İSTANBUL TEKNİK ÜNİVERSİTESİ

ELEKTRİK – ELEKTRONİK FAKÜLTESİ

GÜVENLİ RFT SİSTEMLERİ İÇİN BİR GRAFİK İŞLEMCİSİ ÜZERİNDE SUNUCU TASARIMI VE GERÇEKLENMESİ

BİTİRME ÖDEVİ

KENAN ERDOĞAN

040060306

Bölümü: Elektronik ve Haberleşme Mühendisliği Bölümü

Programı: Elektronik Mühendisliği

Danışmanı: Yrd. Doç. Dr. Sıddıka Berna ÖRS YALÇIN

MAYIS 2011

ÖNSÖZ

Tez çalışmam boyunca bana yol gösteren ve desteğini esirgemeyen danışman hocam Yrd. Doç. Dr. Sıddıka Berna ÖRS YALÇIN'a teşekkürlerimi sunarım.

Ayrıca öğrenim hayatım boyunca beni yalnız bırakmayan ve desteklerini esirgemeyen arkadaşlarıma ve aileme teşekkürlerimi sunarım.

MAYIS 2011 Kenan ERDOĞAN

İÇİNDEKİLER

KISALTMALAR

RFT : Radyo Frekanslı Tanıma

GİB : Grafik İşlemci Birimi

MİB : Merkezi İşlem Birimi

TBCM : Tümleşik Birim Cihaz Mimarisi

OSKP : Ohkubu, Suzuki ve Kinoshita'nın protokolü

GÖA-1 : Güvenli Özet Algoritması-1

OSK/AOP : OSKP'nin sunucu yapısının Gildas Avoine ve Philippe Oechslin tarafından geliştirilmesiyle oluşan yapı

TBCM C : Tümleşik Birim Cihaz Mimarisi C

GİBF : Grafik İşlemci Birimi Fonksiyonu

TABLO LİSTESİ

ŞEKİL LİSTESİ

GÜVENLİ RFT SİSTEMLERİ İÇİN BİR GRAFİK İŞLEMCİSİ ÜZERİNDE SUNUCU TASARIMI VE GERÇEKLENMESİ

ÖZET

Zaman zaman medya tarafından internetten sonra en büyük teknoloji devrimi olarak gösterilen Radyo Frekanslı Tanıma (RFT) (Radio Frequency Identification - RFID) sistemleri günümüzde gelişen teknoloji ile beraber oldukça geniş bir kullanım alanına sahip olmaya başlamıştır. Fiziksel bir etkileşim olmadan, uzaktan ve havadan kitle tanımlaması özelliği ile beraber RFT etiketleri toplu taşıma sistemlerinin kartları, stokların güncel durumunun takibi, kütüphanelerde kitapların etiketlenmesi gibi bir çok durum için kullanılmaya başlanmıştır. Ancak RFT etiketlerinde kullanıcı mahremiyeti açısından ele alınması gereken güvenlik problemleri vardır. Bu çalışmada bu güvenlik problemlerinin birçoğu giderilerek geliştirilen protokollerden Ohkubu, Suzuki ve Kinoshita'nın protokolü (OSKP) ve OSKP'nin sunucu yapısının Gildas Avoine ve Philippe Oechslin tarafından geliştirilmesiyle oluşan yapı (OSK/AOP) ele alınacak ve bu iki farklı sunucu yapısı bir grafik işlemcisi üzerinde gerçeklenecektir.

Ayrıca çok pahalı sistemler kullanılmadığı sürece binlerce etiketin tanımlanması gerektiği durumlarda RFT sistemleri için yapılan sunucular çok yavaş çalışır. Bu çalışmada da çok fazla etiket ile gerçekleme yapılmıştır ve bu süre problemini aşmak için diğer hızlı sistemelere göre oldukça ucuz olan, günümüzde önemi ve kullanım alanı oldukça artan, NVIDIA tarafından geliştirilen Tümleşik Birim Cihaz Mimarisi (TCBM) (Compute Unified Device Architecture - CUDA) paralel programlama mimarisi kullanılmıştır. Bu çalışmada gerçekleme işlemi NVIDIA Geforce GTX 275 grafik kartı kullanılarak yapılmış ve gerekli analizler yapılarak sunucular karşılaştırılmalı olarak anlatılmıştır.

DESIGN AND IMPLEMENTATION OF A SERVER ON GRAPHIC PROCESSOR FOR SECURE RFID SYSTEMS

SUMMARY

Sometimes presented by the media as the next technological revolution after the internet, Radio Frequency Identification (RFID) systems nowadays started to be used in wide range of areas. With the ability of mass identification remotely, on air and without physical contact, RFID tags have been used in areas such as public transportation cards, tracing livestocks and labeling books in libraries etc. However, on RFID tags there are security problems about consumer privacy. In this study the protocol proposed by Ohkubu, Suzuki and Kinoshita (OSKP) and the same protocol but whose server is developed by Gildas Avoine and Philippe Oechslin (OSK/AOP), which resolves many of these security problems, are going to be examined and whose servers are implemented on a Graphichs Processing Unit (GPU).

Moreover, unless using such expensive systems, in the situations that thousands of tags have to be identified the servers for RFID systems works very slowly. In this study the implementations are also made with many tags and to resolve the time problem Compute Unified Device Architecture (CUDA), which is developed by NVIDIA and has rising importance and usage area, is used for the implementation. In this study the implementation is done with the display card NVIDIA Geforce GTX 275 and these two servers are compared with the necessary analyses.

1. GİRİŞ

Bu çalışmada Radyo Frekanslı Tanıma (RFT) (Radio Frequency Identification - RFT) sistemleri için Grafik İşlemci Birimi (GİB) (Graphichs Processing Unit – GPU) üzerinde güvenli bir sunucu tasarlanması ve gerçeklenmesi amaçlanmıştır.

Aynı anda her yerde mevcut olabilen ağ bağlantıları sayesinde toplumun her türlü karmaşık hizmete her yerde ve her zaman erişimi kolayca sağlanabilecektir ve bu yapının en önemli parçalarından biri RFT sistemleridir [1]. Günümüzde çokça kullanılan optik barkodların aksine RFT etiketlerinin birçok faydası vardır. Özellikle kitle tanımlaması en büyük özelliğidir, çünkü RFT etiketleri ile birbirinden farklı birçok kimlik aynı anda, uzaktan, hızlıca ve toplu olarak tanıma işlemine sokulabilir. Ayrıca RFT sistemlerinin kullanılması sahtecilik yapılmasını da zorlaştırmaktadır.

Ancak RFT sistemlerinin bunca faydasının yanı sıra kullanıcı mahremiyeti gibi güvenlik problemlerinden de bahsedilmelidir. Bir yandan RFT ile ilgili beklentiler artarken öte yandan RFT kullanımıyla ilgili kaygılar da artmaktadır. Özellikle kullanıcı mahremiyeti çok önem kazanmaktadır [1].

Bu sistemlerdeki temel problem RFT etiketlerinin temel fonksiyonundan kaynaklanmaktadır. Çünkü RFT etiketlerinin tanımlanması uzaktan yapıldığı için bir başkası tarafından da uzaktan taranabilir ve RFT etiketleri her şeye otomatik olarak cevap verir, bilgisini aktarır. Böyle bir durum bir kişinin izlenebilmesine, geçmiş hareketlerine dair bilgi edinilmesine ya da bilgilerinin çalınıp kullanılabilmesine neden olacaktır [1].

Bunun gibi birçok problem göz önünde bulundurularak birçok RFT kimlik doğrulama protokolü geliştirilmiştir. İşte bu kimlik doğrulama protokollerinden bir

tanesinin incelenmesi ve bu protokol için tasarlanan iki farklı sunucunun gerçeklenmesi bu tezin çalışma konusunun bir kısmını oluşturmaktadır.

Bu çalışma boyunca bu sunucuların gerçeklenmesi için yapılan iki temel gerçekleme bir veri tabanı oluşturulması ve bu veri tabanı üzerinde etiket arama işleminin yapılmasıdır. Ancak bu işlemlerin Merkezi İşlem Birimi (MİB) (Central Processing Unit - CPU) üzerinde yapılması çok pahalı yapılar kullanılmadığı sürece çok uzun sürmektedir. Bu amaçla bu protokol için iki ayrı sunucu gerçeklenmesi bir GİB üzerinde yapılmıştır ve bunun için NVIDIA tarafından geliştirilen GİB üzerinde çalışan Tümleşik Birim Cihaz Mimarisi (TBCM) (Compute Unified Device Architecture - CUDA) paralel programlama mimarisi kullanılmıştır [2]. Bu yapı günlük hayatta kullandığımız masaüstü ve dizüstü bilgisayarların grafik kartlarının birçoğu tarafından desteklenmektedir ve bu yapıyla gerçeklenen işlemler MİB yerine GİB üzerinde GİB'nin çoklu çekirdek yapısı özelliği kullanılarak paralel olarak gerçekleştirilir ve böylece kullanılan grafik kartının özelliklerine göre değişen sayıda binlerce işlem aynı anda paralel olarak gerçekleştirilebilmektedir. Bu sayede gerçeklemelerin çok daha hızlı bir şekilde yapılması hedeflenmiştir. İşte bu mimarinin öğrenilmesi ve gerçeklemelerin bu mimari yardımıyla yapılması da tezin çalışma konusunun diğer kısmını oluşturmaktadır.

Bu amaçla birinci kısımda çalışmanın önemi anlatılmış, ikinci kısımda gerçeklenen sunucular için protokol anlatılmış, üçüncü kısımda TCBM paralel programlama mimarisi ve TCBM C hakkında bilgi verilmiş, dördüncü bölümde kullanılan grafik kartı ile ilgili bilgiler verilmiş, TCBM ile bu sunucuların gerçeklenmesi ve yapılan analizler anlatılmış ve son kısımda gerçeklemeler sonucunda elde edilen sonuçlar yorumlanmıştır.

2. RFT KİMLİK DOĞRULAMA PROTOKOLLERİ

Bu bölümde çalışma boyunca sunucu kısmı iki farklı şekilde gerçeklenen protokol ele alınacak ve çalışma prensipleri anlatılacaktır ancak öncelikle RFT sistemleri ve bu sistemlerdeki güvenlik problemleri hakkında bilgi vermekte fayda vardır.

2.1. RFT Sistemleri

RFT sistemleri etiket (E), okuyucu (O) ve sunucu (S) olmak üzere üç temel yapıdan oluşmaktadır [1].

Etiket kablosuz bir yapıdır, içinde kimlik bilgisini bulundurur ve okuyucu tarafında radyo frekansında bir istek gönderildiği zaman içindeki bilgiyi radyo frekansı kanalından okuyucuya aktarır. Okuyucu da kablosuz bir yapıdır, kimlik doğrulama için etikete istek gönderir ve etiketin gönderdiği kimlik bilgisini alarak sunucuya aktarır. Sunucuda bir veri tabanı vardır ve sunucu okuyucudan aldığı bilgiyi bu veri tabanındaki bilgi ile karşılaştırarak doğrulama işlemini yapar.

2.2. Güvenlik Problemleri

RFT sistemlerindeki temel problem etiketlerin en temel özelliğinden kaynaklanmaktadır, yani etiketlerin kendilerine gelen her isteğe karşılık vermesi, bilgisini aktarması [4]. Bu özellikten dolayı saldırganlar istediği etiketin içindeki bilgiyi etikete istek göndererek alabilirler. Ayrıca saldırganların kullanabileceği başka bir yöntem de gene RFT sistemlerin temel özelliği ile ilgilidir, yani iletişimin havadan yapılması. Bu yüzden saldırgan istediği zaman etikete istek göndermesine gerek kalmaksızın okuyucu ve etiket arasındaki iletişimi takip ederek etiket içindeki bilgiyi çalabilir. Ayrıca saldırgan etikete izinsiz olarak müdahale ederek içindeki kimlik bilgisine ulaşabilir.

Bu şekilde etiket içindeki bilginin ya da etiket tarafından gönderilen bilginin çalınması kullanıcı mahremiyeti açısından iki noktada korku yaratır. Bunlardan ilki herhangi bir okuyucunun etiket içindeki bilgiyi istediği zaman alabilmesinden kaynaklanan kişisel bilgilerin çalınmasıdır. Çünkü bu etiketlerde kullanıcıların kişisel bilgileri de yer alabilmektedir ve kullanıcılar genellikle kişisel bilgilerinin yabancılar tarafından öğrenilmesini istemezler. İzinleri olmaksızın bir başkasının bu bilgileri çalabilme olasılığı da kişilerde kaygı uyandırır.

Bir başka kaygı noktası ise kişilerin takip edilebilmesidir. Çünkü kişilerin taşıdığı herhangi bir etiket ile kişi eşleştirilebilir ve böylece o etiket takip edilerek kişi de takip edilmiş olur. Kişilerin takip edilebilmeleri çok büyük bir mahremiyet problemidir ve kullanıcılarda büyük bir korku uyandırır.

Ayrıca, etiket içindeki kimlik bilgisinin bir kapıyı açmaya yaradığını düşündüğümüzde hırsızın etiket içindeki bilgiyi kullanarak bu kapıdan izinsiz olarak girebileceğini görebiliriz. Bu bir kişiye ait bilgi ile bir başkasının mahremiyetine yapılabilecek bir saldırıya örnek olarak verilebilir.

İşte anlatılan bu problemleri gidermek RFT sistemleri için geliştirilen güvenlik protokollerinin başlıca sağlaması gereken görevlerdir.

