toplama + 4N\,Çarpma\,\} \qquad (\mathbf{4.3})$$

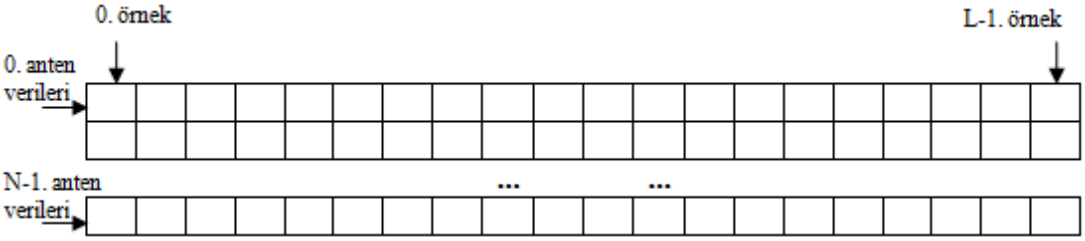
4.2 Veri kopyalama hızı

CPU'dan GPU'ya kopyalanacak örnek veri miktarı NFs, GPU'dan CPU'ya geri dönecek olan örnek veri miktarı ise DFs kompleks örnektir. Her iki yönde veri kopyalama hızlarını yaklaşık olarak eşit kabul edelim. O halde veri kopyalama süreleri N ve D ile orantılıdır.

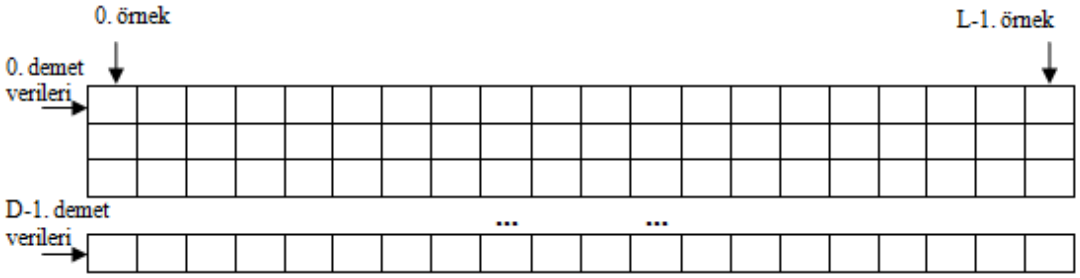
Bazı senaryolarda, verilerin Host'tan Kernel'e (t_u) ve Kernel'den Host'a (t_d) kopyalanma süresi, Kernel'de veri işleme (t_e) süresine göre dominant olabilir. Eğer bu üç adım seri ise algoritmanın çalışma süresi bu üç adımın harcadığı sürenin toplamıdır ($t_t=t_u+t_e+t_d$). Kopyalamaların hesaplama süresi üzerindeki etkisinin azaltılması veya yok edilmesi için GPU 'stream' konusu incelenmelidir [23].

4.3 Çerçeve yapısı

YTA algoritmamız, 1 çerçevelik veriler ile hesaplanmaktadır. Şekil 4.2'de giriş çerçevesi, Şekil 4.3'de ise çıkış çerçevesi gösterilmektedir. Çerçevelerin satır ve sütunları değişken boyuttadır. Giriş çerçevesi için satır sayısının boyutu N'ye (anten elemanı sayısı) eşittir. L ise zamanda alınan örnek sayısıdır ve her iki çerçeve için de sütun sayısına eşittir.



Şekil 4.1 : Giriş çerçevesi.



Şekil 4.2 : Çıkış çerçevesi.

Şekil 4.1'de çerçevenin her bir satırı, farklı anten elemanlarından alınan verilerden oluşmaktadır. Çerçevenin her bir sütunu ise anten elemanlardan alınan zamanda örneklenmiş işaretten oluşmaktadır. Örneklemenin senkron yapıldığı ve her anten elemanının örnekleme frekansının eşit olduğu varsayımı altında bu çerçeve oluşturulmuştur.

Şekil 4.2'da görülen çerçeve ise çıkışta elde edilecek olan çerçevedir. Bu çerçevenin satırları, ışıma demet cevaplarından oluşmaktadır. Çerçevenin sütunları ise çıkış dizisinin zamanda örnekleridir. Dolayısıyla, çıkış çerçevesinin satır sayısı, giriş çerçevesinin satır sayısından farklı olabilir; ancak çıkış çerçevesinin sütun sayısı, giriş çerçevesinin sütun sayısına eşit olmak zorundadır.

4.4 CPU ile YTA algoritması

Işıma demet cevabını, CPU üzerinde gerçekleyelim. Algoritma, bir çerçeve süresi boyunca örneklenmiş veriler ile tasarlanmıştır. Giriş çerçevesi, Şekil 4.2'te gösterilmektedir. Çerçeve uzunluğu, keyfi bir değer olabilir. Fakat bu uzunluk algoritmanın performansını etkileyecektir. ADC verileri, algoritmanın ana girdisidir, Demet katsayıları, anten elemanı sayısı, çerçeve uzunluğu ve biçimlendirilecek demet sayısı ise diğer girdilerdir. Bu girdilerin bazıları zamanla değişebilir ya da sabit olabilir. Algoritmanın çıkışı ise çok yönlü demet cevabıdır. Tasarlanan CPU kodunun algoritması şu şekildedir.

Çizelge 4.1 : CPU, YTA algoritması.

CPU Multi-beamformer for CPU

Girdiler: *ADC verisi (inData), demet katsayı dizileri (beamCoeff), anten eleman sayısı (N), çerçeve uzunluğu (L), demet boyutu (D)*
Çıkış: *YTA çıkış verisi (outData)*
for (integer n=0, n< L; n++){
  for (integer k=0; k<D; k++){
  for (integer j=0; j<N; j++){
  outData[k][i] += inData[j][i]*beamCoeff[k][j];
  }
  }
}

Çizelge 4.1'deki algoritmada iç içe yazılmış 3 'for' döngüsü görülmektedir. Bu döngünün birincisi giriş çerçevesinin sütunlarını gezmektedir. 2. 'for' döngüsü elde edilmek istenen demet sayısı kadar koşmaktadır. Bu döngü aynı zamanda çıkış çerçevesinin satırlarını oluşturmaktadır. Çıkış çerçevesinin satırlarının farklılığını

demet katsayıları belirlemektedir. 3. 'for' döngüsü ise demet cevabı için koşmaktadır. Bu döngü denklem 4.1'de verilen matematiksel ifadeyi gerçeklemektedir.

4.5 GPU ile YTA algoritması

GPU'da gerçeklenen çekirdek kodunun algoritması bir indis dizisi ve iç içe 2 'for' döngüsünden oluşmaktadır. İndisler bir vektör dizisi olarak düşünülebilir. İndis dizisinin her bir elemanı, iş parçacığının yazmacında tutulmaktadır. GPU algoritması, bu indis dizisi ile, CPU algoritmasından bir 'for' döngüsü daha az koşmaktadır. GPU algoritması şu şekildedir.

Çizelge 4.2 : GPU, YTA algoritması.

GPU Multi-beamformer for CPU

Girdiler: *ADC verisi (inData), demet katsayı dizileri (beamCoeff), anten eleman sayısı (N), çerçeve uzunluğu (L), demet boyutu (D)*
GPU içeriği: *blok indisi (blockIdx) * blok boyutu (blockDim) + iş parçacığı indisi (threadIdx)*
Çıkış: *YTA çıkış verisi (outData)*

```
Integer indisler  = (blockIdx * blockDim) + threadIdx;
for (integer z=0; z< D; z++){
        for (integer k=0; k < N; k++){
                outData[z*L + indisler] +=
                        inData[k*L + indisler] * demet katsayıları[z*N + k];
        }
}
```

GPU algoritmasında, örneklenmiş her işaretin hesapları ayrı bir iş parçacığı ile yapılmaktadır. Bu sayede aynı anda birden çok işlem yapılabilmektedir. Ancak, bu iş parçacıklarının sayısı sonsuz değildir; CUDA hesaplama aracı için hesaplama sınırlarını incelemek gerekir [22].