2.3. Ohkubu, Suzuki ve Kinoshita'nın Protokolü

Ohkubu, Suzuki ve Kinoshita'nın Protokolü (OSKP) güvenlik problemlerini gidermek için özet fonksiyonlarından yararlanmaktadır [3]. Bu yüzden öncelikle özet fonksiyonları hakkında bilgi vermekte yarar vardır [5].

2.3.1. Özet Fonksiyonları

Kriptolojide çokça kullanılan bu fonksiyonlar çok uzun olabilecek verileri giriş olarak alır ve giriş uzunluğu ne olursa olsun sabit uzunlukta çıkış üretirler [5]. Bu fonksiyonların başlıca özellikleri şunlardır:

- Bu fonksiyonlar ile çok uzun boyutlu veriler çok daha kısa boyutlu verilere dönüştürülerek tutulabilir ve bu da bu veriler üzerinde yapılacak işlemleri hızlandırır.
- Tersinir değildirler, yani çıkışındaki veriden yola çıkılarak girişindeki veri elde edilemez.

- Özet fonksiyonları farklı girişler için farklı çıkışlar üretirler, girişteki veriler arasında sadece bir bit fark bile olsa çıkışları birbirinden farklı olur.

Yukarıda belirtilen özellikler doğrultusunda aşağıdaki Tablo 2.1'de dördüncü bölümde anlatılacak olan Güvenli Özet Algoritması-1 (GÖA-1) (Secure Hash Algorithm-1 – SHA-1) kullanılarak özet fonksiyonu girişlerine ve çıkışlarına örnek verilmiştir.

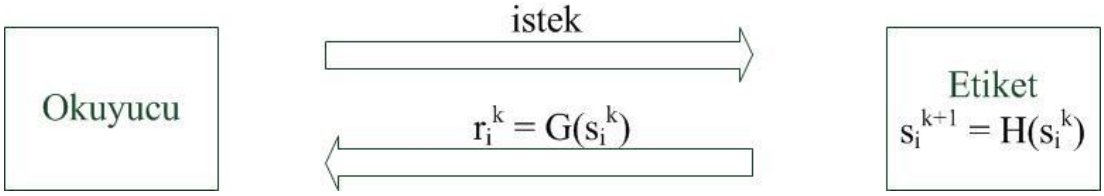
Tablo 2.1 Çeşitli girişler ile GÖA-1 kullanılarak üretilen çıkışlar

Giriş	Çıkış
"abcde"	3de6c57 bfe24bf c328ccd7 ca46b76e adaf4334
"abcdf"	9693da0e 85af20e f1f982b0 17fc6ec2 419848e5
"abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde"	f00079d0 6a357c61 13b96dcd 719b03ca 77557a02

Tablo 2.1'de de görüldüğü üzere her farklı giriş için fonksiyon çıkışı farklıdır ve giriş uzunluğu ne olursa olsun çıkış uzunluğu sabittir (Burada kullanılan GÖA-1 fonksiyonu için çıkış 160 bittir).

2.3.2. Ohkubu, Suzuki ve Kinoshita'nın Protokolünde Okuyucu ile Etiket Arasındaki İletişim

OSKP'de okuyucu etikete istek gönderdiğinde etiket, içindeki gizli kimlik bilgisini (s_i^k) bir özet fonksiyonundan (G) geçirdikten sonra bu bilgiyi ($r_i^k = G(s_i^k)$) okuyucuya gönderir ve kendi içindeki bilgiyi başka bir özet fonksiyonundan (H) geçirerek yeniler ve bu yenilenmiş kimlik bilgisini ($s_i^{k+1} = H(si_i^k)$) saklar [3,4]. Şekil 2.1'de bu işlemi daha net olarak görebiliriz.



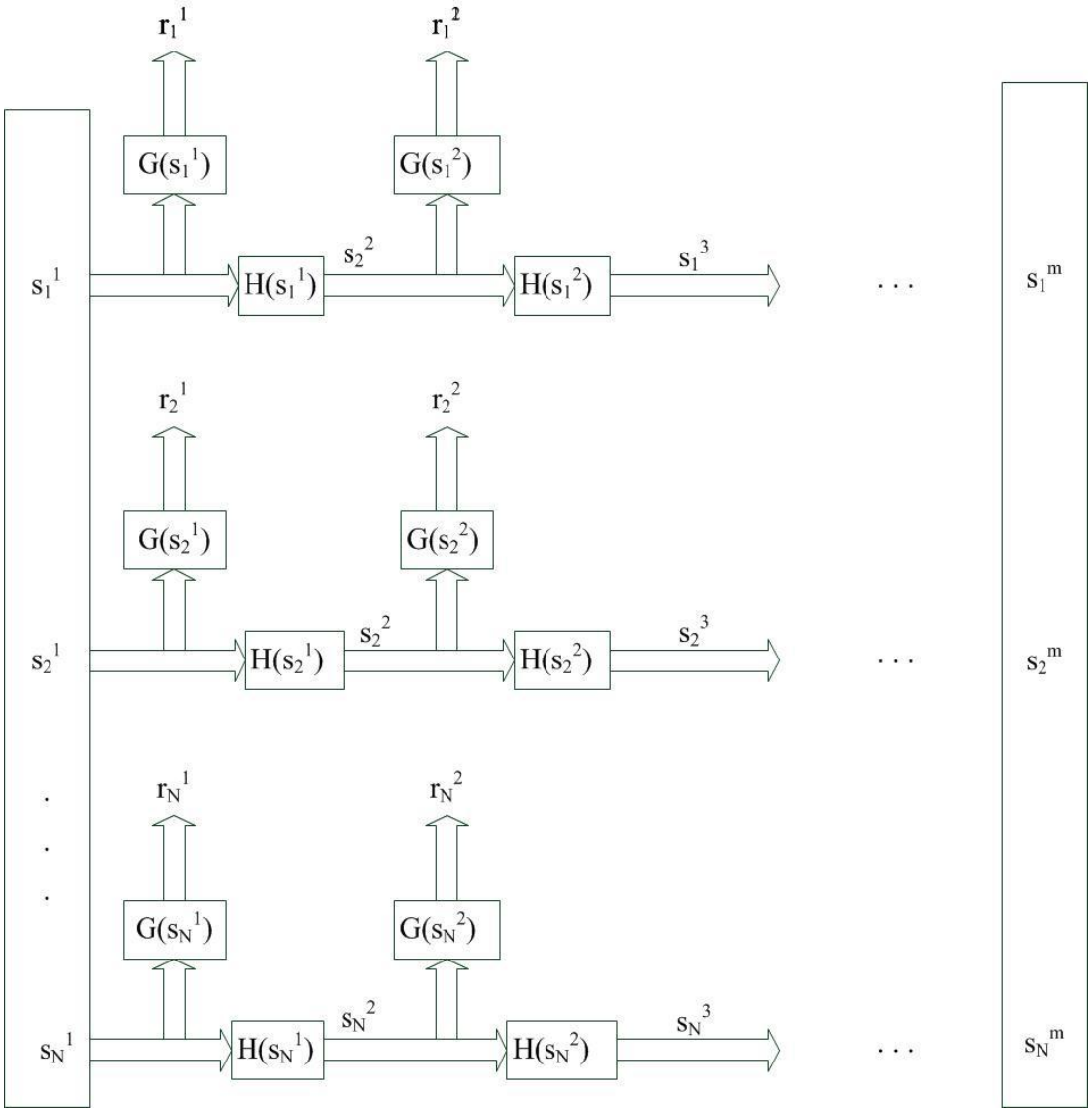
Şekil 2.1 OSKP'de okuyucu ve etiket arasındaki iletişim.

s_i^k ifadesindeki i indeksi kimlik bilgisinin sunucudaki veri tabanındaki numarasını vermektedir ve veri tabanı uzunluğu (N) ile sınırlıdır, yani $1 \leq i \leq N$'dir. Bu değer her etiket için özeldir ve güncellemeler sırasında değişmez. Aynı ifadedeki k değeri ise etiketin ömrüyle (m) ilgilidir, başlangıçta 1 olan bu değer okuyucudan gelen her istekle beraber yenilenir ve artar. Bu k değeri m'yi geçtiğinde ise artık o etiket içindeki kimlik bilgisinin ömrü dolmuştur, yani etiket geçerliliğini yitirmiştir, dolayısıyla $1 \leq k \leq m$'dir.

2.3.3. Ohkubu, Suzuki ve Kinoshita'nın Protokolünde Veri Tabanı Oluşturulması ve Kimlik Doğrulama İşlemi

Bir önceki alt bölümde anlatıldığı gibi her etikette bir kimlik bilgisi saklıdır ve okuyucudan gelen her istekle bu bilgi yenilenir. Veri tabanı uzunluğu ve kimlik bilgisinin ömrü göze alındığında doğrulama işlemi için veri tabanında $m \; x \; N$ tane kimlik bilgisi yer alması gerekmektedir.

Bu protokolde doğrulama işleminin hızını artırmak için hafıza kullanımı artırılarak bu kimlik bilgilerinin hepsi aynı anda veri tabanında saklanır. Bunun için etikette kullanılan H ve G fonksiyonlarının aynıları kullanılır. Şekil 2.2'deki blok diyagramda bu işlemin nasıl gerçekleştiği net bir şekilde anlatılmaktadır.



Şekil 2.2 OSKP'de veri tabanı oluşturulması.

Şekil 2.2'de görülen ilk sütundaki bilgiler etiketlerdeki bilgilerin başlangıç halleridir. Veri tabanı oluşturulurken önce başlangıç değerleri H ve G fonksiyonlarından ayrı ayrı geçirilir. G fonksiyonu çıkışında oluşan r_i^1 değerleri saklanır. Burada H fonksiyonun kullanılmasının amacı etiketlerdeki yenileme işlemini karşılamak içindir ve elde edilen s_i^2 bilgileri s_i^3 ile r_i^2 bilgilerini elde etmek için kullanılır. Bu işlem bu şekilde m kadar, yani r_i^m'ler elde edilene kadar devam eder ve bu süre boyunca bütün r_i^k'ler saklanır. Sonuç olarak r, b bitlik olmak üzere *m x N x b* büyüklüğündeki veri tabanı elde edilmiş olunur.

Sunucu okuyucudan bir kimlik bilgisi aldığında sahip olduğu veri tabanında bu bilgiyi arar. Bulduğu durumda kimlik doğrulaması yapılır ve etiketin veri tabanındaki yeri tespit edilir, aksi halde doğrulama yapılmaz. Burada dikkat edilmesi gereken

nokta doğrulamanın olmaması için veri tabanında olmayan bir kimlik bilgisinin gönderilmesinin yanı sıra etiketin ömür süresinin dolmasıyla alakalı da olabilir. Ömrü dolan kimlik bilgileri etikette ve veri tabanında yenilenene kadar doğrulama için kullanılamaz [3,4].

2.3.4. Ohkubu, Suzuki ve Kinoshita'nın Protokolünde Güvenlik

Bu protokolde öncelikle kullanılan özet fonksiyonları ile etiketin okuyucuya gönderdiği bilginin sabit olması engellenmiştir ve bu bilginin rastgele bir değere sahip olması sağlanmıştır [1,4]. Böylece saldırgan, okuyucu ile etiket arasındaki iletişimi dinleyerek ya da etikete istek göndererek etiket çıkışını elde etse de etiket içindeki kimlik bilgisine ulaşamaz.

Ayrıca etiket çıkışlarının rastgele olmasıyla kişilerin takip edilebilmesi engellenmiştir. Sağlanan bir başka güvenlik ise etiket çıkışı çalınsa bile kişinin geçmiş hareketleri hakkında bilgi edinilemez. Kişilerin ne zaman, ne yaptıkları sadece sistem tarafından bilinebilir, ancak kimi kişiler için bu da bir mahremiyet problemi olarak kabul edilebilir.

Ayrıca bu protokolde iki ayrı özet fonksiyonunun kullanılmasının sebebi ise saldırgan tarafından etiket kurcalanıp içindeki bilgi elde edilse de doğrulama için gerekli olan G fonksiyonu bilinmediği için okuyucuya doğru bilgi gönderilememesidir.

Bu protokolde mevcut olan problem ise saldırganın etiket çıkışını çalarak daha sonra bu bilgiyi kullanabilmesidir. Çünkü veri tabanında kimlik bilgisinin tüm halleri saklıdır ve doğrulama için her seferinde hepsiyle karşılaştırılır, yani bir etiket çıkışı etiketin kendisi tarafından olmasa da bir saldırgan tarafından çalınarak defalarca kullanılabilir.

2.4. Ohkubu, Suzuki ve Kinoshita'nın Protokolünün Geliştirilmesi

Gildas Avoine ve Philippe Oechslin tarafından zaman ve bellek arasındaki orta yolu bulma tekniği kullanılarak OSKP'nin sunucu yapısı geliştirilmiştir [3,4]. Geliştirilen bu yeni sunucu yapısıyla veri tabanındaki bellek ihtiyacı azaltılmaya çalışılmıştır ve bu sırada etiket ile okuyucu arasındaki iletişim tekniği değiştirilmediği için güvenlik aynı şekilde kalmıştır.

2.4.1. Gildas Avoine ve Philippe Oechslin Tarafından Geliştirilen Protokolde Veri Tabanı Oluşturulması

OSKP'nin sunucu yapısının Gildas Avoine ve Philippe Oechslin tarafından geliştirilmesi ile oluşan OSKP'nin gelişmiş hali (OSK/AOP) olan bu protokoldeki yeni sunucuda veri tabanı oluşturulurken F ve R fonksiyonları kullanılarak bir zincir oluşturulmuştur [3,4]. Tek yönlü olan F fonksiyonu şu şekilde tanımlanmıştır:

$$\mathrm{F}: (\mathrm{i}\ ;\ \mathrm{k}) \rightarrow \mathrm{r}_i^k = \mathrm{G}(H^{k-1}(s_i^1)) \tag{2.1}$$

Burada gene N veri tabanı uzunluğu, m etiket ömrü olmak üzere $1 \leq i \leq N$ ve $1 \leq k \leq m$'dir. Ayrıca s_i^1'ler etiketler içindeki gizli kimlik bilgileridir ve bunlar aynı zamanda veri tabanında da yer alırlar. Rastgele çıkış üreten ve bir indirgeme fonksiyonu olan R ise şu şekilde tanımlanmıştır:

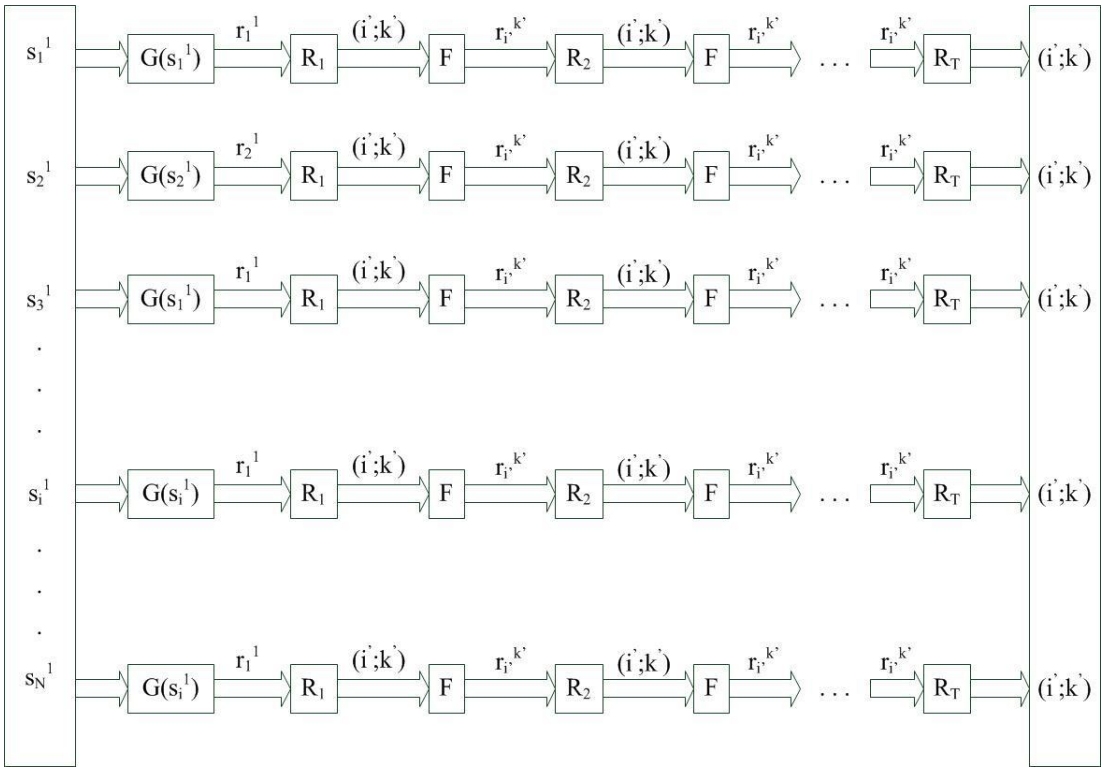
$$\mathrm{R}: r_i^k \rightarrow (i'\ ;\ \mathrm{k}') \tag{2.2}$$

Burada aynı şekilde $1 \leq i' \leq N$ ve $1 \leq k' \leq m$'dir. R fonksiyonuna örnek olarak

$$\mathrm{R(r)} = \left(1 + (\mathrm{r} \bmod \mathrm{n})\ ;\ 1 + \left(\left\lfloor \frac{r}{n} \right\rfloor \bmod \mathrm{m}\right)\right) \tag{2.3}$$

verilebilir. Görüldüğü gibi F fonksiyonunun çıkışı R fonksiyonunun girişi, aynı şekilde R fonksiyonunun çıkışı F fonksiyonunun girişidir ve bu şekilde bir zincir oluşturulmuştur.

Veri tabanı oluşturmak için öncelikle uygun bir zincir uzunluğu (T) seçilir ve bu T kadar veri tabanının içindeki kimlik bilgileri (s_i^1'ler) G fonksiyonundan geçirildikten sonra (çünkü OSKP'deki gibi etiket, içindeki bilgiyi G'den geçirip okuyucuya gönderir) sırasıyla R ve F fonksiyonundan geçer. Sonuç olarak en son (i';k') değerleriyle dolu bir sutün elde edilir ve OSK/AOP'de sadece ilk değerler olan etiketlerin kimlik bilgileri ile bu son sütun saklanır. Böylece kullanılan bellek oldukça azalmış olur. Veri tabanının nasıl oluştuğu Şekil 2.3'de daha net olarak anlatılmıştır:



Şekil 2.3 OSK/AOP'de veri tabanı oluşturulması.

Şekil 2.3'de görüldüğü gibi zincirin her sütununda farklı bir R fonksiyonu kullanılmıştır [3]. Böylece zincir boyunca bir çakışma olması, yani R ve F fonksiyon ikilisinin çıkışının sürekli aynı olması, engellenmiş olunur. Ancak iki satır aynı poziyonda aynı sonucu üretirse bu kimlik bilgilerinin son sütundaki değerleri aynı olabilir. Bu durumda veri tabanı oluşturulduktan sonra bu çakışmalar tespit edilip ilk seferden sonra aynı son değere ulaşan kimlik bilgileri veri tabanından silinmelidir ve yerine yeni kimlik bilgisi eklenip veri tabanı tekrar oluşturulmalıdır.

Ayrıca Denklem (2.1)'de görüldüğü gibi OSK/AOP'de her adımda OSKP'ye göre çok daha fazla özet hesaplaması yapılmaktadır, çünkü i ve k değerleri R fonksiyonundan üretilen rastgele değerlerdir ve bir önceki hesaplamalarla bir ilişkisi yoktur [4]. OSKP'de her adımda birer H ve G işlemi yapılmasına karşın OSK/AOP'de ortalama m/2+1 kadar özet hesaplaması yapılır. Bu yüzden her iki sunucu yapısında da aynı veri tabanı uzunluğu ve aynı etiket ömrü seçildiğinde OSK/AOP'de veri tabanı oluşturulması çok daha uzun sürer. Bu sebepten dolayı OSK/AOP'de veri tabanı güncellenmesi genellikle sistemin çevrim dışı olduğu durumlarda kullanıma göre istenilen periyotlarda yapılır.

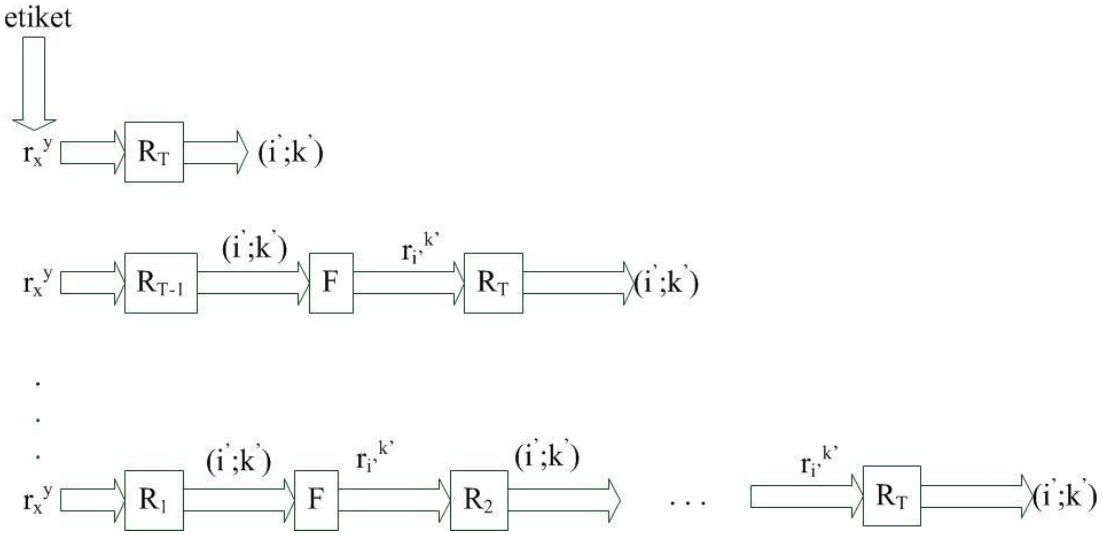
Veri tabanı güncellemesi sisteme yeni etiket eklendiği zaman ya da kullanılan bir etiketin kullanımı dolduğu zaman onu silmek için yapılır. Ancak normal güncellemelerde veri tabanına ekstra etiket bilgileri eklenerek her yeni etiket eklenişinde veri tabanının yeniden hesaplanmasından kurtulabilinir. Bahsedilen normal güncellemeler ise periyodik yapılan ve periyodu genellikle etiketlerin kullanım sıklığına ve ömür uzunluğuna bağlı olan yenilemelerdir [4].

OSK/AOP'de sunulan bir başka öneri ise m'yi küçültmektir [4]. Bunun için veri tabanında her s_i^1'in kaç defa geldiğini hesaplayabiliriz ve yeni veri tabanı oluşturulacağı zaman s_i^1 ile s_i^k (etiketin k defa okunduğu varsayılıyor) yer değiştirilir. Böylece artık m ömür süresi değil, bir etiketin iki veri tabanı oluşturulması arasındaki okunma sayısıdır ve bu işlem protokolün daha hızlı olmasını sağlar.

Son olarak bellek kullanımından bahsedilmelidir. Örneğin $N=2^{20}$, $m=2^7$ ve kimlik bilgilerini ve özet fonksiyonu çıkışları 160 bitlik alınırsa bilindiği üzere OSKP'de veri tabanı için gerekli olan bellek miktarı 2^{20} x 2^7 x 160 bit kadar olur. OSK/AOP'de ise sadece iki sütun saklanır, aynı değerler alındığında bunlardan ilki 2^{20} x 160 bit büyüklüğünde ve diğeri ise seçilen N ve m değerleriyle i 20 bit, k 7 bit olmak üzere 2^{20} x 27 bit büyüklüğündedir. Bu örnekte gerekli bellek miktarları oranlandığında OSK/AOP'de bellek miktarının yaklaşık 235 kat daha az olduğu görülür.

2.4.2. Gildas Avoine ve Philippe Oechslin Tarafından Geliştirilen Protokolde Kimlik Doğrulama

OSKP'de olduğu gibi OSK/AOP'de de etiket ile okuyucu arasındaki ilişki aynıdır ve sunucuya gelen kimlik bilgisi en son G özet fonksiyonundan geçmiş haldedir. Gelen kimlik bilgisinin (r_x^y) doğrulama işlemi son sütundaki (i';k') ikilileri ile karşılaştırılarak yapılır [3,4]. Ancak bu karşılaştırmanın yapılabilmesi için gelen bilginin de sunucuda bir R fonksiyonundan geçirilmesi gerekmektedir. Bu işlem Şekil 2.4'deki gibi yapılmaktadır.

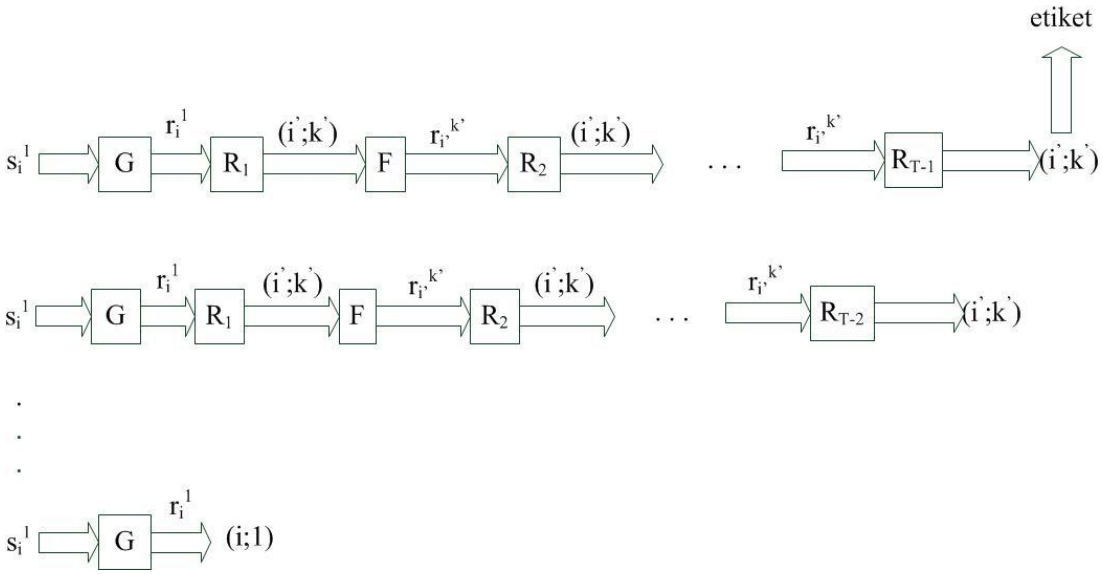


Şekil 2.4 OSK/AOP'de arama işlemi.