Giriş çerçevesinin sütun sayısı, blok sayısı ve iş parçacığı sayısının çarpımına eşittir. Her iş parçacığının gerçekte çalıştığı fiziksel bir çekirdek vardır. Ancak yazılımda, iş parçacığı sayısının çekirdek sayısından fazla olması durumu da mevcuttur. Yani fazla iş parçacığı yaratmak, performansı fazlaca artırmak anlamına gelmemektedir. Bu durumda yazılımda yaratılan iş parçacıkları, GPU'nun çekirdeklerinde sıraya girebilir. GPU'nun çalışma mimarisi açık kaynaklı olmadığından GPU'daki serileşme hakkında kesin bir yorum yapamayız. Ancak GPU'nun kendi içerisindeki sabit yazılımının serileşmeyi optimize ettiğini ve yaratılan her iş parçacığının paralel çalıştığını kabul ediyoruz. Daha sonra iş parçacığı sayısı değiştirilerek optimize bir

kod tasarlanabilir. Her kodun farklı ekran kartları ile farklı performans sergileyebileceği de göz önünde bulundurmak gerekir. Hatta yaratılan iş parçacığı sayısının çokluğu ile elde edilen performans arasında lineer bir ilişki olmadığı da bilinmelidir.

Sonuçta ekran kartlarının güçleri birbirinden farklı olacağından aynı kod farklı ekran kartlarında, farklı performanslarla çalışacaktır. Ekran kartının çekirdek sayısı ve işlemcisinin saat frekansı, performansı etkileyen başlıca faktörlerdir.

4.6 CPU ve GPU algoritmalarının karşılaştırılması

CPU ve GPU algoritmalarını değerlendirmeden önce GPU'nun, CPU'nun alternatifi olmadığını hatırlatmak gerekir. Bölüm 3.7'de heterojen programlamadan bahsedilmişti. GPU, CPU ile uyumlu çalışabilen yardımcı grafik işleme ünitesidir. GPU bazı vektörel işlemleri, CPU'dan daha hızlı yapabilmektedir. Fakat GPU seri işlemlerde, CPU'dan daha yavaş performans gösterebilir. 4.1 ve 4.2 algoritmalarının farkı, GPU algoritmasının bir 'for' döngüsü daha az koşmasıdır. Genel olarak 'for' döngülerinde CPU'nun performansı daha yüksektir; ancak giriş ve çıkış çerçevelerinin sütün sayısının satır sayısından çok daha fazla olduğu göz önünde bulundurulduğunda, GPU algoritmasının döngü sayısı, CPU algoritmasının döngü sayısından çok daha azdır.

Birikimli toplama işlemleri, integral almak gibi işlemler, paralel işlemcilere uygun değildir. GPU ve CPU algoritmalarının en iç kısmındaki 'for' döngüsünde de birikimli toplama işlemi yapmaktadır. Yani bu işlemi paralelleştirmek mantıklı görünmemektedir. CPU algoritmasının 2. 'for' döngüsü ve GPU algoritmasının 1. 'for' döngüsü ise, elde edilmek istenen demet sayısı kadardır. Her demet, farklı demet katsayılarıyla çarpılmaktadır. Farklı demet katsayıları ile çarpılan örnekler tekrar birikimli toplanacaktır. Dolayısıyla bu 'for' döngüsünü de paralelleştirmek mantıklı gözükmemektedir.

4.7 GPU ve CPU donanımının karşılaştırılması

Şekil 4.3'teki ekran çıktısı, Cuda SDK'nın örnek projelerinden 'Cuda Device Qery'den alınmıştır. Bu şekilde, GPU cihazının hem donanımsal özellikleri, hem de yazılımsal özellikleri görülmektedir. Gözlemlenen GPU'nun gerçekte 1536 tane

CUDA çekirdeği bulunmaktadır. Buna karşılık her blok için tanımlanabilen maksimum iş parçacığı sayısı *1024* tanedir. Ayrıca, *1024* tane blok tanımlanabilmektedir. Yani bu GPU, yazılımsal olarak toplamda *1024x1024* tane iş parçacığı yaratabilmektedir. Maksimum iş parçacığı sayısının GPU'nun gerçekteki çekirdek sayısından çok daha fazla olduğu görülmektedir. İş parçacığı sayısını maksimum seçmek her zaman performans kazancı sağlamaz. Bazı durumlarda performans kaybı bile yaşanabilir.

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Abdullah>"C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.0\bin\win32
\Release\deviceQuery.exe"
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.0\bin\win32\Release\deviceQuer
y.exe Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 2 CUDA Capable device(s)

Device 0: "GeForce GTX 680"
  CUDA Driver Version / Runtime Version          6.0 / 5.0
  CUDA Capability Major/Minor version number:    3.0
  Total amount of global memory:                 4096 MBytes (4294967295 bytes)
  ( 8) Multiprocessors x (192) CUDA Cores/MP:    1536 CUDA Cores
  GPU Clock rate:                                1059 MHz (1.06 GHz)
  Memory Clock rate:                             3004 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 524288 bytes
  Max Texture Dimension Size (x,y,z)             1D=(65536), 2D=(65536,65536), 3
D=(4096,4096,4096)
  Max Layered Texture Size (dim) x layers        1D=(16384) x 2048, 2D=(16384,16
384) x 2048
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     2147483647 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  CUDA Device Driver Mode (TCC or WDDM):         WDDM (Windows Display Driver Mo
del)
  Device supports Unified Addressing (UVA):      No
  Device PCI Bus ID / PCI location ID:           4 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simu
ltaneously) >
```

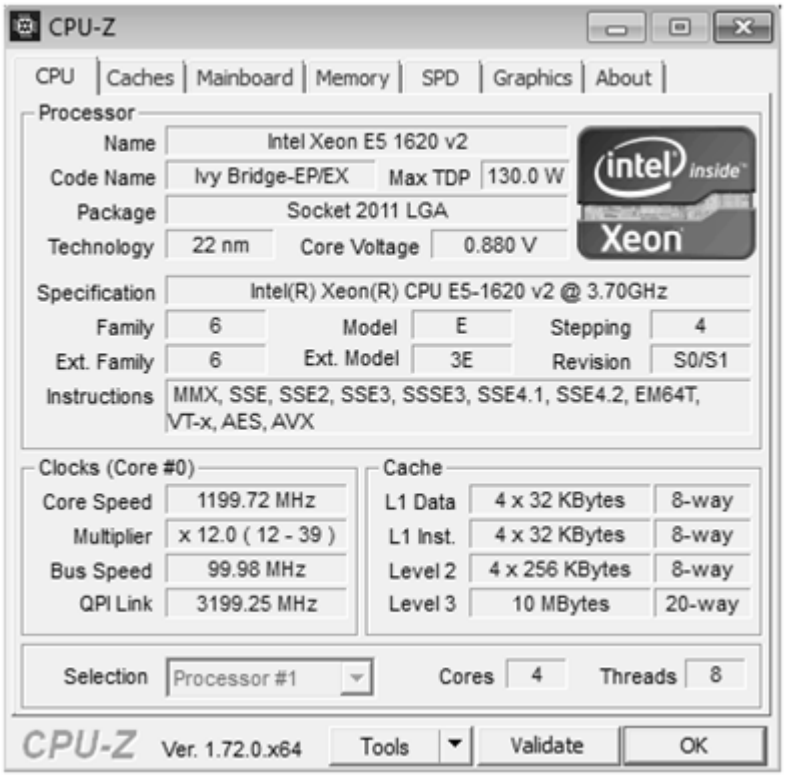
Şekil 4.3 : Simülasyonlarda kullanılan GPU'nun donanımsal ve yazılımsal kabiliyetleri.

Ekran kartlarının hesaplama kabiliyetleri, kuşaklarına göre, farklılık gösterebilir. Nvidia ekran kartları için 'Cuda Capability' versiyonu önemli bir kriterdir. Genelde yüksek versiyonlu ekran kartları, kendisine göre düşük versiyonlu ekran kartlarına yazılımsal üstünlük sağlamaktadır. Ayrıca versiyon farklılıkları, yazılımsal farklılıklara da neden olmaktadır. Çekirdek sayısı, çekirdek çalışma frekansı ve kopyalama hızı da donanımsal olarak fark yaratan bir unsurdur.

Şekil 4.4'de, GPU'nun diğer özelikleri de verilmektedir. Bellek kopyalama hızı da bir başka performans kriteridir. GPU'ların işlem sürelerini azaltmak için asenkron veri kopyalama özelliği mevcuttur. Bu durumda veri kopyalama ve hesaplama süresi örtüştürülebilir [18]. Nvidia grafik kartlarının tek yönlü akış özelliği, Nvidia Tesla ekran kartlarının ise çift yönlü veri akışı, mevcuttur.

Şekil 4.4'deki GPU'nun her bir özelliğinin ayrıntısı verilmeyecektir. Performansı etkileyen faktörler ağırlıklı olarak; hafıza kopyalama hızı, CUDA çekirdek sayısı, CUDA çekirdek hızı olarak kabul edilmiştir.

Algoritmanın koştuğu CPU cihazının özellikleri ise Şekil 4.5'de görülmektedir. Bu ekran çıktısı, CPUID açık kaynaklı programından alınmıştır.



Şekil 4.4 : Simülasyonlarda kullanılan CPU'nun donanımsal özellikleri.

Şekilde görülen CPU, Intel'in 3.7 GHz saat frekanslı, 8 çekirdekli, Xeon model işlemcisidir. CPU için de çekirdek sayısı ve çekirdeklerinin çalışma frekansı, performans açısından önemlidir. Ayrıca RAM'in çalışma frekansı da kopyalama açısından önemlidir. CPU'ların kullandığı RAM'ler, heterojen programlamada 'Host' üzerinde olduğu için kopyalama maliyeti değerlendirilmeyecektir. Genel olarak özellikleri verilen CPU donanımının fiyatı da GPU donanımından daha pahalıdır.

4.8 Çerçeve uzunluğunun belirlenmesi.

Çerçeve uzunluğu, CPU ve GPU algoritmalarının bir seferde işleyeceği veri miktarı kadardır. Bölüm 4.3'te çerçeve yapısı verilmiştir. CPU algoritması, herbir veriyi tek tek for döngüsü ile işlemesinden dolayı, çerçeve uzunluğu ile CPU'nun performansı arasında, kabaca bir yaklaşımla, lineer bir ilişki vardır diyebiliriz. Ancak GPU için bu durum farklıdır. GPU'da yaratılan iş parçacığı sayısı ile performansı arasında bir takım ilişkileri gözetmek durumdayız.

4.9 GPU, YTA algoritmasının blok ve iş parçacığı sayısı

GPU algoritması için blok ve iş parçacığı sayısının çarpımı, çerçeve uzunluğuna eşit ya da daha büyük olması gerekir. Çerçeve uzunluğu, bu çarpımın tam katı olması durumunda, bloklardaki iş parçacıkları her zaman eşit miktarda işlem yapacaktır. Aksi halde, bazı durumlarda çalışmayacak iş parçacıkları olacaktır. GPU algoritmasının performansını arttırmak için çalışmayan iş parçacığı sayısını minimize etmek gerekir. Çerçeve uzunluğu *L*, blok sayısı *B*, her bloktaki iş parçacığı sayısı *T* olmak üzere;

$$BT = L + x \tag{4.4}$$

4.4 eşitliğini oluşturalım. Bu ifadeyi minimum x ile gerçeklemek gerekir. Çünkü minimum x son blok için çalışmayacak iş parçacığı sayısıdır. Ayrıca, bu denklemin çözümünde, $B \leq 1024 \; ve \; T \leq 1024$ sınırları dikkate alınmalıdır. Şekil 4.3'de bakarak maksimum blok sayısı ve her bloktaki maksimum iş parçacığı sayısının 1024 olduğunu hatırlayalım. Bu denklemin her zaman tek bir çözümü olmayabilir. Çerçeve uzunluğunun yeterince uzun olduğunu kabul edelim (L>>1024). Blok sayısını 1024 olarak sabit kabul edip iş parçacığı sayısını hesaplayalım.

$$T = \left\lceil \frac{L}{1024} \right\rceil \tag{4.5}$$