Şekil 2.4'te görüldüğü gibi etiketten gelen r_x^y değeri öncelikle R_T'den geçirilip (i';k') ikilileri ile karşılaştırılır. İkililerin tüm zincir içindeki sütunlardaki dağılımları rastgele olduğu için doğrulama yaparken gelen r_x^y'in son sütundan önceki r_i^k'lardan biri olduğunu varsayarak aramak en hızlısıdır. Bu yüzden gelen etiket bilgisi sunucuda R_T fonksiyonundan geçirilerek elde edilen (i';k') ikilileri son sütundaki (i';k') ikilileri ile karşılaştırılır. Bulunamaması durumunda r_x^y'in bir önceki r_i^k'lardan biri olduğunu varsayılarak r_x^y sunucuda sırasıyla R_{T-1}, F, R_T fonksiyonlarından geçirilip tekrar son sütundaki (i';k') ikilileri ile karşılaştırılır. Bu şekilde sırayla zincirin en başına kadar gidilir ve tüm zincir boyunca gelen etiket bilgisi bulunamazsa doğrulama yapılamaz.

Sırayla yapılan bu karşılaştırmalar sırasında (i';k')'lar aynı ise doğrulama yapılır. Doğrulamanın tamamlanması için öncelikle bu gelen etiketin veri tabanında hangisi olduğu bulunmalıdır, çünkü bilindiği üzere veri tabanındaki bu tabloda bilgiler rastgele yer almaktadır. Doğrulanacak etiketin hangisi olduğunu bulmak için öncelikle ikililerin hangi satırda aynı olduğu bulunur, daha sonra veri tabanında bu satırın en başındaki sütunda yer alan kimlik bilgisinden (s_i^1'den) başlanarak arama yapılırken karşılaştırılan ikililerin aynı olduğu zaman hangi R fonksiyonundan başlandıysa, bu sefer ondan bir önceki R fonksiyonuna kadar veri tabanı oluşturulurken ki gibi sırayla fonksiyonlardan geçirilir ve bir (i';k') ikilisi elde edilir. Burada i' arama yapılırken etiketten gelen bilginin indisidir, yani veri tabanında hangi kimlik bilgisi olduğunu verir ve k' ise etiketin ömrüyle ilgili bilgi verir. Örneğin gelen etiket bilgisinin R_{T-3}'ten önceki sütunda olduğu varsayılarak arama

yapıldığı ve bulunduğu zaman bu etiketin hangisi olduğunu bulmak için aynı satırdaki en baştaki s_i^1 bilgisi G özet fonksiyonundan başlanarak sırayla R ve F fonksiyonlarından R_{T-4}'e kadar geçirilir ve R_{T-4}'ten sonra elde edilen i' onun veri tabanındaki yerini verir. Şekil 2.5'te bu işlemin nasıl yapıldığı anlatılmıştır ve bu şekildeki yukarıdan aşağıya olan sıranın şekil 2.4'teki sıraya karşılık olduğuna dikkat edilmelidir.



Şekil 2.5 OSK/AOP'de etiket bilgisinin veri tabanındaki yerini bulma.

OSK/AOP'de gelen bir bilginin doğrulanamamasının sebebi yalnızca kimlik bilgisinin veri tabanında yer almaması ya da etiketin ömrünün dolması değildir [4]. Bir başka sebep ise OSK/AOP'de kullanılan zaman ve bellek arasındaki orta yolu bulma tekniğinin olasılığa dayanan bir yöntem olmasından kaynaklanır. Her ne kadar yeterli bir zincir uzunluğu seçildiğinde F fonksiyonunun çıkışlarının zincirin adımlarında en az bir kere yer alması gerekse de küçük bir ihtimal de olsa bir s_i^k'nin oluşturulan zincirin herhangi bir adımında yer almaması söz konusu olabilir. Çünkü veri tabanında oluşan tablonun büyüklüğü *T x N* kadardır ve bu tüm r_i^k'lerin yer alabileceği tablo büyüklüğü olan *m x N*'den küçüktür. Bundan dolayı ilk sütunda var olan bir etiket sistem tarafından tanımlanamayabilir. Bunun için önerilen yöntemlerden biri okuyucunun etikete tekrar istek göndererek bu sefer s_i^{k+1}'i sorgulaması, çünkü her iki bilginin de yer almama ihtimali oldukça düşüktür. Önerilen daha gerçekçi bir yöntem ise veri tabanına yeni satırlar eklemektir. Ancak bu satırların ilk sütundaki değerleri etiketlere kimlik bilgisi olarak verilmez, bunlar sadece tabloda var olan r_i^k'leri artırmak içindir. Böylece tüm r_i^k'lerin veri tabanında

oluşturulan tabloda bulunma ihtimali artar. Çünkü veri tabanı büyüklüğü $T\ x\ (N+N_{ek})$ olur ve bu $m\ x\ N$'ye daha yakındır.

Ayrıca OSK/AOP'de doğrulamada ortaya çıkabilecek başka bir problem ise veri tabanında var olmayan bir etiket kimlik bilgisinin doğrulanabilmesidir. Bunun sebebi R fonksiyonlarından kaynaklanmaktadır. Çünkü R fonksiyonu Denklem (2.3)'teki gibi seçildiğinde farklı girişlerin R fonksiyonu çıkışı yapılan mod işleminden dolayı aynı olabilir. Örneğin 14 ve 24 sayıları farklı olsalar da mod 10 değerleri aynıdır.

3. GRAFİK İŞLEMCİ BİRİMİ ÜZERİNDE PROGRAMLAMA

3.1. Tümleşik Birim Cihaz Mimarisi

Bu çalışmada gerçekleme için GİB üzerinde programlama yapılmıştır. Bunun için NVIDIA tarafından ilk defa Kasım 2006 tarihinde tanıtılan TBCM seçilmiştir [6]. Bu mimari NVIDIA tarafından son yıllarda üretilen birçok grafik kartı tarafından desteklenmektedir. TBCM ile GİB, MİB'ye yardımcı bir işlemci olarak iş görebilmekte, kendine ait bir bellek alanına sahip olmakta ve birçok ipliği (thread) paralel olarak yürütebilmektedir [7]. Böylece TBCM ile GİB'nin çoklu çekirdek yapısı üzerinde paralel programlama yapılabilmektedir ve programlama için kullanımı oldukça yaygın ve basit olan C dili TBCM tarafından getirilen eklentiler ile (TBCM C) çok etkin bir şekilde kullanılabilmektedir [6]. Başka bir deyişle karmaşık işlemleri hızlı bir şekilde gerçeklemek için çok üst seviyede dilleri bilmeye gerek kalmamaktadır.

GİB üzerinde programlama yapıldığında çekirdek sayısı artıkça bir programın çalışması daha kısa sürmektedir. Günümüzde grafik kartlarının yüzlerce çekirdeğe sahip olabileceği ve paralel iplik sayısının binlerce olabileceği göz önüne alındığında MİB yerine GİB'nin kullanılmasıyla paralel programlama yaparak gerçekleştirilmesi karmaşık ve yavaş olan birçok işlem çok kolay bir şekilde gerçekleştirilebilmektedir.

3.2. Tümleşik Birim Cihaz Mimarisi C

Bir TBCM programı paralelliğin olmadığı ya da çok az olduğu MİB kodu ve çok fazla paralelliğin olabileceği GİB kodu olmak üzere temel olarak iki bölümden oluşur [2]. Burada GİB üzerinde çalışan kod MİB kodu tarafından çağrılan Grafik İşlemci Birimi Fonksiyonu (GİBF) (kernel) ile yazılır.

3.2.1. Grafik İşlemci Birimi Fonksiyonları

TBCM C'nin C'ye getirdiği en büyük eklenti MİB üzerinde yazılan bir kodla çağrılan, GİB üzerinde çalışan bir fonksiyon yazılabilmesidir [6]. Şekil 3.1'de iki vektörü toplayan bir GİBF'ye ve GİBF'nin çağırılmasına örnek olarak bir kod verilmiştir.

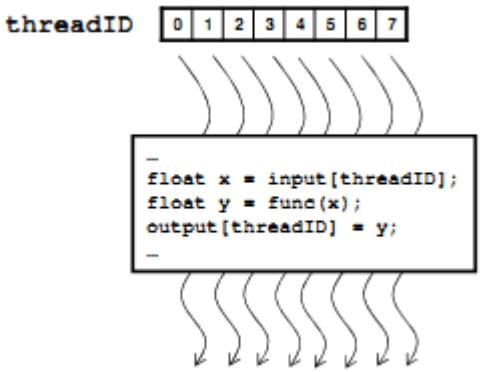
```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
}
```

Şekil 3.1 Bir GİBF'ye ve MİB üzerinden çağrılmasına örnek [6].

Görüldüğü gibi GİBF tanımlanırken __global__ bildirim belirteci kullanılmıştır. Ayrıca görüldüğü gibi MİB kodu içinde GİBF çağrılırken GİB üzerinde kaç tane paralel bloğun (block) kullanılacağı ve her blokta kaç paralel ipliğin kullanılacağı <<<...>>> sentaksı içinde belirtilir. Şekil 3.1'de 1 tane blok ve N tane iplik kullanılmıştır. Bir GİBF'yi yürüten her ipliğin blok içinde kendine özel bir kimlik numarası vardır, bunlar GİBF'nin içinde tanımlıdır ve threadIdx değişkeni ile gösterilir [6].

Bir GİBF __global__ belirteci ile tanımlandıysa yalnızca MİB kodu tarafından çağrılabilir, yani başka bir GİBF ile çağrılamaz ve dönüş tipi her zaman void olmalıdır. Ancak __device__ belirteci ile yazılan GİBF'ler ise yalnızca başka GİBF'ler tarafından, yani GİBF kodu tarafından çağrılabilir ve dönüş tipi void olmak zorunda değildir [2].

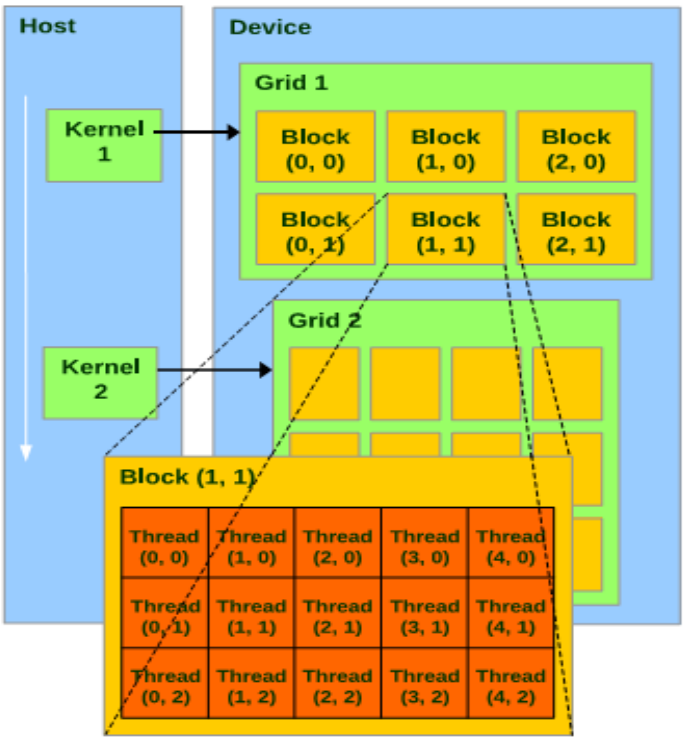
Ayrıca, aynı anda ancak bir GİBF çalışabilir ancak birçok iplik aynı anda bu GİBF'yi yürütebilir [7] (bkz. Şekil 3.2).



Şekil 3.2 Aynı anda birçok ipliğin bir GİBF'yi yürütmesi [7].

3.2.2. İplikler ile İlgili Bilgiler

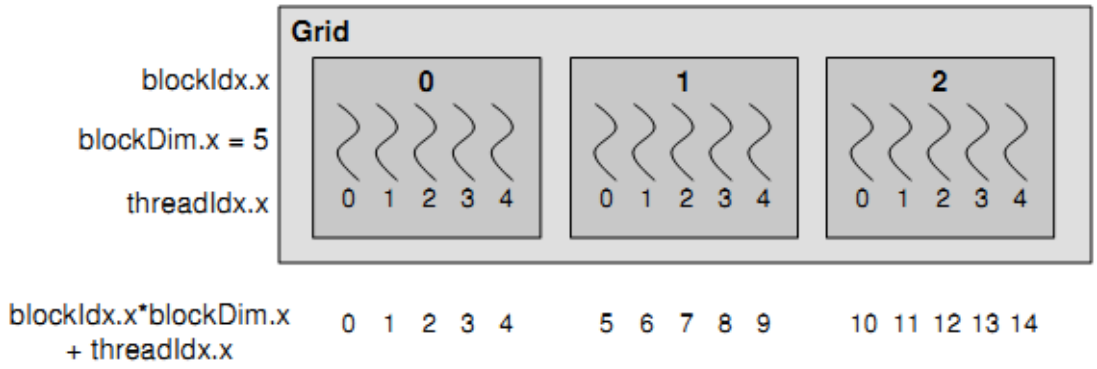
İplikler paralelliğin birimi olarak tanımlanır ve MİB ipliklerinden farklı olarak yaratılması daha kolaydır, kaynak kullanımı daha azdır [7]. Bir GİBF'yi yürüten bu iplikler blok denilen yapılar içinde yer alır. Bu bloklar kodun yazıldığı amaca bağlı olarak bir boyutlu, iki boyutlu ya da üç boyutlu olabilmektedir. Bir GİBF boyunca çalışan bloklar ise ızgara (grid) denilen yapı içinde yer alır ve bu yapı bir boyutlu ya da iki boyutlu olabilmektedir. Yani bir GİBF içinde birçok iplik bloğu yer alan ızgara tarafından yürütülür [2] (bkz. Şekil 3.3).



Şekil 3.3 Izgara, blok ve iplik yapısı [8].

Şekil 3.3'te görüldüğü gibi her bloğun ve ipliğin bir dizin numarası vardır. Ayrıca bir önceki alt başlıkta bahsedildiği gibi bir GİBF'yi yürüten her ipliğin bir kimlik numarası vardır. Bu ikisi arasında direk bir ilişki vardır. Tek boyutlu blok için indeks numarası ile kimlik numarası aynıdır. (D_x, D_y) büyüklüğünde iki boyutlu blok için indeks numarası (x, y) olan bir ipliğin kimlik numarası $x + yD_x$ olur. (D_x, D_y, D_z) büyüklüğünde üç boyutlu blok için indeks numarası (x, y, z) olan bir ipliğin kimlik numarası $x + yD_x + zD_xD_y$ olur [6].

GİBF kodu yazımı sırasında ipliklerin kontrolü için kimlik numaraları çok önemlidir. Az önce bahsedilen indeks numaraları GİBF içinde tanımlı olan threadId.x, threadId.y ve threadId.z değişkenleri ile ifade edilir. Blokların ızgara içindeki indeks numaraları ise GİBF içinde tanımlı olan blockId.x, blockId.y değişkenleri ile ifade edilir. Blok ve ızgara boyutlarını ise aynı şekilde GİBF içinde tanımlı olan blockDim.x, blockDim.y, blokDim.z, gridDim.x ve gridDim.y değişkenleri ifade eder [9]. Şekil 3.4'te tek boyutlu bir blok ve ızgara için iplik kimlik numarasının kod içinde nasıl belirlendiği gösterilmiştir.

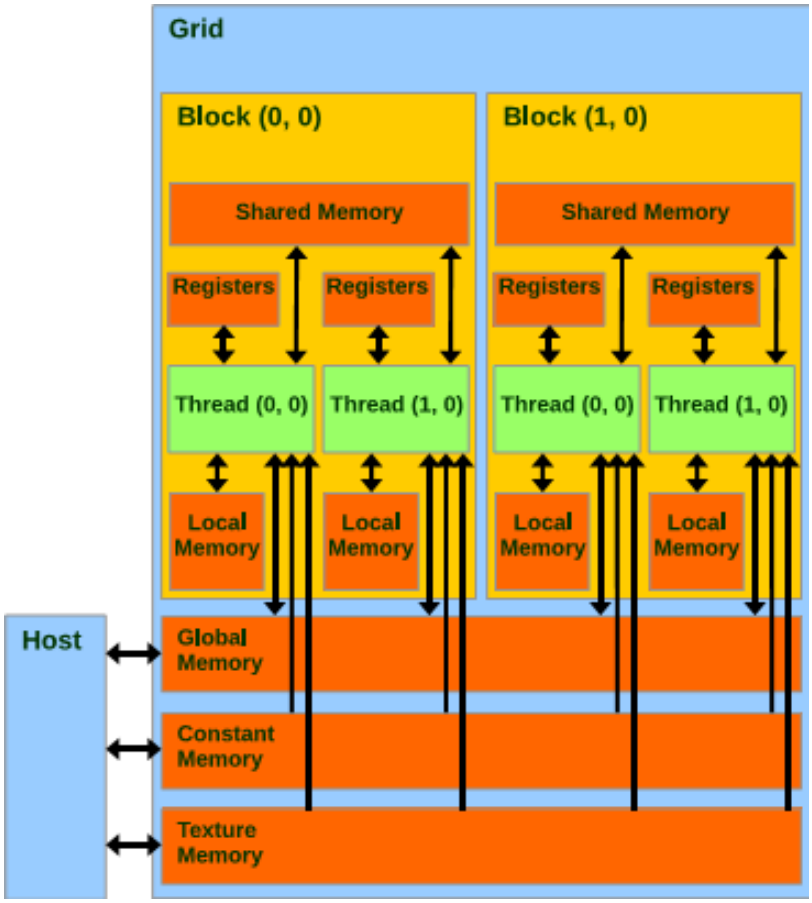


Şekil 3.4 Tek boyutlu blok ve ızgara için iplik kimlik numaraları [7].

3.2.3. Bellekler ve Bellek Yönetimi

TBCM'de MİB ile GİB farklı bellek alanlarına sahiptir [7]. GİB'de her ipliğin kendine ait 32 bitlik kütükleri vardır ve her iplik sadece kendi kütüğüne erişebilir . Ayrıca her blokta sadece blok içindeki ipliklerin erişip veri paylaşabildiği paylaşımlı bellek (shared memory) vardır ve paylaşımlı belleğin ömrü bloğun ömrü kadardır, yani GİBF süresi kadardır. Bu tipteki belleklerde tanımlı olan değişkenlere erişim çok hızlıdır. MİB ile GİB'nin ortak kullandığı bellekler ise global, değişmez ve doku bellekleridir. MİB kodu ile GİB kodunun her ikisi de global belleğe veri yazabilir ya

da bellekten veri okuyabilir, ancak değişmez bellek (constant memory) ile doku belleğe (texture memory) sadece MİB iki yönlü ulaşabilir, GİB kodu ise bu belleklerden sadece veri okuması yapabilir. Şekil 3.5'te tüm bu bellek türleri gösterilmiştir.



Şekil 3.5 Bellek yapısı [8].

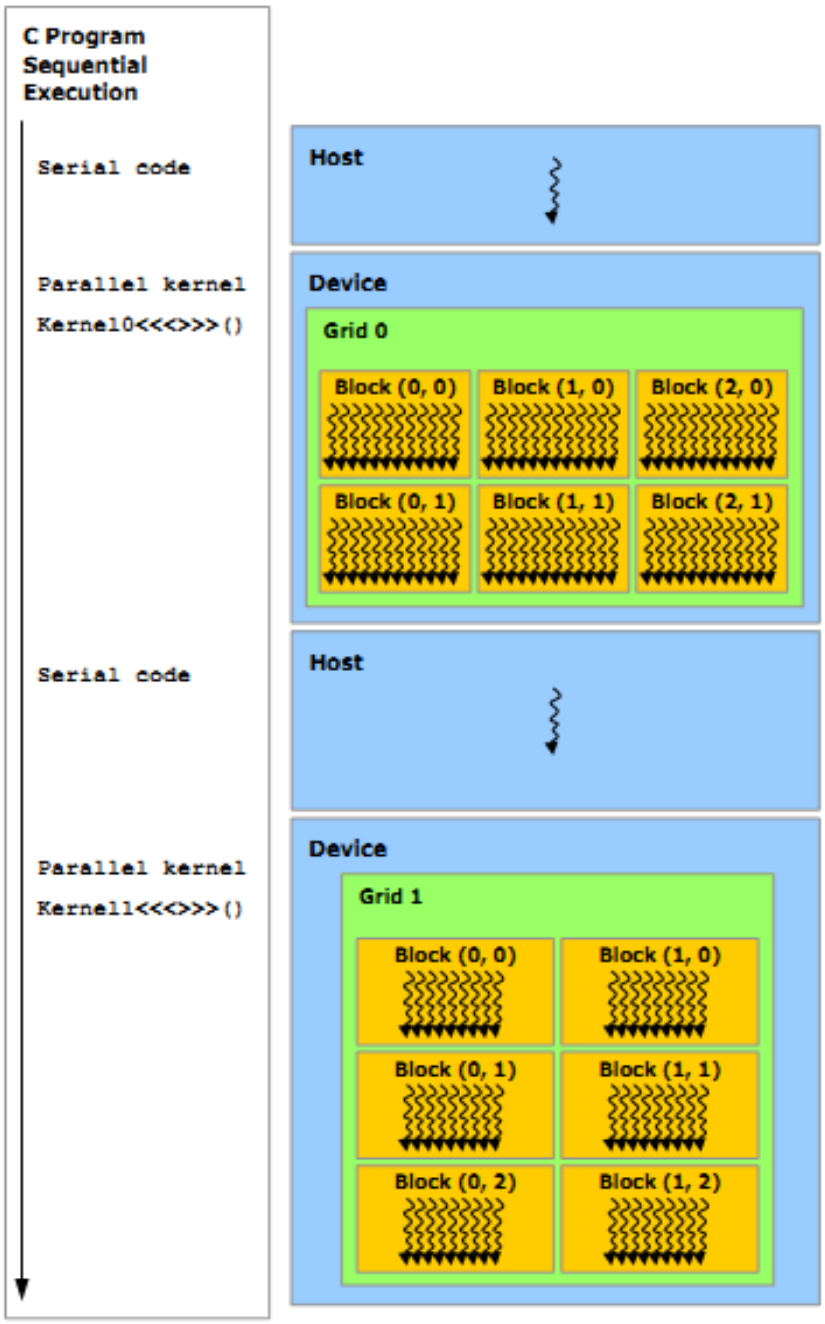
GİB üzerinde çalışacak bir GİBF yazılacağı zaman öncelikle GİB belleği üzerinde yer ayrılmalıdır ve işlem bittikten sonra bu bellek alanları boşaltılmalıdır [9]. Bunun için kullanılan fonksiyonlar cudaMalloc(.) ile cudaFree(.) fonksiyonlarıdır. Daha sonra GİB belleği üzerinde ayrılan bu bellek alanına MİB belleğinden veri aktarılmalıdır ve GİBF bittikten sonra bu sefer GİB belleğinden MİB belleğine veri aktarılmalıdır. Bunun için kullanılan fonksiyon ise cudaMemcpy(.) fonksiyonudur ve ayrıca cudaMemcpy(.) fonksiyonu ile MİB'den MİB'ye ya da GİB'den GİB'ne de veri transferi mümkündür. Tüm bu TBCM C fonksiyonlarının C'de kullanılan malloc(.), free(.) ve memcpy(.) fonksiyonlarına benzediklerine ve bunlara ek olarak geliştirildiklerine dikkat edilmelidir.

Son olarak paylaşımlı bellekten biraz daha bahsetmekte fayda vardır [9]. Paylaşımlı bellek yonga üzerinde yer alır, global belleğe göre ulaşılması daha hızlıdır ve bant genişliği çok yüksektir. En önemlisi ise her bloğun kendine ait bir paylaşımlı belleği vardır ve blok içindeki iplikler bu bellek üzerinden iletişim kurabilirler ancak bir bloktaki iplikler başka bloktaki ipliklerle iletişim kuramazlar. Dolayısıyla GİB üzerinde kod yazılırken paylaşımlı bellekten yararlanmak verimi oldukça artıracaktır. Kod içinde bir değişkenin paylaşımlı bellek içinde yer alacağını belirtmek için ise __shared__ belirteci kullanılır. Paylaşımlı bellekte önemli olan başka bir nokta ise ipliklerin senkronize edilmesidir. Çünkü bu bellek ile iplikler veri alışverişi yapabilmektedir ve eğer iplikler senkronize edilmezse veri alışverişi doğru bir şekilde gerçekleşmeyebilir. Örneğin bir ipliğin paylaşımlı belleğe bir değer yazdığını ve başka bir iplik bu değeri alması gerektiğini düşünelim. İkinci ipliğin doğru değeri alabilmesi için ilk iplik değeri yazdıktan sonra bu bellek alanına erişmelidir ancak bilindiği gibi tüm iplikler paralel çalışır ve bu veri alışverişin doğru çalışması iplikler senkronize edilmediyse tamamen şansa bağlıdır. Bunun için TBCM C'de __syncthreads(.) fonksiyonu yer almaktadır. Bu fonksiyon çağrıldığında blok içindeki tüm iplikler birbirini bekler ve en geç iplik kod içinde bu noktaya ulaştıktan sonra tüm iplikler devam eder.

3.2.4. Heterojen Programlama

Daha önce de bahsedildiği gibi bir TBCM C programı paralelliğin olmadığı ya da çok az olduğu MİB kodu ve paralelliğin çokça olduğu, MİB'ye yardımcı işlemci olarak çalışan ayrı fiziksel bir cihaz üzerinde çalışan kod (GİB kodu) olmak üzere iki temel bölümden oluşur [2]. Şekil 3.6'da bu çalışma şekli açık olarak anlatılmıştır.

Ayrıca bir program yazarken algoritmanın nerelerinde paralelliğin uygulanabileceğine dikkat edilmelidir. Çünkü paralel iplik yapısı her zaman olarak uygulanamayabilir ya da uygulansa bile etkin bir performans elde edilemeyebilir. Bu yüzden bir uygulama paralel gerçekleneceği zaman paralelliğin etkin bir şekilde uygulanıp uygulanamayacağı dikkatlice düşünülmelidir.



Şekil 3.6 Heterojen programlama [6].

3.2.5. Merkezi İşlem Birimi ile Grafik İşlemci Birimi Kodlarının Örnek Kod ile Karşılaştırılması

Bu alt bölümde iki vektörü toplayıp sonucu üçüncü bir vektöre yazan iki farklı kodun fonksiyonlarına örnekler verilecektir. Bunlardan ilki C'de yazılmıştır ve fonksiyon MİB üzerinde çalışır. Diğeri ise TBCM C ile yazılmış ve vektör toplama işlemi GİB üzerinde gerçekleşir. Şekil 3.7'de sırasıyla MİB üzerinde çalışan fonksiyon ve global belleği kullanan GİBF gösterilmiştir.

```
//MİB üzerinde çalışan fonksiyon
void vektoplama1(int *v1, int *v2, int *v3)
{
    //for döngüsü içinde iki vektörün toplanması
    for(int s=0;s<vektoruzunlugu;s++)
    v3[s]=v1[s]+v2[s];
}

//GİB üzerinde çalışan fonksiyon
__global__ void vektoplama2(int *v1, int *v2, int *v3)
{
    //Grid içinde thread kimlik numaralarının belirlenmesi
    int idx=threadIdx.x+blockDim.x*blockIdx.x;
    //Paralel toplama işlemi
    v3[idx]=v1[idx]+v2[idx];
}
```

Şekil 3.7 MİB üzerinde çalışan fonksiyon, global belleği kullanan GİBF.