Bu durumda son blok için çalışmayacak iş parçacığı sayısı *T*'den küçük olacaktır. Çözümün, her durum için uyarlamalı olduğunu söyleyemeyiz. Fakat blok sayısını maksimize ederek, her bloktaki iş parçacığı sayısını minimize ettğimiz için, son blok için çalışmayacak iş parçacığı sayısı en fazla *T-1* olacaktır.

Bir sonraki bölümde tasarlanan CPU ve GPU algoritmaları farklı senaryolar ile simüle edilecektir. GPU algoritmasını iyileştirmek için çerçeve boyutu belirlenecek ve GPU algoritmasının gerçek zamanda çalışma sınırları belirlenecektir.

5. PERFORMANS ANALİZİ

Bu bölümde, YTA algoritmasının CPU ve CPU üzerindeki performansları karşılaştırılacaktır. Ayrıca, YTA'nın GPU üzerindeki performansının en iyi olduğu aralıktaki parametrelerinde gerçek-zaman sınırları belirlenecektir.

Denklem 4.3'ü ele alalım.

$$O(.) = DFs\,\{\,(4N-2)\,Toplama + 4N\,Çarpma\,\} \qquad (\mathbf{5.1})$$

Bu denkleme göre YTA algoritmasındaki işlem sayısı, *N* (anten sayısı), *D* (Demet sayısı) ve *Fs*'e (örnekleme frekansı) bağlıdır. 3 parametre de işlem sayısını lineer artırmaktadır. *D* ve *Fs* parametresi, toplama ve çarpma işlem miktarlarının önünde çarpan olarak bulunmaktadır. *N* is çarpma işleminde daha fazla görünmektedir; fakat *N*'nin etkisi de lineerdir. Dolayısıyla işlem yükünün *N*'nin artışı ile de lineer artmasını bekleyebiliriz.

5.1 Anten sayısı ile işlem süresi arasındaki ilişki

Anten sayısının işlem süresi ile ilişkisi, CPU ve GPU algoritmalarının üzerinde incelenecektir. Bu sayede çok fazla anten elemanı gerektiren radar uygulamaları için de gerçek-zaman sınırı belirlenecektir. 5.1 denklemine göre algoritmanın işlem maliyetinin anten sayısı ile lineer arttığını kabaca bir yaklaşımla söyleyebiliriz. Denklem 5.1'deki işlem miktarını etkileyen diğer parametreleri ise bu simülasyonda sabitleyelim. Bu parametrelerin değeri ise şöyledir.

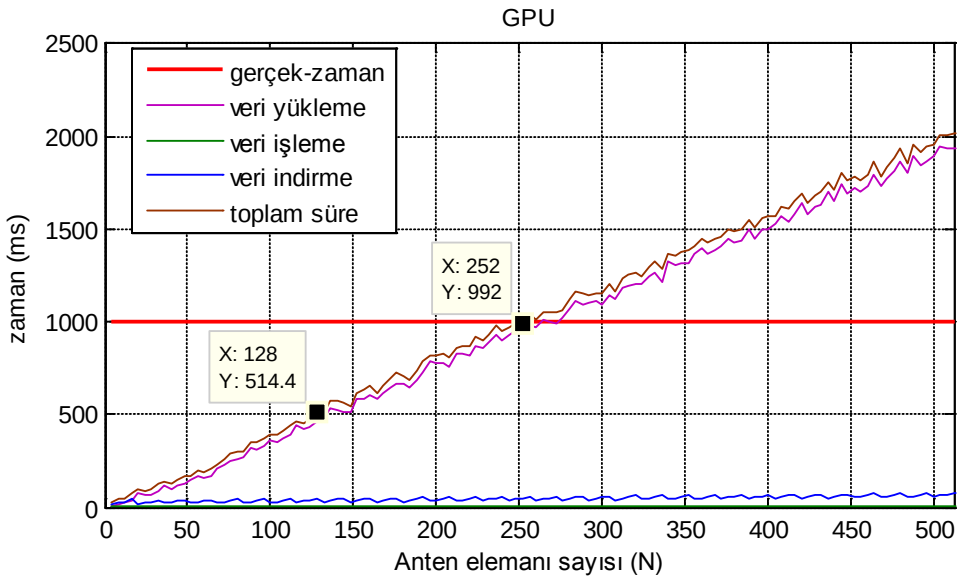
- *Fs* örnekleme frekansı, 2.5 MHz;
- Örnekleme çözünürlüğü, 16 bit reel ve 16 bit sanal bileşen;
- *D* demet boyutu, 1;
- Algoritmanın çerçeve boyutu, (2^{15})'dir. (Çerçeve boyutu, örneklerden oluşmaktadır)

Simülasyon için üretilen sinyalin parametreleri ise şunlardır.

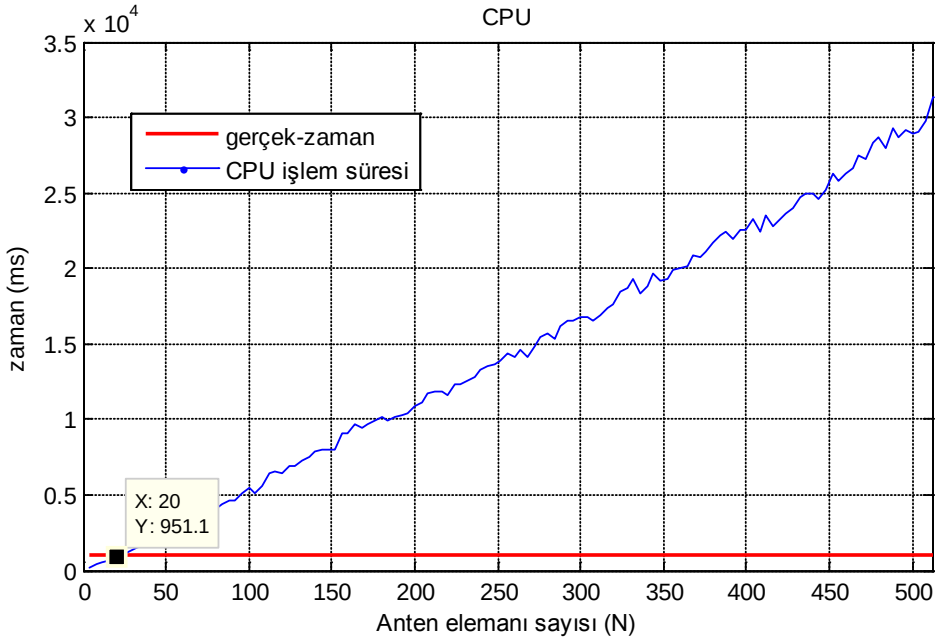
- f_c 2.5 GHz olan bir taşıyıcı;

- 2 Mhz frekanslı kompleks sinusoidal mesaj işareti;
- c ışık hızı $3x10^8$
- d antenler arasındaki mesafe $c/(2\ f_c)$ kabul edilmiştir.

Anten sayısı, 4'ten 512'ye kadar, 4er 4er artırılmıştır. Her anten sayısı ile simülasyon 1024 kez tekrarlanmıştır. Buna göre simülasyon sonucu, Şekil 5.1'de GPU için; Şekil 5.2'de ise CPU için elde edilmiştir.



Şekil 5.1 : GPU algoritmasının anten elemanı sayısı ile işlem süresi arasındaki ilişki



Şekil 5.2 : CPU algoritmasının anten elemanı sayısı ile işlem süresi arasındaki ilişki

Şekil 5.1'deki sonuçlarda GPU'nun işlem süresi görülmektedir. Daha önceki bölümlerde bahsettiğimiz GPU heterojen programlama için veri yükleme (t_u), veri işleme (t_e), veri indirme (t_d) süreleri bu grafikte görülmektedir. Ayrıca seri olarak yapılan bu üç süreye toplam süre (t_t) olarak tanımladık ($t_t=t_u+t_e+t_d$). Grafikten görüldüğü üzere; toplam süre t_u'nun etkisi altındadır. Düşük miktarda t_d'nin etkisi de vardır; ancak t_e çok düşük zaman harcadığı için grafiğin en alt kısmında kalmıştır. O halde veri işleme süresi çok düşüktür. Yani denklem 5.1'deki işlem yükünün hesapları, bu senaryoda çok etkisizdir. Daha çok veri kopyalama süreleri sonuçlara yansımaktadır. Anten elemanı sayısının artışı, GPU'ya kopyalanan veri yükleme miktarını arttırmıştır. Bu durumda t_u'nun etkisinin dominant olması beklenebilirdi. Bir başka gözlemlenmesi gereken husus, t_u'nun veri miktarının artışı ile lineer artış göstermesidir. O halde GPU'ya farklı boyutlarda yüklenecek veri miktarının harcadığı süreyi de kestirebiliriz.

Simülasyon parametrelerinden biçimlendirme demet boyutunu 1 olarak almıştık. Dolayısıyla simülasyonun her adımında geri dönen veri miktarı eşit olduğu için t_d seyrinde kalmıştır. O halde bu senaryoda t_d de toplam süre üzerinde etkisiz kalmıştır. Daha önceki bölümlerde bahsedilen, GPU'lu algoritmaların performanslarını sınırlayan etkilerin asenkron akış mekanizması ile çözülmesi, bu senaryo için mantıklı görünmemektedir; çünkü işlem süresinin çoğunu t_u oluşturmaktadır.

GPU algoritması bu senaryoda gerçek-zaman eşiğinin altında 252 antenli başarı göstermiştir. Anten sayısının daha da artırılmasına olanak sağlanması için örnekleme frekansının ya da örnekleme çözünürlüğünün düşürülmesi gerekmektedir. Örnekleme frekansı ve örnekleme çözünürlüğünün düşürülmesi, GPU'ya kopyalanacak veri boyutunu azalttığı için performans üzerinde lineer ve oransal bir etki göstereceğini söyleyebiliriz. Bu durumda, daha dominant olan t_u azalacak ve performans artışı sağlanacaktır. Ancak belirtmek gerekir ki; örnekleme frekansının ve örnekleme çözünürlüğünün azaltılması, işaret kalitesinin de düşürülmesi anlamına gelmektedir.

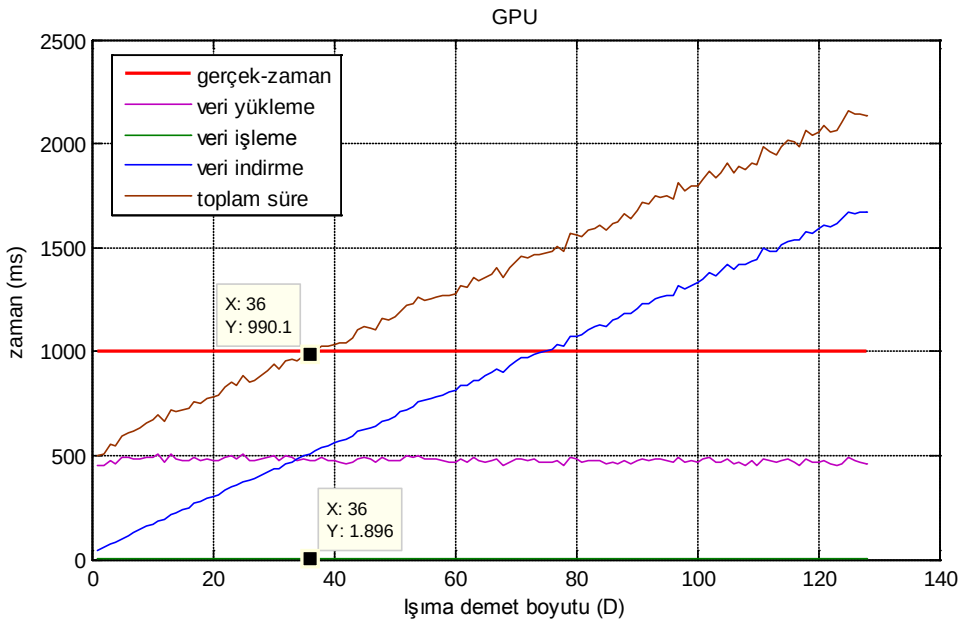