Şekil 3.7'de görüldüğü gibi iki fonksiyon birbirine çok benzerdir. Ancak GİBF'de *for* döngüsü kullanılmamıştır. Çünkü aktifleştirilen tüm iplikler aynı anda bu fonksiyonu yürütür ve bu da kodda idx değişkenine ipliklerin kimlik numaralarının atanması ve vektör toplama işleminde idx değişkeninin dizilere endeks numarası olarak verilmesiyle sağlanmıştır.

Kodda vektoruzunlugu değişkeni 33280000 yapıldığında ve iplik sayısı 512, dolayısıyla blok sayısı da 65000 seçildiğinde bu iki ayrı fonksiyonun vektör toplama işlemini gerçekleştirme süreleri Tablo 3.1'de gösterilmiştir.

Tablo 3.1 MİB üzerinde çalışan fonksiyon, global belleği kullanan GİBF

	İşlemi gerçekleştirme süresi (msn)
MİB üzerinde çalışan fonksiyon	150
Global belleği kullanan GİBF	126.74

Tablo 3.1'de görüldüğü gibi GİB üzerinde çalışan vektör toplama fonksiyonu yaklaşık %15,5 daha hızlıdır. Vektör toplama işlemi gibi basit bir işlemde gözüken bu fark daha karmaşık işlemlerde daha da artabilmektedir. Özellikle TBCM C'nin paralel iplik yapısı ve GİB'nin kendine ait bellek yapısı ile çok karmaşık işlemler çok rahat ve hızlı bir şekilde çözülebilmektedir. Ancak burada yazılacak algoritmanın paralel programlama yapısına uygun olmasına dikkat edilmelidir.

4. GERÇEKLEME

4.1. Gerçekleme için Kullanılan Grafik Kartının Özellikleri

Bu çalışmada sunucuların gerçeklenmesi işlemi NVIDIA Geforce GTX 275 grafik kartı kullanılarak yapılmıştır. Tablo 4.1'de bu grafik kartının TBCM ile gerçekleme açısından önemli olan özellikleri verilmiştir.

Tablo 4.1 Kullanılan grafik kartının özellikleri

Cihazın ismi	Geforce GTX 275
TBCM sürücü sürümü	3.20
TBCM çalıştırma sürümü	3.20
Global belleğin toplam büyüklüğü	911605760 bayt
Toplam çoklu işlemci sayısı	30
Her çoklu işlemcideki ve toplam çekirdek sayısı	8, 240
Toplam değişmez bellek büyüklüğü	65536 bayt
Her bloktaki paylaşılan belleğin toplam büyüklüğü	16384 bayt
Her bloğun erişebildiği toplam kütük sayısı	16384
Her bloktaki maksimum iplik sayısı	512
Her bloğun maksimmum boyutları	512 x 512 x 64
Her ızgaranın maksimum boyutları	65535 x 65535 x 1
Saat hızı	1.40 GHz

Tablo 4.2'de ise grafik kartının bant genişliği özellikleri verilmiştir. Görüldüğü gibi transfer büyüklüklerinin hepsi aynıdır ancak en hızlısı GİB'den GİB'ye transferdir ve MİB'den GİB'ye ve GİB'den MİB'ye transfer hızı neredeyse aynıdır.

Tablo 4.2 Bant genişliği

	Transfer büyüklüğü (bayt)	Bant genişliği (Mb/sn)
MİB'den GİB'ye bant genişliği	33554432	2954
GİB'den MİB'ye bant genişliği	33554432	2999.8
GİB'den GİB'ye bant genişliği	33554432	104042.6

4.2. Gerçekleme için Kullanılan Özet Fonksiyonları

Bölüm 2'de anlatıldığı gibi gerçekleme boyunca özet fonksiyonu olarak GÖA-1 kullanılmıştır [10]. GÖA-1 2^{64} bit uzunluğundan küçük girişler için 160 bitlik bir çıkış üretir. GÖA-1 daha önce de bahsedilen her giriş için farklı çıkış üretme ve çıkışından yola çıkılarak girişini elde edememe özelliklerine sahip olacak şekilde tasarlanmıştır. Tablo 2.1'de çeşitli girişler için GÖA-1 özet fonksiyonunun çıkışlarına örnekler verilmiştir.

Gerçekleme boyunca iki farklı özet fonksiyonu kullanılmıştır (H ve G). Bunların ikisi için de GÖA-1 kullanılmıştır ancak aralarındaki farklılığı sağlamak için G fonksiyonu için GÖA-1'in girişine verilecek mesaj önce giriş ile aynı uzunlukta, sabit bir dizi ile özel veya işlemine sokulmuştur, daha sonra GÖA-1'e giriş olarak verilmiştir. Böylece H ve G arasında farklılık sağlanmıştır.

4.3. Yazılımında Paralelliğin Kullanıldığı Yerler

OSKP için yazılan programda veri tabanı oluşturulurken her ömür için kimlik bilgileri (r_i^k'lar) ayrı bir dizide saklanmıştır. Yani ömür süresi kadar dizi yaratılmıştır. Bu dizilerden her birinin oluşturulması kendi içinde paraleldir, ancak bu dizilerin hepsinin oluşturulması ise seridir. Yani sonuçta veri tabanında elde

edilecek her kimlik bilgisi dizisi elde edilirken paralellik kullanılır ancak bu dizilerin ayrı ayrı oluşturulması ise *for* döngüsü içinde seri gerçeklenir.

OSKP'de doğrulama işlemi sırasında ise her ömür için elde edilen dizi içinde arama paralel yapılır, ancak aramanın ilk ömür bilgilerinden son ömür bilgilerine kadar her dizide ayrı ayrı yapılması seridir, yani önce bir dizi içinde arama paralel yapılır sonra diğer diziye geçilir.

OSK/AOP için yazılan programda ise veri tabanı oluşturulmasında Şekil 2.3'te de görülen ilk ve son sütundaki bilgiler iki ayrı dizide saklanmıştır. Burada ilk sütundan son sütunun oluşturulması sırasında her kimlik bilgisinin işlemi bir iplik tarafından yapılmıştır, yani paralellik kullanılmıştır. Ancak her satırdaki kimlik bilgisinin işlemleri yapılırken her zincir adımı paralel gerçeklenmemiştir, yani her iplik zincirin adımlarını seri yürütmüştür. Başka bir deyişle son sütun oluşturulurken her satır birbirinden bağımsız, paralel ve kendi içinde seri yürütülmüştür.

Ayrıca OSK/AOP'de veri tabanının oluşturulmasının tamamlanması için iki ayrı fonksiyon daha yazılmıştır. Bunlar veri tabanını genişletmek ve son sütundaki çakışan bilgileri silmek içindir. Veri tabanının genişletilmesi için yazılan satır ekleme fonksiyonunun çalışması veri tabanının oluşturulmasıyla aynıdır. Çakışan bilgilerin silinmesinde bir bilginin diğer bilgilerle karşılaştırılması ve aynı olanların silinmesi işlemi paraleldir ancak her bilgi için bu işlem sırayla yapılmıştır. OSK/AOP'de doğrulama işleminde ise son sütunda arama işlemi paralel olarak yapılmıştır.

4.4. Analiz Sonuçları

Yapılan analizler için veri tabanındaki etiket sayısının, yani veri tabanının uzunluğunun (N) maksimum değeri 290 000, ömür süresinin (m) maksimum değeri 128 ve OSK/AOP için zincir uzunluğu (T) 54 alınmıştır. Bellek ve süre analizleri yapılmıştır.

4.4.1. Kullanılan Bellek Miktarları

İki ayrı sunucu yapısının gerçeklenmesinde kullanılan bellek alanları Tablo 4.3'te verilmiştir. Burada etiket kimlik bilgisi olan r'nin bit uzunluğunun (b) kullanılan GÖA-1 özet fonksiyonundan dolayı 160 bit (20 bayt) olduğundan bahsedilmelidir.

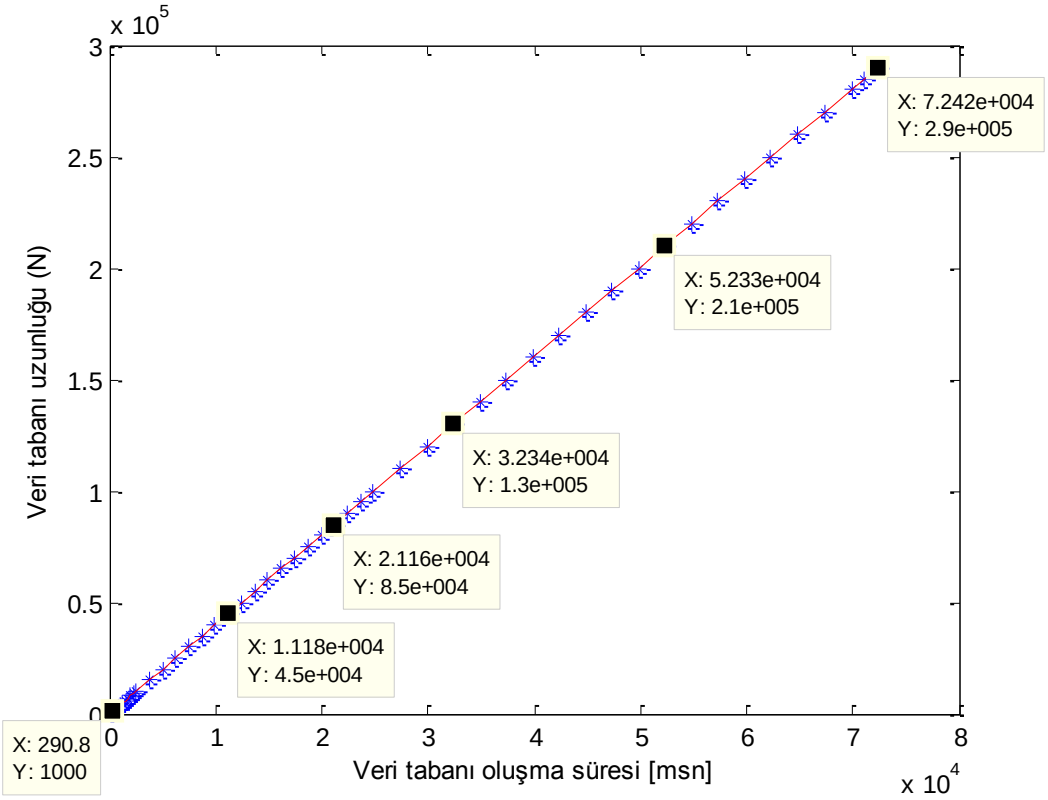
Tablo 4.3 Kullanılan bellek alanları

OSKP	N x m x b = 290000 x 128 x 20 = 742400000 bayt
OSK/AOP	2 x N x b = 290000 x 2 x 20 = 11600000 bayt

Beklenildiği gibi OSK/AOP'de kullanılan bellek alanı OSKP'de kullanılandan 64 kat daha azdır.

4.4.2. Süre Analizi

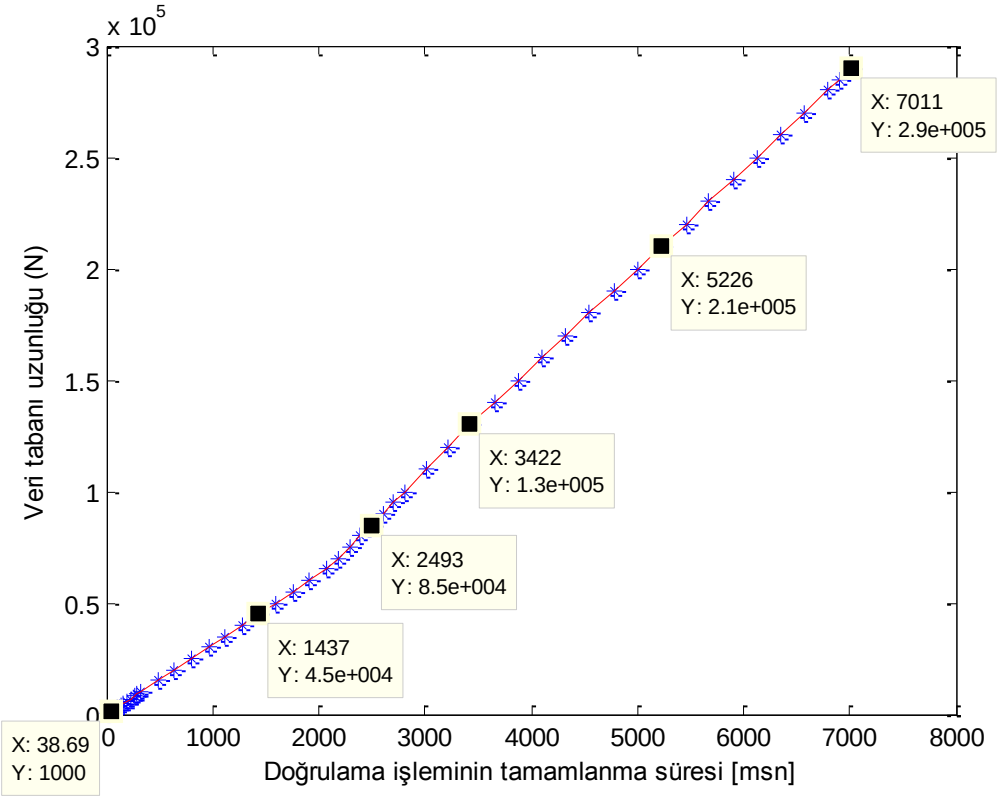
Her bir sunucu yapısı için veri tabanının oluşturulması ve doğrulama işleminin tamamlanması sürelerinin analizi yapılmıştır. Bu analizler m ve N'nin değişimine göre yapılmıştır. Öncelikle OSKP için m 128'de sabit tutulup N sayısı artırılmış ve maksimum N değeri belirlenmiştir. Maksimum N değerini GİB'nin kullanabildiği global belleğin maksimum değeri sınırlamaktadır ve OSKP için maksimum N değeri yaklaşık 290000 olarak bulunmuştur. Şekil 4.1'de m değeri 128 olduğunda artan N değerlerine göre veri tabanının oluşma süresinin değişim grafiği verilmiştir.