CPU için veri işleme süresi, toplam süre olarak tanımlanmıştır ve bu süre Şekil 5.2'deki grafikte aynı simülasyon senaryosu ile görülmektedir. Bu grafiğe göre; CPU'nun işlem süresi de benzer şekilde, anten sayısı ile lineer bir artış göstermiştir. Ancak CPU, 20 antenli bir sistemi gerçek zamanda işlemeyebilmiştir. CPU'nun

performansını arttırmak için bir takım paralelleştirme senaryoları literatürde mevcuttur; ancak CPU bunu gerçeklese bile çok fazla kaynak harcayacağı unutulmamalıdır. GPU'nun zaten doğasında olan paralel işlem yapabilme kabiliyeti, YTA algoritması için daha efektiftir. YTA algoritmanın işlem yükünü GPU'ya vermek CPU'yu da rahatlatacaktır. CPU bu sayede başka algoritmaları gerçeklemek için kaynak ayırabilecektir ve verimli heterojen bir yazılıma imkan tanıyacaktır.

5.2 Biçimlendirilen demet sayısı ile işlem süresi arasındaki ilişki

CPU algoritmasının performası, fazlaca yavaş olmasından dolayı, bu aşamadan sonra simülasyonlar sadece GPU algoritmansı üzerinden yapılacaktır.

Demet sayısı, aynı anten dizisi ile birden çok yönden gelen verilerin gerçek-zamanda alınması ve işlenmesi için arttırılmıştır. Böyle bir senaryonun demet katsayıları yardımıyla belli bir kapsama alanı oluşturması beklenmektedir. Bu senaryo aynı zamanda gerçek-zamanlı elektronik tarama gerektiren bir uygulama için de simüle edilebilir. Simülasyon parametreleri ise 5.1 bölümündeki parametrelerle aynıdır. Yalnız biçimlendirilen demet sayısı (D) simüle edileceğinden bu kez anten sayısı 128 olarak sabitlenmiştir ve demet sayısı serbest bırakılmıştır. Demet sayısı, 1'den 128'e kadar, 1er 1er artırılmıştır. Her demet sayısı ile simülasyon 1024 kez tekrarlanmıştır. Simülasyon sonucu Şekil 5.3'te görülmektedir.



Şekil 5.3 : GPU ışıma demet sayısı ile işlem süresi arasındaki ilişki.

Bu sonuca göre; GPU algoritmasının toplam harcadığı süre (t_t), 128 antenli, D=36 boyutlu YTA sistemini gerçek-zamanda işleme kabiliyeti göstermiştir. Veri işleme süresi (t_e), yine çok düşük mertebelerde kalmıştır. Oysa denklem 4.3'e göre anten elemanı sayısı ve biçimlendirilen demet boyutu işlem yükünü arttıran etkiye sahiplerdi. D=36 iken t_e 1.89 milisaniye olarak gözlemlenmektedir. Bu süre t_u ve t_d'e göre ihmal edilebilir düzeydedir. Bu demektir ki; toplam işlem süresi veri yükleme ve veri indirme sürelerinin etkisi altındadır.

Bir önceki simülasyonda asenkron kopyalama mekanizması GPU algoritmasının performansını arttırmak için önerilmemişti. Ancak bu senaryoda asenkron kopyalama mekanizması sistem performansını arttırabilir; bu sayede daha çok biçimlendirilmiş ışıma demeti elde etmeye imkan sağlayabilir. Çünkü bu senaryoda veri yükleme ve veri indirme süreleri birbirlerine göre baskın değildir.

5.3 Çerçeve boyutuna göre GPU algoritmasının performansı

GPU algoritmasının performansı çerçeve boyutuna bağlıdır. Çerçeve boyutu aynı zamanda GPU algoritmasının blok ve iş parçacığı sayısını değiştiren etkiye sahiptir. Bu bölümde çerçeve uzunluğunun performans üzerindeki etkisi incelenecektir. Simülasyon senaryosu şu şekildedir.

- *Fs* örnekleme frekansı, 20 MHz;
- Örnekleme çözünürlüğü, 16 bit reel ve 16 bit sanal bileşen;
- *N* anten sayısı, 8;
- *D* demet boyutu, 12;

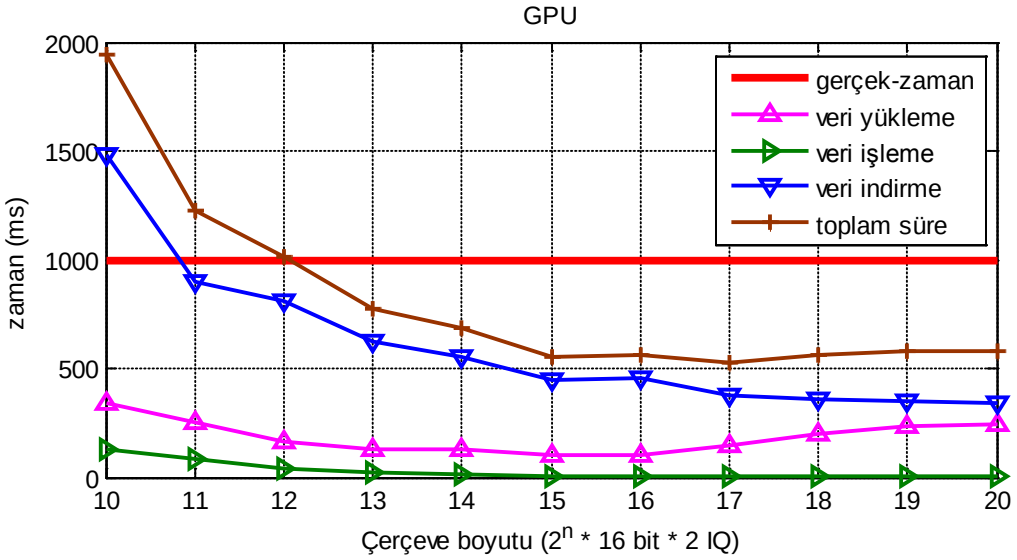
Simülasyon için üretilen sinyalin parametreleri ise şunlardır.

- f_c 1.8 GHz olan bir taşıyıcı;
- 2 Mhz frekanslı kompleks sinusoidal mesaj işareti;

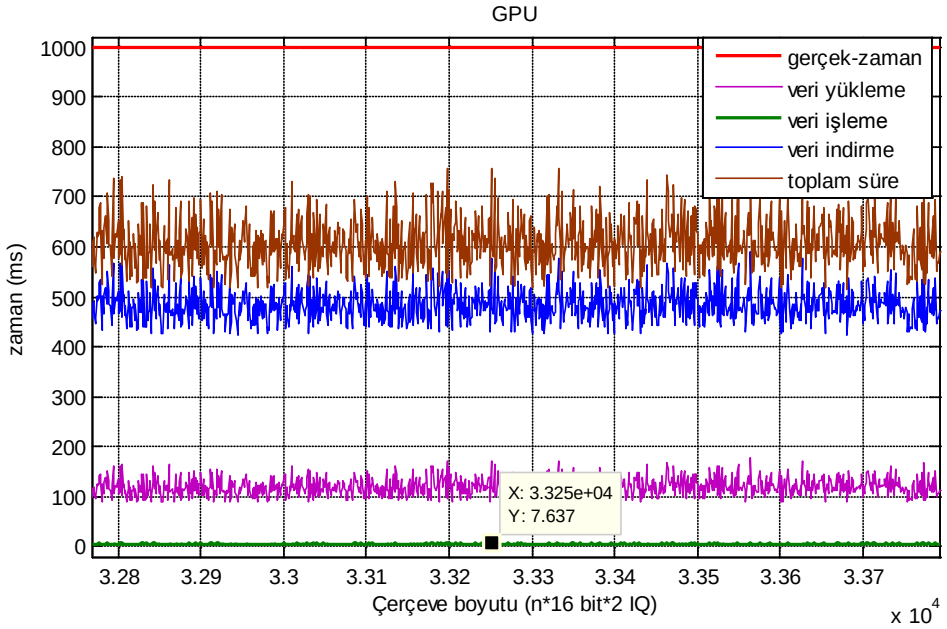
Bu senaryo, geniş bantlı haberleşme sistemleri için de bir performans göstergesi olması amacıyla belirlenmiştir. Öncelikli olarak çerçeve boyutunun optimum değerini bulmak için kabaca bir yaklaşımla simülasyon yapılmıştır. Bu nedenle ilk simülasyonda çerçeve boyutu, 2'nin kuvvetlerinden seçilmiştir. Seçilen çerçeve boyutuna göre, GPU algoritmasının blok ve iş parçacığı sayıları belirlenmiştir. Blok sayısı 1024 olarak sabitlenmiş ve iş parçacıklarının sayısı çerçeve boyutunun blok

sayısına bölünmesi ile belirlenmiştir. Bu durumda iş parçacığı sayısı çerçeve boyutunun 1/1024'üdür.

Gerçeklenen simülasyonun sonucu Şekil 5.4 ve Şekil 5.5'de görülmektedir.



Şekil 5.4 : GPU işlem sürelerinin, çerçeve boyutu ile değişimi (kabaca yaklaşım)



Şekil 5.5 : GPU işlem sürelerinin, çerçeve boyutu ile değişimi (hassas yaklaşım)

Şekil 5.4 ve Şekil 5.5'deki simülasyon sonucu, her çerçeve boyutu için 1024 kez tekrar edilmiştir. Böylece daha kararlı sonuçlar elde edilmesi amaçlanmıştır. Şekil 5.5'deki simülasyon sonucuna göre çerçeve boyutunun 2^{13}'e kadarki artışı, GPU'nun performansını arttırdığını, bu noktadan sonraki artışın ise GPU'nun performansını fazlaca değiştirmediğini söyleyebiliriz. Performans iyileşmesi tüm süreler (t_u, t_e, t_d)