Şekil 4.1 OSKP için Veri tabanı oluşma süresi – Veri tabanı uzunluğu grafiği.

Şekil 4.1'de görüldüğü üzere N değeri arttıkça veri tabanı oluşma süresi de neredeyse doğrusal olarak artmaktadır. Çünkü N sayısı artıkça GİB üzerinde yapılan işlem sayısı ve yoğunluğu artmaktadır, dolayısıyla işlemin gerçekleşme süresi de artmaktadır.

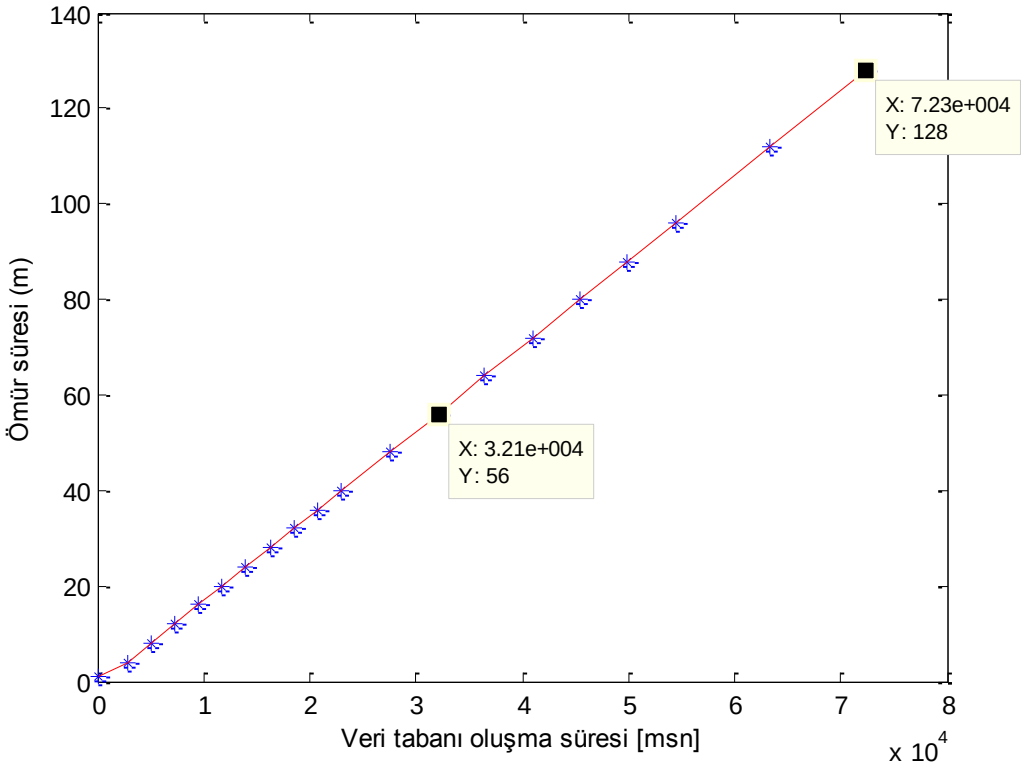
Şekil 4.2'de ise m değeri 128 olduğunda artan N değerlerine göre doğrulama işleminin tamamlanma süresinin değişim grafiği verilmiştir. Bu grafikte verilen süreler en uzun hal içindir, yani aranan bilginin veri tabanında yer almadığı durum içindir ve burada da aynı şekilde doğrulama işleminin tamamlanma süresi ile veri tabanı uzunluğu arasında doğrusala çok yakın bir ilişki vardır.



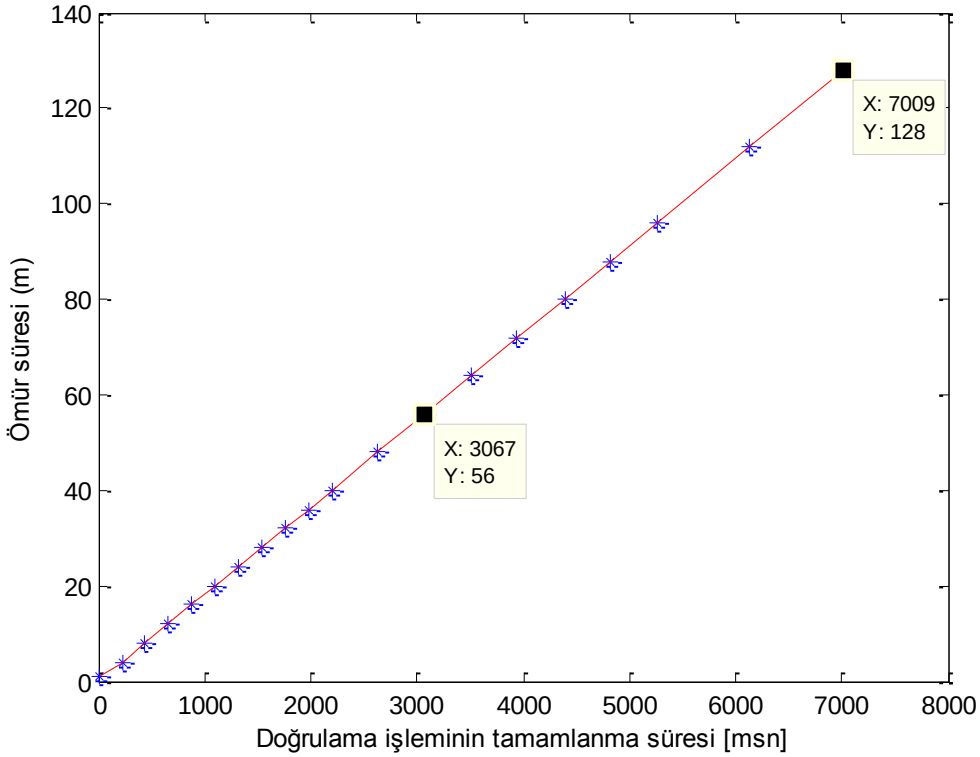
Şekil 4.2 OSKP için Doğrulama işleminin tamamlanma süresi – Veri tabanı uzunluğu grafiği.

Daha sonra N değeri 290000'de sabit tutulup m artırılarak aynı grafikler tekrar oluşturulmuştur. Şekil 4.3'te ve Şekil 4.4'te ömür süresinin değişimiyle veri tabanı oluşma süresinin ve doğrulama işleminin tamamlanma süresinin değişimi grafikleri verilmiştir. Burada da doğrulama işlemi için süreler en uzun hal için, yani aranan bilginin veri tabanında yer almaması durumu için ölçülmüştür. Beklenildiği gibi bu grafiklerde değişim doğrusaldır. Çünkü bir önceki alt başlıkta da anlatıldığı gibi OSKP'nin gerçeklenmesinde veri tabanı oluşturulurken ömür süresi kadar dizi

oluşturulmuştur ve her ömre ait kimlik bilgileri bu dizilerde saklanmıştır. Bu dizilerin her birinin oluşturulması ve her birinde arama işleminin yapılması paralel olmasına rağmen dizilerin ayrı ayrı oluşturulması ve bu dizilerde ayrı ayrı arama işlemi seri olarak yapılmıştır. Dolayısıyla m ile süreler arasındaki ilişki doğrusal olmaktadır.

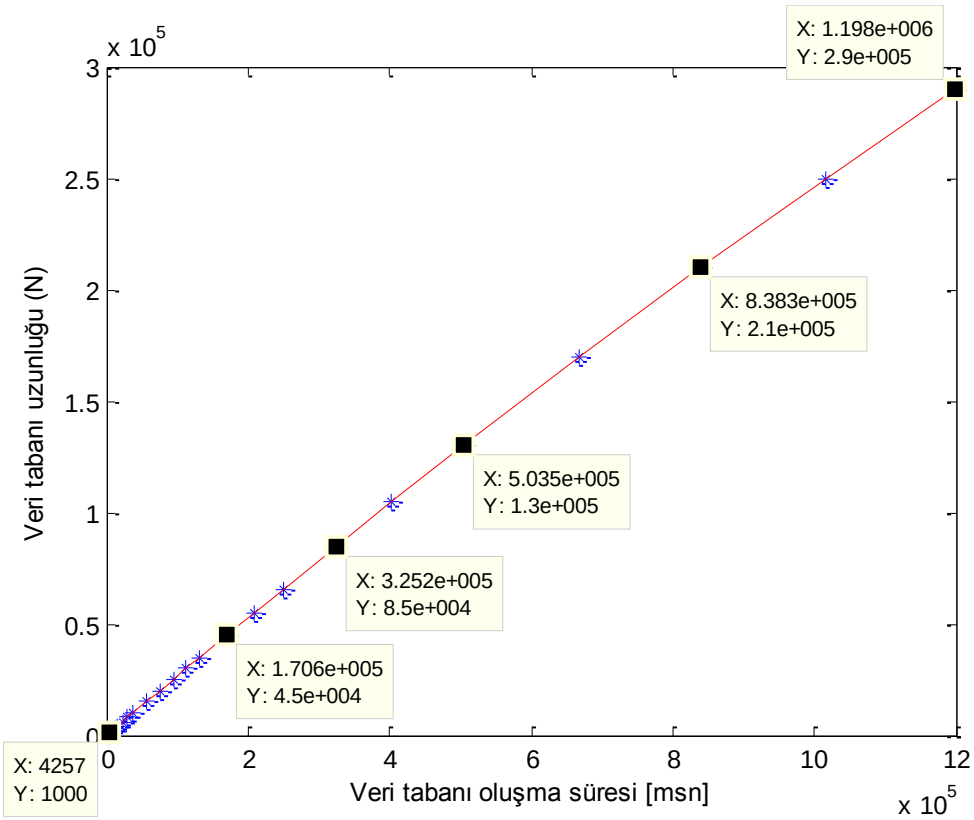


Şekil 4.3 OSKP için Veri tabanı oluşma süresi – Ömür süresi grafiği.

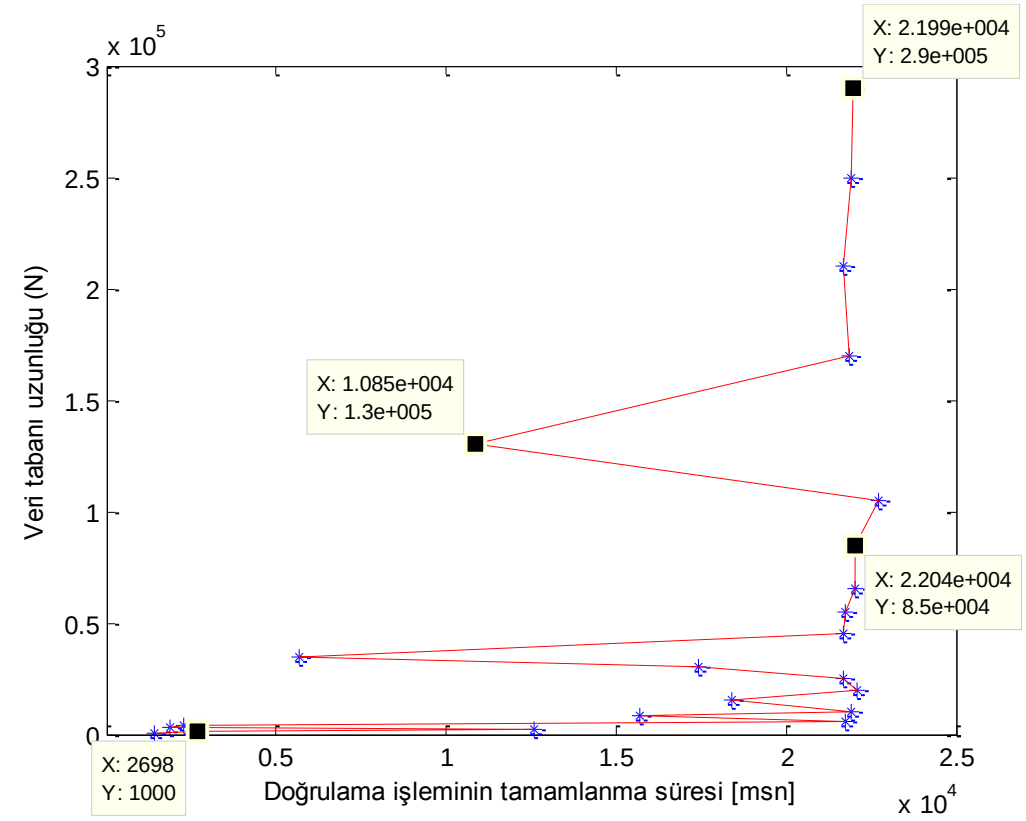


Şekil 4.4 OSKP için Doğrulama işleminin tamamlanma süresi – Ömür süresi grafiği.