için geçerlidir. Yine veri işleme süresi (t_e) en düşük zamanı harcamıştır. Bu senaryoda da veri indirme süresi (t_d) daha baskındır. Biçimlendirilen demet boyutunun anten elemanı sayısından daha fazla olması bu duruma yol açmıştır.

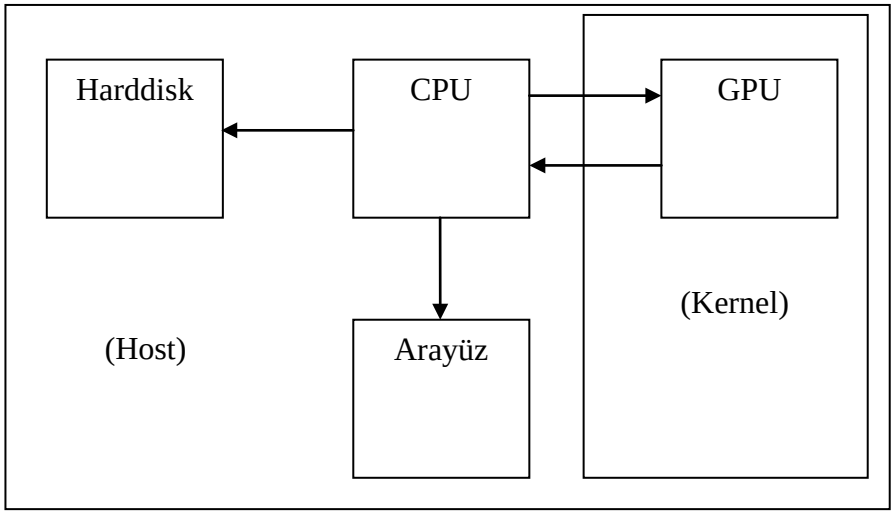
Şekil 5.5'de ise çerçeve boyutunun daha hassas değişimlerine karşı GPU algoritmasının performansı, simüle edilmiştir. Bu simülasyon her çerçeve değeri için 512 kez koşturulmuştur. Çerçeve boyutu ise 32768'den başlayarak 1er 1er arttırılmıştır. Son çerçevenin boyutu, başlangıçtaki çerçeve boyutunun 1024 örnek fazlasıdır. Bu simülasyonda çerçeve boyutunun 1024'ün katı olmadığı durumdaki performansı ölçülmek istenmiştir. Simülasyon sonucuna göre GPU algoritmasının performansı küçük değişimler ile hassas değişimlere tepki vermiş; yine de çerçeve boyutunun değiştiği tüm aralıkta, verileri gerçek zamanda işleyebilmiştir. Veri işleme süresi (t_e), GPU'lu YTA sisteminde baskın rol oynamadığından yine düşük kalmıştır.

Bölüm 4.9'da bahsedilen, çerçeve boyutunun blok ve iş parçacığı sayısının çarpımının tam katı olmaması durumunda, yaratılan iş parçacıklarının bazıları her zaman çalışmamaktadır. Bu durum, veri işlemede performans kaybına yol açması bekleniyordu. Ancak gerek çerçeve boyutunun yüksek değerlerde olması, gerek de t_e veri işleme süresinin simülasyon senaryolarında baskın rol oynamaması, performans kaybını net bir şekilde göstermemektedir.

Sonuç olarak GPU'da bir seferde işlenen veri çerçevesinin boyutu performansı bir noktaya kadar artırdığını, kabaca kestirilen çerçeve boyutunun etrafındaki hassas değişimlerin ise GPU algoritmasının performansını fazlaca etkilemediğini söyleyebiliriz.

6. YAZILIM ARAYÜZÜ

GPU programlamanın melez programlama olduğunu söylemiştik. Bu tez kapsamında tasarladığımız yazılım da melez yapıdadır. Şekil 6.1'e bakarak melez program için bazı kavramların üzerinde duralım.



Şekil 6.1 : Melez program yapısı

Melez programda CPU, Host; GPU cihazı ise Kernel konumundadır. Kernel'de veri işlemek için önce Host'tan Kernel'e veriler kopyalanır. Daha sonra Kernel'in cevabı, Host'a geri döndürülür. Host'un diğer cihazlarla erişimi vardır. Kernel ise herhangi bir cihaz ile doğrudan iletişim kuramaz; ancak Host üzerinden iletişim kurabilir. GPU'ların kendi aralarında, Host olmadan haberleşmesi mümkündür; ancak bizim senaryomuz bu şekilde değildir.

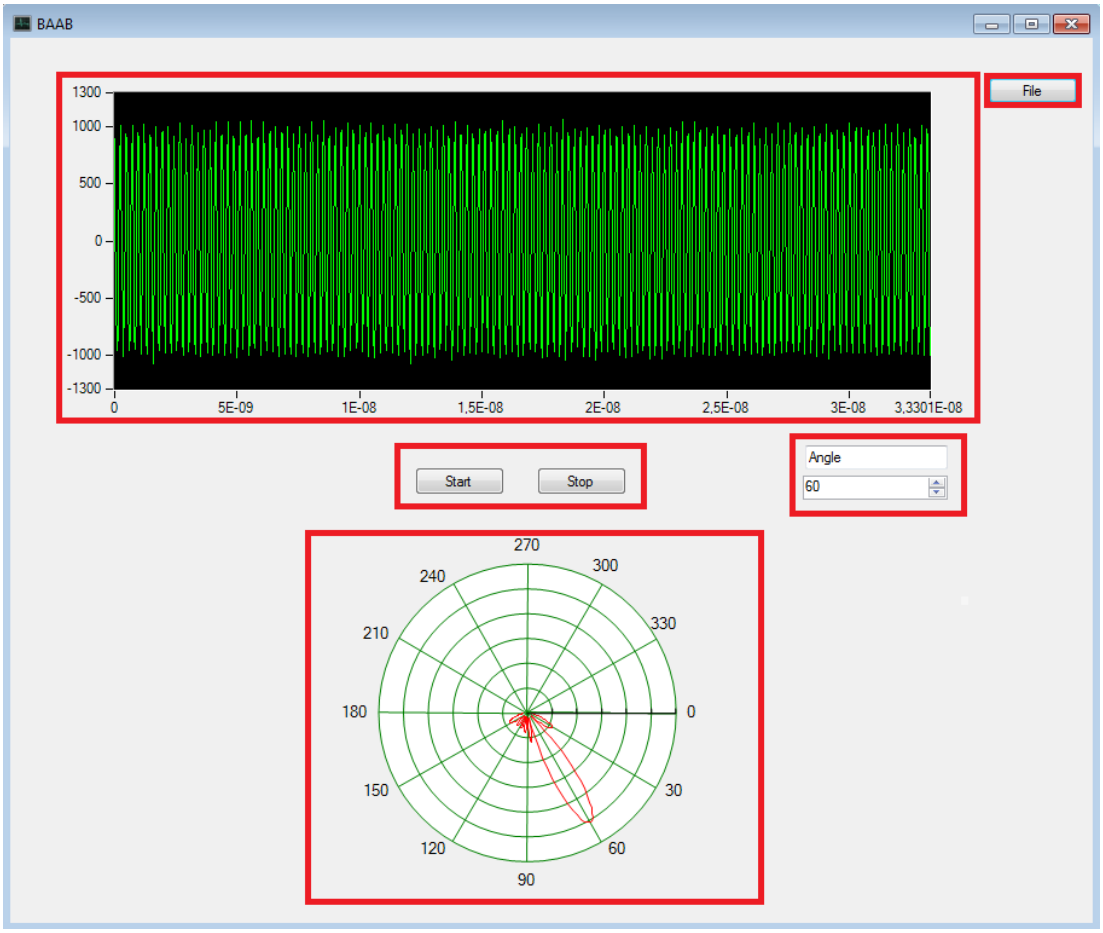
Bu yazılımda Host'un görevi simüle edilecek verileri harddisk'ten okumak, bu verileri kernel'e kopyalamak, elde edilen sonuçları geri almak ve eğer yazılımda arayüz var ise sonuçları arayüzde göstermektir. Kernel'in görevi ise, Host'tan aldığı çok büyük boyuttaki verileri anten verilerini biçimlendirmektir.

YTA, benzer işlemlerin çok defa tekrarlanmasını içerir. Dolayısıyla paralel işlemeye uygun bir yazılım tasarlamak mümkündür. Denklem 4.1'deki hesaplamalar tümüyle

GPU üzerinde yapılmaktadır. CPU ise yazılımın diğer parçalarıyla haberleşmeyi sağlamakta ve GPU'dan gelen verilerin enerjilerini hesaplayıp ışıma örüntüsünü oluşturmaktadır.

6.1 Yazılım ana penceresi

Yazılım arayüzü tek bir pencereden oluşmaktadır. Bu pencere üzerinde, dosya okuma tuşu, zaman bölgesi çizdirme penceresi, ışıma yönünü göstermek için polar grafik, açı değiştirme penceresi ve yazılım akışını başlatma ve durdurma tuşları bulunmaktadır. Şekil 6.2'de arayüz ana penceresi görülmektedir.



Şekil 6.2 : Yazılım arayüzü ana penceresi.

6.2 Zaman bölgesi çizdirme penceresi

Zaman bölgesi çizdirme penceresi, demet cevabı sonucu elde edilen işaret hakkında fikir vermesi açısından gösterilmiştir. Zaman bölgesi çizdirme penceresinde, elde edilen demet cevapları sürekli çizilmektedir. Bu penceredeki işaretler, işlem gücü

harcamaması için seyreltilmiştir. Yani bant genişliği büyük olan işaretlerin çiziminde hata olabilir. Bu durum göz ardı edilebilir.

Zaman bölgesi çizimleri, pencere pencere yapılmaktadır. İşlenen her pencere doğrudan çizilmektedir. Herhangi bir örtüşme yoktur. Bu yüzden pencere başında ve sonunda faz sürekliliği beklenmemelidir.

6.3 Açı değiştirme penceresi

Açı değiştirme penceresi ile açılar değiştirilerek, farklı kanalların zaman bölgesi çizimleri görüntülenir. Açı değiştirme penceresinin giriş değerleri tamsayılardır ve açı değerleri, 0 ile 180 derece arasında sınırlandırılmıştır. Bu pencerenin girişine verilen her açının bire-bir karşılığı hesaplanmaz. Yazılım, açı çözünürlüğüne göre kuantalanmış belli açıları hesaplamalarda kullanır. Giriş değerine verilen açının yerine en yakın komşusu hesaplamalarda kullanılır.

6.4 Polar grafik

Bu grafik, her bir kanalda hesaplanan gücü göstermektedir. Her bir kanala karşılık gelen açılar, polar grafiğin açı değerlerinde görülmektedir. Kanallarda elde edilen güç belirli bir süre içerisinde hesaplanmaktadır. Üretilen işaretlerin sürekli sinüsoidal işaret olmasından dolayı kanalın gücü herhangi bir aralık içerisinde hesaplanabilir. En genel halde kanalların gücü, sinusoidal işaretin 1 periyot süresinde hesaplanmıştır. Paket paket iletilen ya da periyodik olmayan işaretlerde kanal gücünün hesaplamalarında hatalı sonuçlar olabilir. Özellikle paketlerin iletilmediği aralıklarda hesaplama yapılması o kanalın gücünü düşük gösterecektir. Fakat periyodik olmayan işaretler için ışıma örüntüsü, genel halde değişmeyecektir.

Polar grafiğin çözünürlüğü açı çözünürlüğüne bağlıdır. Dolayısıyla ne kadar fazla demet cevabı varsa o kadar net bir grafik elde edilecektir. Polar grafiğin açı ekseni, derece cinsinden etiketlenmiştir. Çizimler, [0 180) derece arasında yapılmaktadır.

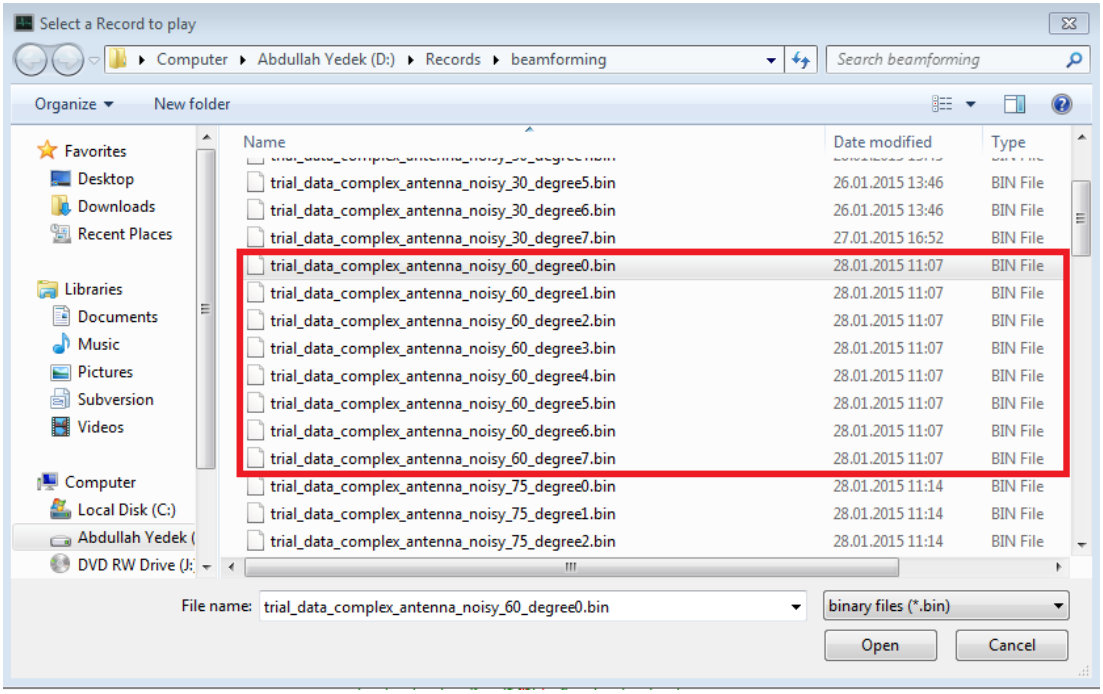
6.5 Start-Stop düğmesi

Start-Stop düğmesi, veri akışını başlatmak ve durdurmak için eklenmiştir. Yazılım açıldığında veri akışı başlamaz. Veri akışı, start düğmesi ile yapılır. Ancak, yazılımda giriş verisi yoksa, veri akışı başlamaz, yazılım bu durumda kendini

kapatır. Öncelikle simüle edilecek verinin sisteme verilmesi gerekir. Eğer yazlım çalışırken veri akışı durdurulursa, tekrardan başlatıldığında veri akışı kaldığı yerden devam eder.

6.6 File düğmesi

Bu düğme ile dosya okuma işlemi yapılır. Dosya okuma işleminin öncesinde anten sayısı kadar dosya olmalıdır. Dosya isimlerinin sadece son karakteri, anten elemanına özgü indeksi içermelidir. Aksi halde dosya okuma işlemi gerçekleşmeyecektir. Pencere açıldıktan sonra, sıfır indeksli dosya seçilip "Open" tuşuna basılmalıdır. Bu durumda, diğer dosyalar da otomatik olarak okunacaktır. Şekil 6.3'de File düğmesine basınca ekranda çıkan dosya açma penceresi görülmektedir. 8 anten için 8 farklı dosya bulunmaktadır.



Şekil 6.3 : Dosya okuma penceresi.

7. SONUÇ

Uyarlamalı anten dizileri ya da akıllı anten teknolojisi, kablosuz haberleşme, mobil iletişim, uydu haberleşmesi ve RADAR uygulamaları için esnek, ucuz ve başarılı sistemler sunmaktadır. Bu tezde sunulan YTA, akıllı algoritmaları uygulanabilir kılmasıyla beraber ucuz ve esnek bir çözüm önerdi. YTA sistemi alıcı anten modeli üzerinden tasarlandı. Sistemin bileşenleri, bir dizi resiprok ve yönsüz anten dizisi, RF frekans alçaltıcı elemanları, analog/sayısal çevirici ve yazılım ile kontrol edilebilir biçimlendirici modülünden oluşmaktadır. Ancak anten ve RF elemanlarının, sayısallaştırıcının gerçeklemesi tez kapsamında yapılmadı. Dolayısıyla bu elemanların YTA sistemi üzerindeki performansı analiz edilmedi. Simülasyonlar için MATLAB'da sentetik işaretler üretildi.

YTA sisteminin anten ışıma örüntüsü, lineer anten dizisi ve faz tarayıcı anten katsayıları üreten bir yapı baz alınarak çıkartıldı. Işıma örüntüsüne göre YTA, istenildiği takdirde bir çok yöne yönlendirilebileceği gibi, istenmeyen yönlerden gelen işaretleri de zayıflatabilmeye imkan verebilmektedir. YTA aynı anda birden çok ışıma yönüne kanalize olması yüksek işlem gücü gerektirdi. İşlem yüküne çözüm olarak CPU ve GPU algoritmaları tasarlandı ve simülasyonları yapıldı. Paralelleştirilen YTA algoritması, GPU üzerinde yüksek performans gösterdi, bu sayede geniş bantlı veriler bile gerçek-zaman eşiğinin altında, hızlıca işlendi. CPU algoritması ise çok yavaş kaldı ve verileri gerçek zamanda işleyemedi.

GPU'ların performansının tek seferde algoritmaya verilen veri hacmine bağlı olduğu, veri hacmi artıkça GPU'ların daha iyi performans sergilediği bilinmekteydi. Bu nedenle YTA için bir çerçeve boyutu belirlenmesi gerekiyordu. Simülasyonlar ile GPU algoritmasının önce kaba sonra hassas boyutlardaki değişimlere karşı performansı belirlendi. GPU algoritması, çerçeve boyutunun artışına bir noktaya kadar performansını artırarak cevap verdi, sonrasında ise fazlaca değişiklik göstermedi. Bu durum GPU ile gerçeklenen YTA sisteminin başka sistemlere entegrasyonun ve adaptasyonun hızlı ve başarılı olacağının göstergesidir.

YTA'nın GPU üzerindeki işlem süresi iki farklı işlem süresi ile değerlendirildi. Birincisi GPU'nun sadece işlem yapma süresi, ikinci ise GPU'nun seri olarak yaptığı veri yükleme, veri işleme ve veri indirme sürelerini içeren toplam süre olarak tanımlandı. GPU üzerindeki YTA algoritması, çok hızlı veri işlemesine rağmen verileri yükleme ve verileri geri indirme noktasında, çok büyük boyutlarda transfer yaptığından ve donanımlar arasında iletim bant genişliğinin sınırlamalarından dolayı yavaş kaldı. Buna rağmen GPU toplam harcanan sürede YTA algoritmasını, çoğu kez gerçek zaman eşiğinin altında işledi. Toplam harcanan sürede iyileştirme yapmak içim çözüm olarak asenkron kopyalama senaryosu önerildi; fakat bu tez kapsamında asenkron senaryo simüle edilmedi.

Sonuçta GPU ile sunulan YTA esnek, ucuz ve başarılı bir çözümdür. YTA GPU'ların gelişmesi ile birlikte işlem yapabilme kabiliyetini arttıracaktır. YTA, alıcı model üzerinden tasarlanmasına rağmen küçük revizyonlar ile verici olarak da işlev görebilir. Ayrıca YTA'nın yazılımsal yapısı farklı sensor uygulamaları için de revize edilebilir.