Bu grafiklerin elde edilmesinden sonra aynı grafikler OSK/AOP'deki sunucu yapısı için de oluşturulmuştur. Ayrıca bu sunucu yapısı için N'nin maksimum değeri çok daha fazla olabileceği halde bir önceki sunucu yapısıyla karşılaştırabilmek için gene 290000 alınmıştır. Şekil 4.5 ve Şekil 4.6'da m'nin 128, T'nin 54 olduğu durumda değişen N değerleri için veri tabanı oluşma süresi ve doğrulama işlemi tamamlanma süresi değişim grafikleri verilmiştir. Bu sunucu yapısında doğrulama işleminin tamamlanma süresinin en uzun hali ise zincir yapısının en başındaki bir etiket bilgisinin (r_i^1) aranıp yerinin bulunmasıdır. Bu yüzden süre ölçümlerinde ikinci etiket bilgisi (r_2^1) giriş olarak verilmiştir.



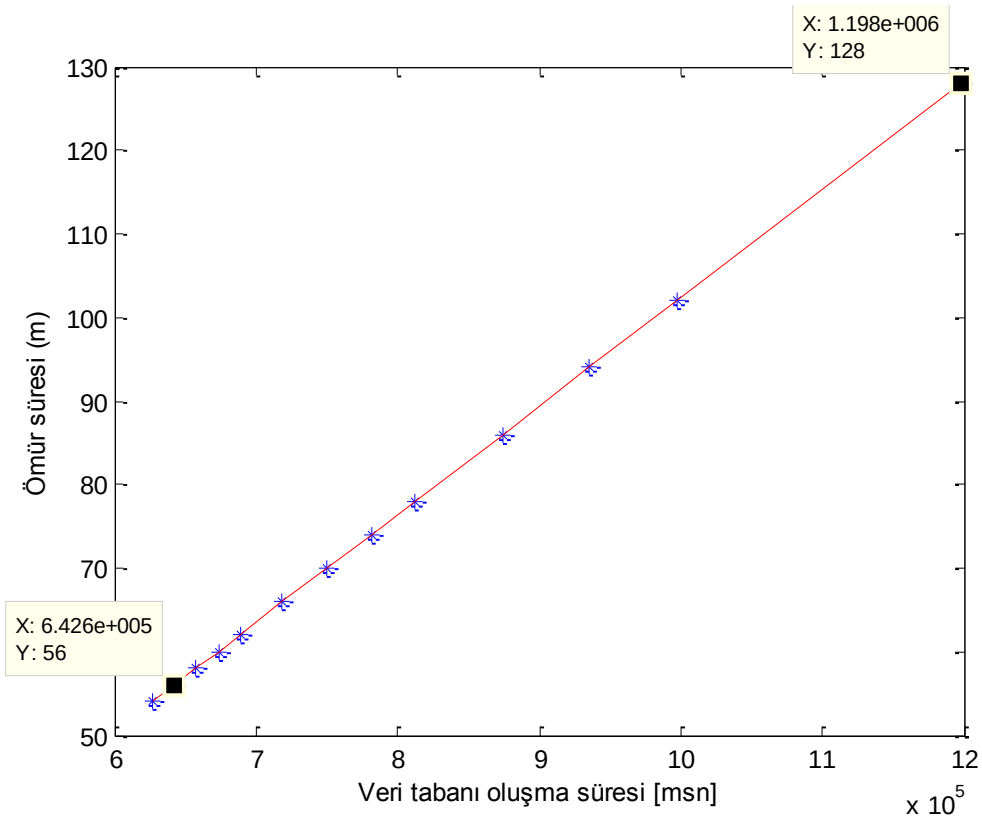
Şekil 4.5 OSK/AOP için Veri tabanı oluşma süresi – Veri tabanı uzunluğu grafiği.



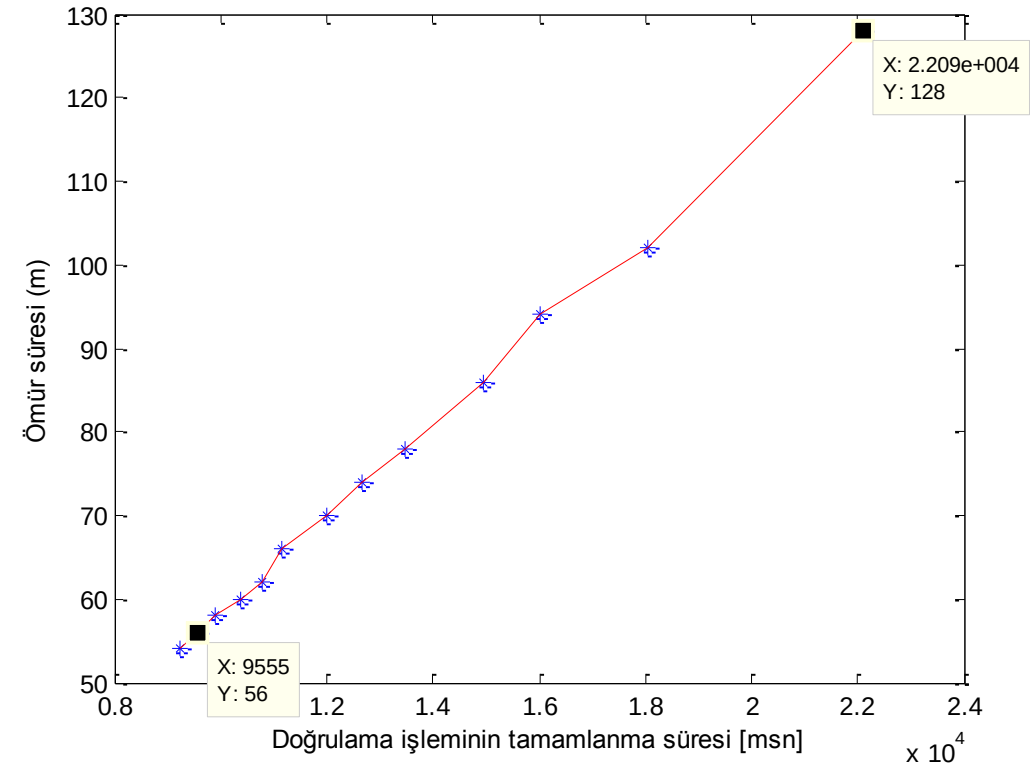
Şekil 4.6 OSK/AOP için Doğrulama işleminin tamamlanma süresi – Veri tabanı uzunluğu grafiği.

Şekil 4.5'teki grafik beklenildiği gibi doğrusaldır. Çünkü OSKP'deki sunucu yapısıyla aynı şekilde burada da N değeri arttıkça GİB üzerindeki işlem sayısı ve yoğunluğu, dolayısıyla işlemin gerçekleşme süre artmaktadır. Şekil 4.6'da ise görüldüğü gibi bazı ani değişimler vardır. Oysa grafikte, sağ tarafındaki gibi birbirine çok yakın ve hafif bir doğrusal değişimin olduğu süreler bekleniyordu. Buradaki ani süre düşmelerinin sebebi ise kullanılan R fonksiyonudur (bkz. Denklem 2.3). Çünkü N değeri değiştikçe aynı girişler için R fonksiyonunun çıkışı farklılaşmaktadır ve daha önce de bahsedildiği gibi farklı girişler için R fonksiyonu mod işleminden dolayı aynı çıkışı verebilmektedir. Dolayısıyla N'nin bazı değerlerinde R fonksiyonu farklı girişler için zincirin farklı yerlerinde aynı çıkışı verebilir, yani aranan etiket bilgisinin zincirin bir noktasındaki çıkışı zincirin bambaşka bir yerinde de yer alır ve onların doğrulanma işlemi daha kısa sürebilir. Şekil 4.6'daki grafiğin sağ tarafındaki başarılı bulmaların sürelerinin değişimin çok hafif olmasının sebebi ise OSKP'deki gibi burada m değeri kadar bu aramanın tekrarlanmamasıdır, yani bu hafif değişimin m değeri kadar katlanmamasıdır.

En son olarak ise N 290000'de, T 54'te sabit tutularak Şekil 4.7 ve Şekil 4.8'de değişen ömür süreleriyle veri tabanı oluşma süresi ve doğrulama tamamlanma süresi değişim grafikleri oluşturulmuştur. Bu şekillerde görüldüğü üzere m değeri arttıkça süreler de artmaktadır. Bu kullanılan R (bkz. Denklem 2.3) ve F (bkz. Denklem 2.1) fonksiyonlarından kaynaklanmaktadır. Çünkü m değeri arttıkça R'nin çıkışında k'nın alabileceği değerler büyüyebilmektedir ve bu da F fonksiyonunda alınan H özet fonksiyonu sayısını artırabilmektedir.



Şekil 4.7 OSK/AOP için Veri tabanı oluşma süresi – Ömür süresi grafiği.



Şekil 4.7 OSK/AOP için Doğrulama işleminin tamamlanma süresi – Ömür süresi. grafiği.

Bu bölümde son olarak Tablo 4.4'te maksimum değerler için, yani N'nin 290000, m'nin 128 ve T'nin 54 olduğu durum için iki ayrı sunucu yapısının veri tabanı oluşturması ve doğrulama işlemini tamamlaması süreleri verilecektir.

Tablo 4.4 İki sunucu yapısının süre açısından karşılaştırılması

	Veri tabanı oluşma süresi [msn]	Doğrulama tamamlama süresi [msn]
OSKP'nin sunucusu	72423,234	7010,745
OSK/AOP'nin sunucusu	1198142,141	21993,004

Tablo 4.4'te görüldüğü gibi OSKP'nin sunucu yapısı gelişmiş hali olan OSK/AOP'de veri tabanı oluşma süresi yaklaşık 16.5 kat, doğrulama tamamlama süresi ise yaklaşık 3.14 kat artmıştır.

5. SONUÇLAR VE TARTIŞMA

Bu çalışmada bir protokolün için iki farklı sunucu yapısı GİB üzerinde gerçeklenmiştir. Bunun için sunucu yapısındaki iki temel işlem, yani veri tabanı oluşturma ve doğrulama işlemleri gerçeklenmiştir. GİB üzerinde yapılan bu gerçeklemelerin paralel programlama yapısına nasıl uyarlanacağı tasarlanmıştır ve ilgili kodlar yazılmıştır.

Daha sonra gerçeklenen bu iki sunucu ile ilgili analizler yapılmıştır. Her iki sunucu yapısı için kullanılan bellek miktarları hesaplanmış. Sonra gene her iki sunucu yapısı için veri tabanı uzunluğunun ve ömür süresinin değişmesiyle ayrı ayrı veri tabanı oluşturma ve doğrulama işlemini tamamlama grafikleri çizdirilmiştir.

Analizler sonucunda görülmüştür ki aynı değerler için OSKP'de veri tabanı oluşturulması ve doğrulama işleminin süresi OSK/AOP'ye göre çok daha kısadır. Ancak bu yapıdaki dezavantaj kullanılan bellek alanının çok fazla olmasıdır. Kullanılan bellek alanının çok fazla olması veri tabanı uzunluğunu kısıtlamaktadır, çünkü GİB'nin kullandığı global belleğin bir sınırı vardır. Daha gelişmiş grafik kartları kullanılarak bu bellek sınırlaması sorunu iyileştirilebilir. OSK/AOP'de ise ileri sürüldüğü gibi kullanılan bellek alanı oldukça düşüktür, ancak süreler çok artmıştır. Ayrıca bu sunucu yapısında kullanılan R fonksiyonundan (bkz. Denklem 2.3) dolayı zaman zaman aranan etiketin yeri yanlış bulunabilmektedir ya da veri tabanında yer almayan bir bilgi arama sonucunda bulunabilmektedir.

KAYNAKLAR

[1] **Ohkubo, M., Suzuki K. ve Kinoshita S.,** 2003. Cryptographic approach to "privacy-friendly" tags, *RFID Privacy Workshop*, MIT, USA.

[2] **Kirk, D.B. ve Hwu, W.W.,** 2010. Programming Massively Parallel Processors. Morgan Kaufmann Publishers, Burlington.

[3] **Avoine, G., Dysli, E. ve Oeschslin P.,** Reducing time complexity in RFID systems, *Selected Areas in Cryptography - SAC 2005*, s. 291-306, Kingston, Canada.

[4] **Avoine, G. ve Oeschslin P.,** 2005. A scalable and provably secure hash-based RFID protocol, *International Workshop on Pervasive Computing and Communication Security - PerSec 2005*, s. 110-114, Kauai Island, Hawaii, USA.

[5] Cryptographic hash function, 20 Mayıs 2011 tarihinde http://en.wikipedia.org/wiki/Cryptographic_hash_function sayfasından alınan

[6] **NVIDIA.,** 2010. CUDA C Programming Guide.

[7] **NVIDIA.,** 2008. CUDA Technical Training, Volume I: Introduction to CUDA Programming.

[8] **Membarth, R.,** 2009. CUDA Parallel Programming Tutorial.

[9] **Sanders, J. ve Kandrot, E.,** 2010. CUDA by Example: An Introduction to General-Purpose GPU Programming. Edwards Brothers, Michigan.

[10] Secure Hash Standard, 20 Mayıs 2011 tarihinde http://www.itl.nist.gov/fipspubs/fip180-1.htm sayfasından alınan

ÖZGEÇMİŞ

Ad Soyad: Kenan ERDOĞAN
Doğum Yeri ve Tarihi: İstanbul, 1988
Lise: Adnan Menderes Anadolu Lisesi 2002-2006
Lisans: İstanbul Teknik Üniversitesi, Elektronik Mühendisliği - 2006