KAYNAKLAR

[1] Chryssomallis, M. (2000). Smart antennas. *Antennas and Propagation Magazine*, 42(3), 129-136.

[2] Winters, J. H. (1998). Smart Antennas for wireless system. *Personal Communications, IEEE*, 5(1), 23-27.

[3] Ertel, R. B., Cardieri, P., Sowerby, K. W., Rappaport, T. S., & Reed, J. H. (1998). Overview of spatial channel models for antenna array communication systems. *Personal Communications, IEEE*, 5(1), 10-22.

[4] Widrow, B., Duvall, K., Gooch, R. P., & Newman, W. (1982). Signal cancellation phenomena in adaptive antennas: Causes and cures, *Antennas and Propagation, IEEE Transactions on*, 30(3), 469-478.

[5] Seo, J., Chen, Y. H., De Lorenzo, D. S., Lo, S., Enge, P., Akos, D., & Lee, J. (2011) A Real-Time Capable Software-Defined Receiver Using GPU for Adaptive Anti-Jam GPS Sensors, *Sensors*, 11(9), 8966-8991.

[6] Hottinen, A., Kuusela, M., Hugl, K., Zhang, J., & Raghothaman, B. (2006). Indrustrial embrace of smart antennas and MIMO. *Wireless Communications, IEEE*, 13(4), 8-16.

[7] Geise, A., Jacob, A. F., Kuhlmann, K., Pawlak H., Gieron, R., Siatchouna, P., Lohmann, D., Holzwarth, S., Litschke, O., Heckler, M. V. T., Greda, L., Dreher, A., & Hunscher, C. (2007). Smart antenna terminals for broadband mobile satellite communications at Ka-Band. Antennas, *ITG-Fachbericht-INICA 2007*, Munich

[8] Fenn A. J., Temme D. H., Deleney W. P., & Courtney W. E. (2000)**.** The development of phased-array radar technology. *Lincoln Laboratory Journal*, 12(2), 321-340

[9] Yun, S. W., Jin, Y., Hyeon, S., & Choi, S. (2014, Haziran). Implementation of 4-element beamforming antenna module for B4G base station system using GPU. *In Consumer electronics (ISCE 2014), The 18th IEEE International Symposium on*, Jeju Island, (Sf.1-2). IEEE.

[10] Kohno, R. (1997). Software Antenna and Its Communication Theory for Mobile Radio Communications. *Personal Wireless Communications, 1997 IEEE International Conference*, Mumbai, (Sf. 227-233).

[11] Ali Akbarian, H., Volskiy, V., Wolhuter R., & Vandenbosch, G., (2010, Nisan). Analogue versus digital for baseband beam steerable array used for LEO satellite applications. *In 4th European Conference on Antennas and Propagation (EuCAP) 2010*, Barcelona, (Sf. 1-4).

[12] Balanis, C. A. (2012). *Antenna Theory: Analysis and Design*, John Wiley & Sons. Cilt 2, Bölüm 6.

[**13] Haynes, T.** (1998). A Primer on Digital Beamforming, *Spectrum Signal Processing, 11.*

[14] Mitola, J. (1995). The software radio architecture. *Communication Magazine, IEEE*, 33(5), Sf.26-38.

[15] Lee, D., Dinov, I., Dong, B., Gutman, B., Yanovsky, I., & Toga, A. (2011). CUDA optimization strategies for compute-and memory-bound neuroimaging algorithms. *Computer methods and programs in biomedicine,* Amsterdam, 106(3), 175-187.

[16] Keckler, S. W., Dally W. J., Khailany, B., Garland, M., & Glasco, D. (2011). GPUs and the future of parallel computing. *IEEE Micro,* 31(5), 7-17.

[17]Nickolls J., Dally W. (2010). The GPU Computing Era, *IEEE Micro,* 30(2), 56-69.

[18]Stam J. (2009, Ekim). Maximizing GPU efficiency in extreme throughput applications. *In Proceedings of the GPU Technology Conference 2009.*

[19]Edwards N. (2013). Theoretical vs. actual bandwidth: PCI express and thunderbolt, Alındığı tarih:4.5.2015, Adres: http://www.tested.com/tech/457440-theoretical-vs-actual-bandwidth-pci-express-and thunderbolt/

[20] Galloy M. (2013). CPU vs GPU performance, Alındığı tarih: 4.5.2015,Adres: http://michaelgalloy.com/2013/06/11/cpu-vs-gpu-performance.html

[21] (2012) Beamforming math, *mathscinotes*, Alındığı tarih: 4.5.2015, http://www.mathscinotes.com/2012/01/beamforming-math/

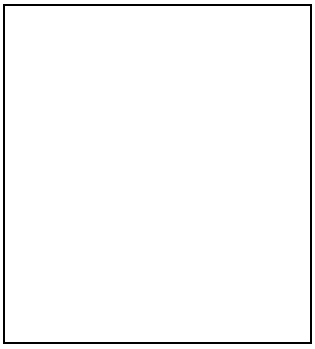
[22] GPU-accelerated Aplications, Alındığı tarih:4.5.2015, Adres: http://www.nvidia.com.tr/content/gpu-applications/PDF/GPU-apps-catalog- mar2015.pdf

[23] 2015,CUDA C Programming Guide, NVIDIA Corporation. Alındığı tarih: 4.5.2015,Adres: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[24] Nvidia. GeForce GTX 680 Specifications, Alındığı tarih: 03.05.2015, Adres: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-680/ specifications

[25]CUDA C Best Practice Guide, Mart 2015, Alındığı tarih: 03.05.2015, Adres: http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_ Guide.pdf

ÖZGEÇMİŞ



Ad Soyad **: Abdullah Bakırtaş**

Doğum Yeri ve Tarihi **: Trabzon, 1989**

E-Posta **: abdullah.bakirtas@tubitak.gov.tr**

ÖĞRENİM DURUMU:

- **Lisans** **:** (2007-2012), İstanbul Teknik Üniversitesi, Elektrik Elektronik Fakültesi, Telekomünikasyon Müh.
- **Yükseklisans** **:** (2012-2015), İstanbul Teknik Üniversitesi, Elektronik Haberleşme Müh., Telekomünikasyon